

Technische Hochschule Deggendorf
Fakultät Angewandte Informatik
Studiengang Bachelor Angewandte Informatik

Titel
Thema 3: PXE-Boot Server

Projektarbeit im Studienfach Betriebssysteme
Bachelor Angewandte Informatik (B. AI)
An der Technischen Hochschule Deggendorf

Vorgelegt Von:
Hamid Hekmatnezhad
Matrikelnummer: 22405958
Task: Implementierung

Ahmet Akkurt
Matrikelnummer: 22407067
Task: Sicherheitskonzept

Prüfer:
Rainer Pöschl,
Stefan Kunze

Links:
- [GitHub](#)
- [GitLab \(THD\)](#)

Inhaltsverzeichnis

1.	Einleitung und Zielsetzung	3
1.1	Hintergrund und Motivation	3
1.2	Projektziel	3
1.3	Technische Rahmenbedingungen und Sicherheit	3
1.4	Sicherheitsanforderungen und Einsatzbereiche von PXE-Boot-Servern	4
2.	Systemarchitektur und Dienste	4
2.1	Das hybride Bereitstellungsmodell (Drei-Schichten-Modell)	4
2.1.1	Schicht 1: Adressierung und Steuerung via DHCP (dnsmasq)	5
2.1.2	Schicht 2: Initialer Boot-Transfer via TFTP	6
2.1.3	Schicht 3: Hochgeschwindigkeits-Datenverteilung via HTTP	6
2.1.4	Rationale des hybriden Ansatzes	6
2.2	Netzwerk-Konfiguration und Isolierung	7
2.2.1	Schutz vor produktiven Netzwerkstörungen	7
2.2.2	Adressierungsschema und Erreichbarkeit	8
2.2.3	Funktionskontrolle in der Laborumgebung	8
3.	Verzeichnisstruktur und Ressourcenmanagement	9
3.1	TFTP-Verzeichnisstruktur (/srv/tftp)	9
3.2	Nginx-Web-Struktur und Speichererweiterung (/var/www/html)	10
3.3	Besonderheit: Ressourcenmanagement und Speichererweiterung	10
4.	Automatisierung der Installation	10
4.1	Automatisierungsmethoden im Überblick	11
4.1.1	Das Konzept der Antwortdateien (Answer Files)	11
4.1.2	Klassifizierung der Automatisierungstechnologien nach Distribution	12
4.1.3	Vorteile der zentralen Bereitstellung via Nginx	13
4.2	Automatisierung von Fedora und Rocky Linux (Kickstart)	13
4.2.1	Detaillierte Analyse der Kickstart-Konfiguration (ks.cfg)	13
4.2.2	Integration ins PXE-Boot-Menü	14
4.3	Automatisierung von CachyOS (Arch-basiert)	15
4.3.1	Implementierung der JSON-basierten Automatisierung	16
4.3.2	Integration ins PXE-Boot-Menü	16
4.4	Automatisierung von Debian 11 und Kali Linux (Preseeding)	17
4.4.1	Zentrale Konfigurationsaspekte der preseed.cfg	17
4.4.2	Technische Herausforderungen und spezifische Lösungen	18
4.4.3	Integration in das PXE-Boot-Menü	20
5.	Technische Herausforderungen und Troubleshooting	21
5.1	Herausforderung bei der Hardware-Erkennung (CD-ROM-Bypass)	21
5.2	Konsistenz zwischen Kernel und Repository	22
5.3	Speicherplatzmangel und Erweiterung der Festplattenkapazität	22
5.4	Optimierung der Installationsmedien (Netinstall vs. Full-ISO)	23

6.	Absicherung und Einsatzbereiche des PXE-Boot-Servers	24
6.1	Absicherung des PXE-Boot-Servers	24
6.1.1	Bedrohungsmodell und Sicherheitsrisiken	24
6.1.2	Netzwerkisolation des PXE-Servers	24
6.1.3	Absicherung der DHCP- und TFTP-Dienste	25
6.1.4	Schutz der Boot-Images und Installationsdateien	26
6.1.5	Secure Boot und Vertrauenskette	26
6.2	Einsatzbereiche von PXE-Boot	26
6.2.1	Automatisierte Betriebssystembereitstellung	26
6.2.2	Nutzung in Unternehmens- und Rechenzentrumsumgebungen	27
6.2.3	Einsatz in Schulungs-, Test- und Laborumgebungen	27
6.2.4	Unterstützung von Diskless- bzw. Thin-Client-Umgebungen	27
6.3	Vergleich: Traditionelles PXE (TFTP) vs. Modernes HTTP-Boot	28
7.	Fazit und Ausblick	29
7.1	Zusammenfassung der Ergebnisse	29
7.2	Reflexion und Mehrwert	29
7.3	Ausblick	29
	Literaturverzeichnis	30
	Anhang	31

1. Einleitung und Zielsetzung

1.1 Hintergrund und Motivation

In modernen IT-Infrastrukturen ist die effiziente und konsistente Bereitstellung von Betriebssystemen eine der grundlegenden Aufgaben der Systemadministration. Die manuelle Installation von Betriebssystemen über physische Medien wie USB-Sticks oder DVDs ist bei einer größeren Anzahl von Zielsystemen zeitaufwendig, fehleranfällig und schwer skalierbar.

Um diese Herausforderungen zu bewältigen, wird in diesem Projekt ein **PXE-Boot-Server** (Preboot Execution Environment) implementiert. PXE ermöglicht es Computern, ein Betriebssystem direkt über das Netzwerk zu booten und zu installieren, ohne dass lokale Speichermedien erforderlich sind. Dies ist besonders in Rechenzentren, Testlaboren und großen Unternehmensnetzwerken der Standardweg zur Systemverteilung.

1.2 Projektziel

Das Hauptziel dieser Arbeit ist der Aufbau eines funktionsfähigen Boot-Servers als virtuelle Maschine, der eine automatisierte Installation von fünf verschiedenen Linux-Distributionen ermöglicht:

- **Fedora**
- **Rocky Linux**
- **CachyOS**
- **Debian**
- **Kali Linux**

Ein wesentlicher Schwerpunkt liegt dabei auf der **Automatisierung des Installers**. Das System soll so konfiguriert werden, dass nach dem Start des Boot-Vorgangs keine weiteren manuellen Eingriffe ("Zero-Touch-Installation") erforderlich sind. Dies wird durch den Einsatz von Konfigurationsdateien wie **Preseed** (für Debian/Kali), **Kickstart** (für Fedora/Rocky) und **json File** (für CachyOS) realisiert.

1.3 Technische Rahmenbedingungen und Sicherheit

Gemäß den Projektvorgaben wird der Server in einem **abgeschotteten Netzwerk** (Isolated Network) betrieben. Dies ist eine kritische Sicherheitsmaßnahme, um sicherzustellen, dass der integrierte DHCP-Server nicht mit dem produktiven Unternehmensnetzwerk kollidiert oder unbefugte Kommunikation nach außen stattfindet.

Zusätzlich zur praktischen Implementierung befasst sich dieser Bericht mit der Analyse von Sicherheitsaspekten des Boot-Servers sowie dem Vergleich zwischen dem klassischen PXE-Boot (basiert auf TFTP) und der moderneren Alternative, dem **HTTP-Boot**.

1.4 Sicherheitsanforderungen und Einsatzbereiche von PXE-Boot-Servern

Der erfolgreiche Einsatz eines PXE-Boot-Servers zur automatisierten Betriebssysteminstallation setzt nicht nur eine funktionale Implementierung voraus, sondern erfordert auch eine sorgfältige Betrachtung sicherheitsrelevanter Aspekte. Insbesondere durch die Nutzung zentraler Netzwerkdienste wie DHCP und TFTP entsteht ein erhöhtes Gefährdungspotenzial, da Fehlkonfigurationen oder unzureichende Schutzmaßnahmen weitreichende Auswirkungen auf das gesamte Netzwerk haben können.

Während die technische Umsetzung und Automatisierung des PXE-Boot-Prozesses die Effizienz der Systembereitstellung maßgeblich steigert, rückt mit zunehmender Verbreitung dieser Technologie auch die Absicherung der Infrastruktur in den Fokus. Ein ungeschützter PXE-Boot-Server kann von unautorisierten Clients missbraucht oder zur Verteilung manipulierter Betriebssystem-Images genutzt werden.

Vor diesem Hintergrund widmet sich dieser Abschnitt der Analyse zentraler Sicherheitsrisiken beim Betrieb eines PXE-Boot-Servers und stellt geeignete Maßnahmen zur Absicherung der verwendeten Dienste und der Netzwerkinfrastruktur vor. Ergänzend werden typische Einsatzbereiche von PXE-Boot betrachtet, um den praktischen Nutzen der Technologie einzuordnen und gleichzeitig deren Grenzen und Risiken aufzuzeigen.

2. Systemarchitektur und Dienste

Die technische Infrastruktur des PXE-Boot-Servers wurde so konzipiert, dass sie maximale Stabilität in einer isolierten Umgebung bietet. Dabei wurde ein hybrider Ansatz gewählt, der verschiedene Netzwerkprotokolle kombiniert, um die Stärken der jeweiligen Dienste optimal zu nutzen.

2.1 Das hybride Bereitstellungsmodell (Drei-Schichten-Modell)

In diesem Projekt wurde eine hybride Architektur implementiert, um die Effizienz, Geschwindigkeit und Zuverlässigkeit der Betriebssystemverteilung zu optimieren. Da kein einzelnes Protokoll alle Anforderungen an einen modernen Netzwerk-Boot-Prozess erfüllt, basiert dieser Ansatz auf der synergetischen Zusammenarbeit von drei technologischen Schichten: DHCP, TFTP und HTTP.

2.1.1 Schicht 1: Adressierung und Steuerung via DHCP (dnsmasq)

Die erste Phase des PXE-Prozesses beginnt mit dem **Dynamic Host Configuration Protocol (DHCP)**. Der Server fungiert hierbei als Wegweiser für den Client und liefert die grundlegenden Netzwerkparameter:

- **IP-Adresszuweisung:** Der Server weist dem Client eine gültige IP-Adresse innerhalb des isolierten Subnetzes zu, um die Kommunikation im Netzwerk zu ermöglichen. (192.168.1.50 - 192.168.1.100)
- **Next-Server (Option 66):** Diese Option informiert den Client über die IP-Adresse des Servers, von dem die weiteren Boot-Dateien bezogen werden können (in diesem Fall der PXE-Server selbst).
- **Boot-Filename (Option 67):** Hier wird der Pfad zur Bootloader-Datei (z. B. `pxelinux.0`) definiert. Ohne diese expliziten Parameter würde der PXE-Vorgang nach der IP-Zuweisung abbrechen.

Config DHCP:

PATH: `/etc/dnsmasq.conf`

```
# deactivate DNS
port=0

# interface Listen to ...
interface=enp0s3

# DHCP IP range 50 - 100
dhcp-range=192.168.1.50,192.168.1.100,255.255.255.0,12h

# pxe Loader
dhcp-boot=pxelinux.0

# activate TFTP
enable-tftp
tftp-root=/srv/tftp
```

2.1.2 Schicht 2: Initialer Boot-Transfer via TFTP

Sobald der Client seine Anweisungen vom DHCP-Server erhalten hat, nutzt er das **Trivial File Transfer Protocol (TFTP)** für die initiale Startphase.

- **Leichtgewichtiges Bootstrapping:** TFTP wird verwendet, um den Bootloader sowie die absolut notwendigen Kerndateien – den Linux-Kernel (`vmlinuz`) und die Initial RAM Disk (`initrd.gz`) – zu übertragen.
- **Limitierungen von TFTP:** Da TFTP auf UDP basiert und standardmäßig sehr kleine Datenblöcke verwendet, ist es für die Übertragung großer Datenmengen (wie komplette ISO-Images) extrem ineffizient und langsam. Daher dient es in diesem Aufbau lediglich als "Brücke", um den eigentlichen Installer in den Arbeitsspeicher des Clients zu laden.

2.1.3 Schicht 3: Hochgeschwindigkeits-Datenverteilung via HTTP (Nginx)

Um die systembedingten Leistungsengpässe von TFTP zu überwinden, wird für die datenintensive Installationsphase der leistungsstarke **Nginx-Webserver** als HTTP-Backend eingesetzt. Im Gegensatz zu TFTP, das auf dem verbindungslosen UDP basiert, nutzt HTTP das TCP-Protokoll, was eine deutlich höhere Stabilität und Geschwindigkeit bei der Übertragung großer Datenmengen ermöglicht.

- **Repository-Hosting:** Nginx stellt die extrahierten Inhalte der Betriebssystem-ISOs (z. B. die Verzeichnisse `pool` und `dists` bei Debian) als lokales Netzwerk-Repository bereit. Dies ermöglicht dem Installer den Zugriff auf tausende Pakete mit der vollen Bandbreite des Netzwerks, was die Installationszeit im Vergleich zu physischen Medien oder TFTP-basierten Übertragungen erheblich verkürzt.
- **Automatisierung der Installation:** Die Konfigurationsdateien für die automatisierte Installation ("`preseed.cfg`" für Debian/Kali oder "`ks.cfg`" für Fedora/Rocky) werden ebenfalls über den Nginx-Webserver zentral ausgeliefert. Über den Boot-Parameter "`url=http://192.168.1.10/...`" erhält der Installer alle notwendigen Anweisungen für die Partitionierung, Benutzererstellung und Paketauswahl direkt vom Server. Dadurch wird eine vollständige "Zero-Touch-Installation" ohne manuelle Interaktion realisiert.

Durch den Einsatz von Nginx als zentralem Verteilungspunkt wird sichergestellt, dass der PXE-Server auch bei der gleichzeitigen Bereitstellung auf mehreren Zielsystemen eine hohe Performance und Zuverlässigkeit beibehält.

2.1.4 Rationale des hybriden Ansatzes

Die Entscheidung für diesen hybriden Aufbau resultiert aus der Notwendigkeit, die universelle Kompatibilität der PXE-Spezifikation mit den Anforderungen an eine moderne, schnelle und automatisierte Softwareverteilung zu vereinen. Die logische Trennung der Aufgaben bietet folgende Vorteile:

- **Überwindung technologischer Barrieren:** Während DHCP und TFTP aufgrund ihrer tiefen Integration in die Netzwerk-Firmware (NIC) unersetzlich für die Initialisierung der Hardware sind, stoßen sie bei der Übertragung moderner Betriebssystem-Images an ihre Leistungsgrenzen. Durch den Wechsel zu HTTP (Nginx) nach dem Laden des Kernels wird dieser Flaschenhals effektiv umgangen.
- **Stabilität durch Protokollwechsel:** Der Übergang vom verbindungslosen UDP (TFTP) zum verbindungsorientierten TCP (HTTP via Nginx) stellt sicher, dass auch bei schlechten Netzwerkbedingungen oder gleichzeitigen Installationen auf mehreren Clients keine Datenpakete verloren gehen.
- **Zentralisierte Automatisierung:** Die Nutzung von HTTP als Träger für Automatisierungsskripte (`preseed.cfg` und `ks.cfg`) ermöglicht eine dynamische und zentrale Verwaltung der Konfigurationen. Dies ist die Grundvoraussetzung für

die realisierte "Zero-Touch"-Installation, da der Installer die Parameter direkt per URL-Abruf ohne manuelle Bestätigung verarbeiten kann.

- **Ressourceneffizienz im isolierten Netzwerk:** Durch die Bündelung von DHCP und TFTP in **dnsmasq** und die Nutzung von Nginx für den Datentransfer wurde ein schlankes System geschaffen, das trotz minimalem Ressourcenverbrauch eine maximale Performance bei der Bereitstellung der fünf verschiedenen Linux-Distributionen bietet.

Zusammenfassend lässt sich sagen, dass dieser hybride Ansatz die breite Hardware-Unterstützung klassischer PXE-Umgebungen mit der Skalierbarkeit und Flexibilität moderner Web-Technologien kombiniert. Dies führt nicht nur zu einer signifikanten Zeitsparnis, sondern garantiert eine konsistente und fehlerfreie Bereitstellung über alle gewählten Distributionen hinweg.

2.2 Netzwerk-Konfiguration und Isolierung

Ein kritischer Aspekt bei der Bereitstellung eines PXE-Boot-Servers ist die Konfiguration der Netzwerkumgebung. Gemäß den Projektanforderungen wurde das System in einem vollständig **abgeschotteten Netzwerk** (Isolated Network) realisiert.

2.2.1 Schutz vor produktiven Netzwerkstörungen (Vermeidung von Rogue DHCP)

Ein zentraler Aspekt bei der Bereitstellung eines PXE-Boot-Servers ist die strikte Trennung vom produktiven Firmen- oder Heimnetzwerk. Da der eingesetzte Dienst **dnsmasq** als DHCP-Server agiert, sendet er Broadcast-Pakete aus, um auf Anfragen von Clients zu reagieren.

Wäre der Server mit einem physischen Netzwerk verbunden, in dem bereits ein autorisierter DHCP-Server (z. B. ein Router oder ein Windows-Server-Domain-Controller) existiert, käme es zu einem **DHCP-Konflikt**. In einem solchen Szenario würde der PXE-Server als sogenannter „**Rogue DHCP Server**“ fungieren. Dies hätte zur Folge, dass unbeteiligte Rechner im Netzwerk fälschlicherweise IP-Adressen und PXE-Boot-Parameter von unserem Server beziehen könnten, was zu massiven Verbindungsstörungen und unvorhersehbaren Boot-Vorgängen im gesamten Netzwerk führen würde.

Um diese produktiven Störungen technisch auszuschließen, wurde in der Virtualisierungsumgebung für die Netzwerkschnittstelle **enp0s3** der Modus „**Internes Netzwerk**“ (Internal Network) gewählt. Diese Konfiguration stellt sicher, dass:

- DHCP-Broadcasts physikalisch auf die virtuellen Maschinen innerhalb des Hosts begrenzt bleiben.
- Keine Kommunikation mit dem Gateway des Host-Systems oder dem Internet stattfindet.

- Eine kontrollierte Testumgebung geschaffen wird, in der nur die Ziel-VMs auf die PXE-Angebote reagieren können.

2.2.2 Adressierungsschema und Erreichbarkeit

Für eine stabile Kommunikation innerhalb der isolierten Umgebung wurde ein statisches Adressierungsschema gewählt. Dies ist essenziell, da sowohl der TFTP-Dienst als auch der HTTP-Server (Nginx) unter einer festen IP-Adresse erreichbar sein müssen, damit die PXE-Clients die Boot-Dateien und Repositories zuverlässig finden können.

Das Netzwerk wurde nach folgenden Parametern konfiguriert:

- **Netzwerkadresse:** 192.168.1.0 / 24
- **Subnetzmaske:** 255.255.255.0
- **Server-IP (Statisch):** 192.168.1.10
 - Diese IP dient als Ankerpunkt für alle Dienste. Sie wurde in der Netzwerkkonfiguration des Host-Systems fest zugewiesen, um sicherzustellen, dass die DHCP-Option 66 (`next-server`) immer auf das korrekte Ziel zeigt.
- **DHCP-Pool (Dynamisch):** 192.168.1.50 bis 192.168.1.100
 - Dieser Bereich wurde in der `dnsmasq.conf` über die Direktive `dhcp-range` definiert. Er bietet ausreichend Platz für die gleichzeitige Installation mehrerer Test-Clients.
- **Leasing-Zeit:** 12h (Stunden)
 - Die gewählte Lease-Time stellt sicher, dass ein Client während des gesamten Installationsprozesses dieselbe IP-Adresse behält, was Protokollfehler während des Wechsels von TFTP zu HTTP verhindert.

Da es sich um ein abgeschottetes Labornetzwerk handelt, wurde bewusst auf die Konfiguration eines **Standard-Gateways** und eines **externen DNS-Servers** verzichtet. Dies unterstreicht den "Air-Gapped"-Charakter der Umgebung und stellt sicher, dass sämtlicher Datenverkehr ausschließlich über den eigenen PXE-Server abgewickelt wird.

2.2.3 Funktionskontrolle in der Laborumgebung

Die Entscheidung für eine isolierte Netzwerkumgebung basiert primär auf der Notwendigkeit einer kontrollierten Testumgebung. Da die Installation von fünf verschiedenen Betriebssystemen (Fedora, Rocky, CachyOS, Debian, Kali) eine hohe Anzahl an DHCP-Leases und Dateitransfers erfordert, bietet die Isolation folgende funktionale Vorteile für die Implementierung:

- **Reproduzierbarkeit:** Durch die Abschottung von externen Einflüssen ist gewährleistet, dass jeder Boot-Vorgang unter identischen Bedingungen stattfindet. Dies ist essenziell für das Debugging der PXE-Konfigurationen und der Automatisierungsskripte.
- **Vermeidung von Fehlkonfigurationen:** In einer geschlossenen Umgebung kann der PXE-Server (`dnsmasq`) exklusiv auf die Anfragen der Test-Clients reagieren. Es

wird verhindert, dass Clients versehentlich falsche Boot-Parameter von anderen, im Host-Netzwerk aktiven DHCP-Servern erhalten.

- **Optimierte Bandbreite:** Da der gesamte Datenverkehr zwischen dem Nginx-Server und den Clients auf das interne virtuelle Segment beschränkt ist, steht für die Übertragung der ISO-Images die maximale Bandbreite der virtuellen Schnittstelle zur Verfügung, ohne das physische Netzwerk zu belasten.

Zusammenfassend dient die Isolation des Netzwerks als rein technische Maßnahme, um eine stabile und störungsfreie Infrastruktur für die Bereitstellung der Betriebssysteme sicherzustellen.

3. Verzeichnisstruktur und Ressourcenmanagement

Eine konsistente Verzeichnisstruktur ist entscheidend für die Stabilität des PXE-Servers. Dabei wird strikt zwischen dem Boot-Bereich (TFTP) und dem Daten-Bereich (Nginx/HTTP) getrennt.

3.1 TFTP-Verzeichnisstruktur (</srv/tftp>)

Das Verzeichnis </srv/tftp> enthält ausschließlich die Dateien, die für die erste Phase des Bootvorgangs über das UDP-Protokoll notwendig sind. Hierzu gehören der Bootloader, das PXE-Menü sowie die Kernels und Initrds der verschiedenen Distributionen.

```
/srv/tftp/
└── os-images
    ├── cachyos
    │   ├── initramfs-linux-cachyos-lts.img
    │   └── vmlinuz-linux-cachyos-lts
    ├── debian
    │   ├── initrd.gz
    │   └── linux
    ├── fedora
    │   ├── initrd.img
    │   └── vmlinuz
    ├── kali
    │   ├── initrd.gz
    │   └── vmlinuz
    └── rocky
        ├── initrd.img
        └── vmlinuz
└── pxelinux.0
    └── pxelinux.cfg
        └── default
```

3.2 Nginx-Web-Struktur und Speichererweiterung (/var/www/html)

Das Web-Verzeichnis dient als zentrales Repository für die Antwortdateien (Preseed/Kickstart/JSON) und die entpackten Betriebssystem-Images.

```
/var/www/html/
├── cachyos_config.json
├── debian_preseed.cfg
└── extract
    ├── storage
    │   ├── cachyos | debian | fedora | kali | rocky
    ├── fedora_ks.cfg
    ├── kali_preseed.cfg
    ├── os-data
    │   └── iso (Original ISO-Dateien)
    └── rocky_ks.cfg
```

3.3 Besonderheit: Ressourcenmanagement und Speichererweiterung

Im Verlauf des Projekts stieß das System aufgrund der großen ISO-Dateien (insgesamt fünf verschiedenen Distributionen) an die Kapazitätsgrenze der Root-Partition.

Um einen Systemstillstand zu verhindern und eine saubere Trennung von System- und Nutzdaten zu gewährleisten, wurde eine **zusätzliche virtuelle Festplatte mit 150 GB** in den Server integriert. Diese wurde permanent unter dem Pfad </var/www/html/extract/storage> eingehängt (Mount-Point). Alle entpackten Betriebssystem-Dateien sowie die ISO-Images wurden in diesen Bereich verschoben. Diese Maßnahme stellt sicher, dass die System-Partition nicht durch das Wachstum der Installations-Repositories beeinträchtigt wird.

4. Automatisierung der Installation

Das Kernziel dieses Projekts ist die Realisierung einer vollständig automatisierten Installation (Unattended Installation), bei der nach dem ersten Boot-Vorgang keine manuellen Eingaben durch den Administrator mehr erforderlich sind. Dieses Konzept wird durch die Bereitstellung von sogenannten „Antwortdateien“ (Answer Files) umgesetzt, die alle während der Installation gestellten Fragen – von der Partitionierung bis hin zur Benutzererstellung – vorab beantworten.

4.1 Automatisierungsmethoden im Überblick

Das Ziel der „Zero-Touch-Installation“ besteht darin, den gesamten Prozess von der Hardware-Initialisierung bis zum fertigen Betriebssystem ohne eine einzige manuelle Eingabe des Administrators zu durchlaufen. Um dies zu erreichen, wird das Konzept der **unbeaufsichtigten Installation (Unattended Installation)** angewendet.

4.1.1 Das Konzept der Antwortdateien (Answer Files)

Der herkömmliche Installationsprozess eines Linux-Betriebssystems ist interaktiv gestaltet. Der Installer (z. B. der Debian-Installer oder Anaconda) pausiert an verschiedenen Stellen, um notwendige Konfigurationen vom Benutzer abzufragen. Um das Ziel einer „Zero-Touch-Installation“ zu erreichen, wird dieses interaktive Verfahren durch den Einsatz von **Antwortdateien** (Answer Files) ersetzt.

Funktionsprinzip

Eine Antwortdatei ist eine einfache Textdatei, die vordefinierte Werte für alle Fragen des Installers enthält. Anstatt auf eine Tastatureingabe zu warten, liest der Installer die benötigten Informationen direkt aus dieser Datei aus. In dieser Implementierung werden je nach Distribution die Formate **Preseed** (`.cfg`) oder **Kickstart** (`.ks`) verwendet.

Der automatisierte Ablauf im Detail:

1. **Übergabe der Boot-Parameter:** Beim Starten eines Betriebssystems über das PXE-Menü übergibt der Bootloader spezifische Kernel-Parameter an den Installer. Ein entscheidender Parameter ist hierbei die Anweisung, wo die Antwortdatei zu finden ist (z. B. `url=http://192.168.1.10/preseed.cfg` oder `inst.ks=http://192.168.1.10/ks.cfg`).
2. **Initialisierung des Netzwerks:** Der Installer lädt zunächst die minimal notwendigen Treiber und konfiguriert die Netzwerkschnittstelle über DHCP (dnsmasq), um eine Verbindung zum HTTP-Server herzustellen.
3. **Abruf via HTTP:** Sobald die Netzwerkverbindung steht, lädt der Installer die Antwortdatei vom **Nginx-Webserver** herunter.
4. **Automatisierte Ausführung:** Der Installer verarbeitet die Datei sequenziell. Alle Schritte, einschließlich der Festplattenpartitionierung, der Auswahl der Softwarepakete, der Einrichtung von Benutzerkonten und der Zeitzonkonfiguration, werden ohne menschliches Mitwirken durchgeführt.

Bedeutung für das Projekt

Durch dieses Konzept wird sichergestellt, dass die Installationen nicht nur schneller, sondern auch **standardisiert und reproduzierbar** sind. Menschliche Fehler bei der Eingabe von Konfigurationsdaten werden ausgeschlossen, was besonders bei der Bereitstellung von fünf verschiedenen Distributionen in einer Testumgebung von großem Vorteil ist.

4.1.2 Klassifizierung der Automatisierungstechnologien nach Distribution

Da die im Projekt verwendeten Linux-Distributionen auf unterschiedlichen Installer-Frameworks basieren, ist eine differenzierte Herangehensweise bei der Automatisierung erforderlich. Jede Distribution hat ihre eigene Methode, um die Antwortdateien zu verarbeiten.

In der folgenden Tabelle sind die eingesetzten Technologien für die fünf Distributionen zusammengefasst:

Distribution	Installer-Framework	Automatisierungs methode	Dateiformat
Fedora	Anaconda	Kickstart	<code>ks.cfg</code>
Rocky Linux	Anaconda	Kickstart	<code>ks.cfg</code>
CachyOS	Archinstall / Scripts	Scripting	<code>.sh / .json</code>
Debian 11	Debian-Installer	Preseeding	<code>preseed.cfg</code>
Kali Linux	Debian-Installer	Preseeding	<code>preseed.cfg</code>

Kurzbeschreibung der Frameworks:

- **Kickstart (Fedora/Rocky):** Kickstart ist der Standard für Red Hat-basierte Systeme. Die Datei `ks.cfg` ist logisch in Sektionen unterteilt (z. B. Partitionierung, Paketwahl, Post-Installation-Skripte), was eine sehr strukturierte Konfiguration des Anaconda-Installers ermöglicht.
- **Automated Scripting (CachyOS/Arch):** Da Arch-basierte Systeme wie CachyOS oft keinen klassischen grafischen Installer im herkömmlichen Sinne verwenden, basiert die Automatisierung hier auf Skripten, die die Festplatte vorbereiten und das Basissystem via Netzwerk-Repositories installieren.* **Preseeding (Debian/Kali):** Diese Methode injiziert Antworten direkt in die Datenbank des Debian-Installers (`debconf`). Sie ist sehr mächtig, da sie nahezu jede Frage, die während der Installation auftaucht, vorab beantworten kann.
- **Preseeding (Debian/Kali):** Diese Methode injiziert Antworten direkt in die Datenbank des Debian-Installers (`debconf`). Sie ist sehr mächtig, da sie nahezu jede Frage, die während der Installation auftaucht, vorab beantworten kann.

4.1.3 Vorteile der zentralen Bereitstellung via Nginx

Die Entscheidung, die Antwortdateien (`preseed.cfg`, `ks.cfg`) nicht lokal in die Boot-Images (`initrd`) zu integrieren, sondern über den **Nginx-Webserver** bereitzustellen, bietet wesentliche infrastrukturelle Vorteile für die Automatisierung:

- **Zentrale Wartbarkeit:** Änderungen an der Installationskonfiguration – wie das Hinzufügen von Softwarepaketen oder das Ändern von Benutzerpasswörtern – können direkt auf dem Server vorgenommen werden. Es ist nicht notwendig, die TFTP-Boot-Dateien (`initrd`) zu entpacken, zu modifizieren und neu zu erstellen.
- **Stabilität durch Protokollwechsel:** Während TFTP auf dem verbindungslosen UDP basiert, nutzt HTTP (Nginx) das verbindungsorientierte TCP. Dies garantiert, dass die oft kritischen Konfigurationsdateien fehlerfrei und mit hoher Geschwindigkeit beim Installer ankommen.
- **Flexibilität und Skalierbarkeit:** Ein einziger Nginx-Server kann problemlos verschiedene Konfigurationen für unterschiedliche Client-Gruppen verwalten. Über die URL-Parameter im PXE-Menü kann präzise gesteuert werden, welche VM welche Konfiguration erhält.
- **Vermeidung von Hardware-Erkennungsproblemen:** Da der Installer die Datei direkt über das Netzwerk via HTTP bezieht, entfällt die oft fehleranfällige Suche nach lokalen Speichermedien oder virtuellen CD-ROM-Laufwerken innerhalb der VM.

Zusammenfassend führt die Kombination aus PXE-Boot (für den Start) und HTTP-Bereitstellung (für die Konfiguration) zu einer hochflexiblen, wartungsfreundlichen und professionellen Installationsumgebung.

4.2 Automatisierung von Fedora und Rocky Linux (Kickstart)

Für die Distributionen Fedora und Rocky Linux wird das **Kickstart-Verfahren** eingesetzt. Diese Systeme nutzen den **Anaconda-Installer**, der eine Konfigurationsdatei im `.ks`-Format (Kickstart) einliest, um alle Installationsschritte ohne Benutzerinteraktion durchzuführen.

4.2.1 Detaillierte Analyse der Kickstart-Konfiguration (ks.cfg)

Für Fedora und Rocky Linux wurde eine Kickstart-Datei entwickelt, die den **Anaconda-Installer** steuert. Im Vergleich zum Preseeding ist die Kickstart-Syntax in funktionale Sektionen unterteilt, was eine präzise Kontrolle über den Installationsverlauf ermöglicht.

Die implementierte Konfiguration umfasst folgende Kernaspekte:

A. Installationsmodus und Quellverzeichnis

- **text:** Um die Installationsgeschwindigkeit zu maximieren und Ressourcen in der virtuellen Umgebung zu schonen, wurde der Textmodus anstelle der grafischen Oberfläche gewählt.
- **url --url="...":** Dies ist der zentrale Ankerpunkt der Automatisierung. Der Installer bezieht alle Pakete direkt über das HTTP-Protokoll vom lokalen **Nginx-Server (192.168.1.10)**, was die Unabhängigkeit vom Internet im isolierten Netzwerk garantiert.

B. Netzwerk- und Lokalisierungsparameter

- **Netzwerk:** Über `network --bootproto=dhcp --activate --noipv6` wird sichergestellt, dass das System sofort nach dem Booten eine IP vom **dnsmasq-Server** erhält. Die Deaktivierung von IPv6 verhindert Verzögerungen bei der Namensauflösung im isolierten Testnetzwerk.
- **Sprache und Zeit:** `lang`, `keyboard` und `timezone` (Europe/Berlin) werden fest vorgegeben, um interaktive Abfragen zu eliminieren.

C. Vollautomatisierte Datenträgerverwaltung

Die Partitionierung erfolgt nach dem "Zero-Touch"-Prinzip:

- **zerombr & clearpart --all --initlabel:** Diese Befehle löschen alle vorhandenen Partitionstabellen und initialisieren die Festplatte neu, ohne dass der Benutzer dies bestätigen muss.
- **autopart --type=plain:** Erzeugt automatisch ein Standard-Layout ohne LVM-Komplexität, was für die Laborumgebung ideal ist.

D. Paketauswahl und Post-Installation

Die Sektion `%packages` definiert den Softwareumfang:

- **@core:** Installiert nur die absolut notwendigen Systemkomponenten.
- **openssh-server:** Wird explizit hinzugefügt, um den sofortigen Fernzugriff nach der Installation zu ermöglichen.
- **-brltty:** Unnötige Dienste (wie Braille-Unterstützung) werden explizit ausgeschlossen, um das System schlank zu halten.
- **reboot:** Weist das System an, nach erfolgreichem Abschluss sofort neu zu starten, womit der automatisierte Zyklus endet.

4.2.2 Integration ins PXE-Boot-Menü

Um die automatisierte Installation von Fedora einzuleiten, wurde ein spezifischer Eintrag in der PXE-Konfigurationsdatei erstellt. Da Fedora den **Anaconda-Installer** nutzt, unterscheidet sich die Syntax der Boot-Parameter deutlich von Debian-basierten Systemen.

Der konfigurierte Eintrag für Fedora sieht wie folgt aus:

```
LABEL fedora
    MENU LABEL 3) Install Fedora
    KERNEL /os-images/fedora/vmlinuz
    APPEND initrd=/os-images/fedora/initrd.img
inst.repo=http://192.168.1.10/extract/storage/fedora
inst.ks=http://192.168.1.10/fedora_ks.cfg
```

```
LABEL rocky
    MENU LABEL 4) Install Rocky Linux
    KERNEL /os-images/rocky/vmlinuz
    APPEND initrd=/os-images/rocky/initrd.img
inst.repo=http://192.168.1.10/extract/storage/rocky
inst.ks=http://192.168.1.10/rocky_ks.cfg
```

Analyse der spezifischen Kernel-Parameter für Fedora:

- **inst.repo=http://192.168.1.10/extract/storage/fedora:** Dieser Parameter ist für den Betrieb in einem isolierten Netzwerk essenziell. Er teilt dem Installer bereits beim Start mit, wo sich das vollständige Installations-Repository befindet. Ohne diesen Parameter würde Fedora versuchen, öffentliche Mirrors im Internet zu erreichen, was in der Laborumgebung zum Abbruch führen würde.
- **inst.ks=http://192.168.1.10/fedora_ks.cfg:** Dies ist das Äquivalent zum url-Parameter bei Debian. Er verweist auf die zentrale Kickstart-Datei auf dem **Nginx-Webserver**, die alle Antworten für den Installationsprozess enthält.
- **Kernel & Initrd:** Die Dateien **vmlinuz** und **initrd.img** werden über den **dnsmasq-TFTP-Dienst** geladen, um das System initial zu booten, bevor der Wechsel zum HTTP-Protokoll für die Datenübertragung erfolgt.

Durch die Kombination dieser Parameter wird ein nahtloser Übergang vom Netzwerk-Boot zur vollautomatischen Installation gewährleistet, wobei alle benötigten Daten ausschließlich vom lokalen PXE-Server bezogen werden.

4.3 Automatisierung von CachyOS (Arch-basiert)

Die Automatisierung von CachyOS stellt eine Besonderheit in diesem Projekt dar. Da CachyOS auf **Arch Linux** basiert, folgt es dem Prinzip des "Rolling Release" und verzichtet auf einen klassischen, starren Installer wie Debian-Installer oder Anaconda. Stattdessen basiert die automatisierte Bereitstellung hier auf einem hochflexiblen Scripting-Ansatz oder dem **archinstall**-Framework.

4.3.1 Implementierung der JSON-basierten Automatisierung

Für die Bereitstellung von CachyOS wurde ein moderner, deklarativer Ansatz gewählt. Anstelle von klassischen Skripten wird eine zentrale Konfigurationsdatei im **JSON-Format** verwendet. Diese Datei definiert den Zielzustand des Systems und wird vom Installer während des Bootvorgangs eingelesen, um eine vollautomatische Installation ("Silent Install") durchzuführen.

Die vorliegende Konfiguration umfasst folgende technische Details:

- **Disk-Management und Partitionierung:**
 - Die Datei erzwingt eine vollständige Löschung der Festplatte `/dev/sda` (`"wipe": true`).
 - Es wird eine primäre Partition angelegt, die 100% des verfügbaren Speicherplatzes einnimmt und direkt als Root-Verzeichnis (`/`) gemountet wird.
- **Infrastruktur-Integration (Nginx-Mirror):**
 - Ein kritischer Aspekt ist die `mirror_config`. Hier wurde ein **Custom Mirror** definiert, der direkt auf den lokalen Nginx-Server verweist:
`http://192.168.1.10/extract/storage/cachyos/`. Dies stellt sicher, dass auch CachyOS im isolierten Netzwerk ohne Internetzugriff installiert werden kann.
- **System-Komponenten und Pakete:**
 - Als Bootloader wird `grub-install` verwendet.
 - Das Paket-Set ist auf das Wesentliche optimiert: Neben dem spezialisierten `linux-cachyos` Kernel werden `base`, `base-devel` und `openssh` für den Fernzugriff installiert.
 - Für das Audio-Subsystem wird der moderne Standard `pipewire` vorkonfiguriert.
- **Benutzer- und Netzwerkverwaltung:**
 - Ein Benutzer namens `cachy` wird automatisch mit Sudo-Berechtigungen angelegt.
 - Die Netzwerkkonfiguration wird über den `NetworkManager` (`nm`) gesteuert, wobei die IP-Zuweisung dynamisch via DHCP erfolgt.

Durch die Verwendung dieses JSON-Schemas wird eine hohe Fehlerresistenz erreicht, da die Syntax vor der Ausführung validiert werden kann und eine konsistente Konfiguration über mehrere Installationen hinweg garantiert ist.

4.3.2 Integration ins PXE-Boot-Menü

Die Einbindung von CachyOS in das PXE-Menü erfordert eine komplexere Parameter-Übergabe als bei herkömmlichen Distributionen. Da Arch-basierte Systeme das gesamte Live-Dateisystem (SquashFS) über das Netzwerk laden müssen, bevor der Installationsprozess beginnt, wurden spezifische HTTP-Boot-Parameter implementiert.

Die Konfiguration in der Datei `/srv/tftp/pixelinux.cfg/default` lautet:

```
LABEL cachyos
    MENU LABEL 5) Install CachyOS
    KERNEL /os-images/cachyos/vmlinuz-linux-cachyos-lts
    APPEND initrd=/os-images/cachyos/initramfs-linux-cachyos-lts.img
    ip=dhcp archisobasedir=arch archisobasedrive=http
    archiso_http_srv=http://192.168.1.10/extract/storage/cachyos/
    archinstall_config=http://192.168.1.10/cachyos_config.json
```

Analyse der spezialisierten Boot-Parameter:

- **ip=dhcp**: Initialisiert die Netzwerkschnittstelle sofort beim Laden des Kernels, um die nachfolgenden HTTP-Anfragen zu ermöglichen.
- **archisobasedrive=http & archiso_http_srv=...**: Diese Parameter sind entscheidend für die webbasierte Bereitstellung. Sie weisen das System an, das Root-Dateisystem nicht auf einem lokalen Datenträger zu suchen, sondern die benötigten Fragmente via HTTP vom **Nginx-Server** nachzuladen.
- **archinstall_config=...**: Dies ist die Brücke zur im vorigen Kapitel beschriebenen Automatisierung. Der Pfad verweist direkt auf die **JSON-Konfigurationsdatei**. Der Installer erkennt diesen Parameter und startet automatisch die Installation basierend auf den darin enthaltenen Definitionen, ohne dass ein Benutzer eingreifen muss.
- **Kernel-Wahl**: Durch die Nutzung des `linux-cachyos-lts` Kernels wird eine stabile Basis für die Installation in der virtuellen Laborumgebung sichergestellt.

Diese Konfiguration demonstriert die nahtlose Verzahnung von PXE (für den Boot-Loader) und HTTP (für das Betriebssystem-Image und die Automatisierungslogik).

4.4 Automatisierung von Debian 11 und Kali Linux (Preseeding)

Sowohl Debian 11 als auch Kali Linux basieren auf dem **Debian-Installer (d-i)**. Daher wurde für beide Distributionen das **Preseeding-Verfahren** implementiert. Dabei handelt es sich um eine Konfigurationsdatei (`preseed.cfg`), die alle Antworten auf die Fragen des Installers vorab definiert.

4.4.1 Zentrale Konfigurationsaspekte der preseed.cfg

Die Automatisierung von Debian 11 und Kali Linux basiert auf der Datei `preseed.cfg`, die dem Installer alle notwendigen Antworten injiziert. In der vorliegenden Implementierung wurden folgende Kernbereiche konfiguriert, um eine unterbrechungsfreie Installation zu garantieren:

A. Hardware-Anpassung und CD-ROM-Bypass

Da der Debian-Installer standardmäßig versucht, ein physisches Medium zu finden, wurde die Hardware-Erkennung gezielt modifiziert. Dies ist der wichtigste Schritt für den PXE-Boot:

- `d-i cdrom-detect/search_devices boolean false`: Verhindert, dass der Prozess stoppt, wenn kein physisches CD-ROM-Laufwerk gefunden wird.
- `d-i mirror/protocol string http`: Zwingt den Installer, das Netzwerkprotokoll HTTP für den Bezug der Pakete zu nutzen.

B. Lokale Repository-Steuerung (Nginx Integration)

Ein entscheidendes Merkmal ist die Umleitung der Paketquellen auf den eigenen Nginx-Server. Dadurch wird kein Internetzugriff benötigt:

- **Mirror-Server:** `192.168.1.10`
- **Verzeichnis:** `/extract/storage/debian`
- **Authentifizierung:** `d-i debian-installer/allow_unauthenticated boolean true` (Ermöglicht die Installation auch ohne GPG-Signaturprüfung im lokalen Testnetzwerk).

C. Automatisierte Partitionierung (Expert Recipe)

Anstelle einer einfachen Standardpartitionierung wurde ein „Expert Recipe“ verwendet, um volle Kontrolle über das Festplatten-Layout zu haben:

- **Label:** `msdos`
- **Methode:** `regular` (Vermeidung von komplexem LVM für die Testumgebung).
- **Automatisierung:** Alle Bestätigungsdialoge (`confirm_write_new_label`, `confirm_nooverwrite`) wurden auf `true` gesetzt, damit der Installer die Festplatte ohne Rückfrage löscht und neu beschreibt.

D. Benutzerverwaltung und Abschluss

Sicherheitsrelevante und administrative Einstellungen werden ebenfalls automatisiert:

- **Accounts:** Sowohl der `root`-Account als auch ein Standard-Benutzer (`user`) werden mit dem Passwort `qweasd` angelegt.
- **GRUB-Bootloader:** Die Installation von GRUB im Master Boot Record (MBR) erfolgt automatisch ohne Benutzerinteraktion.
- **Abschluss:** Durch `finish-install/reboot_in_progress note` wird das System nach erfolgreicher Installation automatisch neu gestartet.

4.4.2 Technische Herausforderungen und spezifische Lösungen

Bei der Implementierung der automatisierten Installation für Debian 11 und Kali Linux traten spezifische Hürden auf, die durch gezielte Anpassungen in der `preseed.cfg` gelöst wurden.

1. Die Problematik der Medien-Erkennung (CD-ROM-Loop)

Der Standard-Installer von Debian sucht zu Beginn nach einem physischen CD-ROM-Laufwerk, um die Installationskomponenten (Anna) zu laden. In einer PXE-Umgebung führt dies ohne Korrektur zu einer Fehlermeldung und zum Abbruch der Automatisierung.

- **Lösung:** Durch das Setzen von `cdrom-detect/search_devices boolean false` und die Anweisung `anna/choose_modules string net-retriever` wurde der Installer gezwungen, die Hardware-Suche zu überspringen und stattdessen die benötigten Module direkt über das Netzwerk (HTTP) vom Nginx-Server zu beziehen.

2. Handhabung nicht authentifizierter Paketquellen

Da in der isolierten Laborumgebung ein lokaler Mirror auf dem Nginx-Server verwendet wird, der keine Verbindung zu den offiziellen Debian-GPG-Schlüsseln hat, würde der Installer die Installation aufgrund „unsicherer Quellen“ verweigern.

- **Lösung:** Die Direktive `d-i debian-installer/allow_unauthenticated boolean true` wurde implementiert. Dies ermöglicht eine reibungslose Installation innerhalb der abgeschotteten Testumgebung, ohne dass eine manuelle Bestätigung der Sicherheitswarnung erforderlich ist.

3. Vollständige Automatisierung der Partitionierung (Confirmation-Bypass)

Ein häufiger Stolperstein ist die Sicherheitsabfrage des Installers vor dem Schreiben der neuen Partitionstabelle („Wollen Sie die Änderungen wirklich auf die Festplatte schreiben?“).

- **Lösung:** Um eine echte „Zero-Touch“-Installation zu erreichen, mussten mehrere Parameter gleichzeitig auf `true` gesetzt werden:
 - `d-i partman/confirm_write_new_label boolean true`
 - `d-i partman/confirm boolean true`
 - `d-i partman/confirm_nooverwrite boolean true` Erst die Kombination dieser Parameter verhindert jegliche interaktive Rückfrage während der Festplattenvorbereitung.

4. Automatischer Reboot nach Abschluss

Standardmäßig wartet der Installer am Ende auf eine Bestätigung (Note), dass die Installation abgeschlossen ist.

- **Lösung:** Mit dem Befehl `d-i finish-install/reboot_in_progress note` wird diese Meldung unterdrückt und das System führt unmittelbar einen Neustart durch, wodurch der Prozess vollständig autonom endet.

4.4.3 Integration in das PXE-Boot-Menü

Der abschließende Schritt der Automatisierung für Debian-basierte Systeme (wie Kali Linux) ist die präzise Konfiguration des PXE-Eintrags. Hierbei werden dem Installer bereits beim Bootvorgang über die APPEND-Zeile kritische Parameter übergeben, die den Weg für die weitere Automatisierung ebnen.

Die Konfiguration für Kali Linux in der Datei “`/srv/tftp/pixelinux.cfg/default`” sieht wie folgt aus:

```
LABEL debian
    MENU LABEL 1) Install Debian
    KERNEL /os-images/debian/linux
    APPEND initrd=/os-images/debian/initrd.gz auto=true
    priority=critical url=http://192.168.1.10/debian_preseed.cfg
    cdrom-detect/search_devices=false mirror/protocol=http
    mirror/http/hostname=192.168.1.10
    mirror/http/directory=/extract/storage/debian/
    netcfg/get_nameservers=192.168.1.10 netcfg/confirm_static=true
```

```
LABEL kali
    MENU LABEL 2) Install Kali
    KERNEL /os-images/kali/vmlinuz
    APPEND initrd=/os-images/kali/initrd.gz auto=true priority=critical
    url=http://192.168.1.10/kali_preseed.cfg netcfg/choose_interface=auto
    netcfg/get_nameservers=192.168.1.10 mirror/protocol=http
    mirror/http/hostname=192.168.1.10
    mirror/http/directory=/extract/storage/kali/
    anna/choose_modules=net-retriever cdrom-detect/search_devices=false
```

Analyse der implementierten Kernel-Parameter:

- **auto=true & priority=critical:** Diese Flags erzwingen den Start des automatisierten Modus und unterdrücken alle nicht kritischen Benutzerabfragen.
- **url=http://192.168.1.10/kali_preseed.cfg:** Teilt dem Installer den Pfad zur zentralen Antwortdatei auf dem Nginx-Server mit.
- **Netzwerk-Vorkonfiguration (netcfg/):** Durch `choose_interface=auto` und `get_nameservers=192.168.1.10` wird die Netzwerkkonfiguration bereits beim Booten fixiert, noch bevor die Preseed-Datei verarbeitet wird. Dies stellt sicher, dass der Abruf der Datei via HTTP stabil erfolgt.
- **Mirror-Vorgaben (mirror/):** Hier wird explizit definiert, dass die Pakete über HTTP vom lokalen Server (192.168.1.10) aus dem Verzeichnis `/extract/storage/kali/` bezogen werden sollen.
- **Bypass-Mechanismen:** * `cdrom-detect/search_devices=false`: Deaktiviert die Suche nach physischen Medien.

- `anna/choose_modules=net-retriever`: Weist den Installer an, zusätzliche Komponenten über das Netzwerk nachzuladen.

Durch diese detaillierte Übergabe von Parametern in der Boot-Phase wird eine robuste "Zero-Touch"-Installation realisiert, die selbst bei Hardware-Erkennungsschleifen nicht unterbrochen wird.

5. Technische Herausforderungen und Troubleshooting

Während der Implementierungsphase traten mehrere technische Hürden auf, die sowohl die Hardware-Erkennung als auch die Integrität der Software-Pakete betrafen. Die Lösung dieser Probleme war entscheidend für die Stabilität des automatisierten Prozesses.

5.1 Herausforderung bei der Hardware-Erkennung (CD-ROM-Bypass)

Problemstellung: Eine der signifikantesten Hürden bei der PXE-Installation von Debian und Kali Linux ist die automatisierte Suche nach einem physischen Installationsmedium. Da die Standard-Installer-Routinen darauf ausgelegt sind, Komponenten von einer CD-ROM zu laden, stoppt der Prozess in einer virtuellen PXE-Umgebung mit einer Fehlermeldung, sobald kein lokales Laufwerk gefunden wird. Dies verhindert eine unterbrechungsfreie "Zero-Touch"-Automatisierung.

Lösung und technischer Workaround: Um diesen Engpass zu überwinden und den Installer zur Nutzung der Netzwerkressourcen zu zwingen, wurden folgende Maßnahmen implementiert:

- **Verwendung von Netboot-Images:** Anstatt der Kernel-Dateien von herkömmlichen Desktop-ISOs wurden dedizierte Netboot-Kernel (`vmlinuz`) und Initrd-Images verwendet. Diese sind speziell dafür vorkonfiguriert, Hardware-Abfragen für lokale Medien zu minimieren.
- **Kernel-Parameter-Injektion:** In der Konfigurationsdatei des PXE-Menüs wurden spezifische Flags an die `APPEND`-Zeile übergeben:
 - `cdrom-detect/search_devices=false`: Dieser Befehl deaktiviert aktiv die Hardware-Suche nach optischen Laufwerken.
 - `anna/choose_modules=net-retriever`: Diese Anweisung instruiert den Installer-Dienst "Anna", benötigte Komponenten (udebs) nicht lokal, sondern über das Netzwerk-Modul (Net-Retriever) vom HTTP-Server zu beziehen.

Ergebnis: Durch diese Konfiguration überspringt der Installer die fehleranfällige Suche nach physischen Datenträgern und wechselt unmittelbar in den Netzwerk-Modus. Dadurch wird eine stabile Kommunikation mit dem Nginx-Server sichergestellt, was die Voraussetzung für eine vollautomatische Installation ist.

5.2 Konsistenz zwischen Kernel und Repository (Fehler: "No kernel modules found")

Herausforderung: Ein kritischer Fehler, der während der Testphase auftrat, war die Meldung "No kernel modules found". Dieses Problem entsteht, wenn die Version des über TFTP geladenen Kernels (`vmlinuz`) nicht exakt mit den Kernel-Modulen übereinstimmt, die im Repository auf dem Nginx-Server bereitgestellt werden. Da Linux-Treiber (Kernel-Module) strikt an eine spezifische Kernel-Version gebunden sind, führt bereits ein minimaler Versionsunterschied dazu, dass Hardware-Komponenten wie Netzwerkkarten oder Festplatten-Controller nicht initialisiert werden können. Dies führt zwangsläufig zum Abbruch der automatisierten Installation.

Lösung: Um eine vollständige Kompatibilität zwischen dem Boot-Vorgang und der Paketinstallation zu gewährleisten, wurde folgendes Verfahren implementiert:

- **Synchronisierte Extraktion:** Es wurde strikt darauf geachtet, dass die Dateien `vmlinuz` und `initrd` direkt aus demselben ISO-Image extrahiert wurden, das auch als Datenquelle für das HTTP-Repository im Verzeichnis `/var/www/html/extract/storage/` verwendet wird.
- **Versionsprüfung:** Vor der Bereitstellung einer neuen Distribution wurde verifiziert, dass die Symlinks und Pfade im PXE-Menü auf die aktuellsten, zum Repository passenden Kernel-Images verweisen.
- **Bereinigung veralteter Dateien:** Um Cache-Konflikte zu vermeiden, wurden bei jeder Aktualisierung der Installationsmedien die alten Boot-Dateien im TFTP-Verzeichnis gelöscht und durch die exakt passenden Versionen der neuen ISO ersetzt.

Ergebnis: Durch diese strikte Synchronisation von Boot-Dateien und Repository-Inhalten wurde sichergestellt, dass der Installer alle notwendigen Treiber laden kann, was eine stabile Hardware-Erkennung über das Netzwerk ermöglicht.

5.3 Speicherplatzmangel und Erweiterung der Festplattenkapazität

Herausforderung: Im Verlauf des Projekts wurde die Speicherkapazität des PXE-Servers durch die gleichzeitige Bereitstellung von fünf verschiedenen Linux-Distributionen (Debian, Kali, Fedora, Rocky Linux und CachyOS) vollständig ausgelastet. Jede Distribution erfordert Speicherplatz für das originale ISO-Image sowie für die entpackten Dateien im HTTP-Repository des Nginx-Servers. Dies führte dazu, dass die Root-Partition des Servers an ihre Grenzen stieß, was die Extraktion weiterer Betriebssysteme verhinderte und die Systemstabilität gefährdete.

Lösung: Um dieses Problem nachhaltig zu lösen, wurde das Storage-Konzept des Servers wie folgt angepasst:

- **Integration einer Zusatzfestplatte:** Dem virtuellen Server wurde eine zusätzliche Festplatte mit einer Kapazität von **150 GB** zugewiesen.
- **Dateisystem-Integration:** Diese Festplatte wurde formatiert und permanent unter dem Pfad `/var/www/html/extract/storage` in die Verzeichnisstruktur eingehängt (Mounting).
- **Datenmigration:** Sämtliche volumeneffektiven Daten (die entpackten ISO-Inhalte) wurden auf diesen neuen Datenträger verschoben.

Ergebnis: Durch diese Maßnahme wurde eine saubere Trennung zwischen dem Betriebssystem des PXE-Servers und den Bereitstellungsdaten (OS-Repositories) erreicht. Dies stellt sicher, dass das Anwachsen der Repository-Daten keine negativen Auswirkungen auf die Systempartition hat und ermöglicht eine problemlose Skalierbarkeit für zukünftige Betriebssystem-Erweiterungen.

5.4 Optimierung der Installationsmedien (Netinstall vs. Full-ISO)

Herausforderung: Zu Beginn des Projekts wurden primär "Netinstall"- oder "Minimal"-Images der Distributionen verwendet, um Speicherplatz und Downloadzeit zu sparen. Diese Images enthalten lediglich ein Basissystem und versuchen während der Installation, zusätzliche Pakete von öffentlichen Spiegelservern im Internet nachzuladen. Da das Projekt jedoch in einem **isolierten Labornetzwerk** ohne Internetzugang realisiert wurde, schlugen diese Installationsversuche fehl, da die künftigen Clients keine Verbindung zu den externen Repositories aufbauen konnten.

Lösung: Die Strategie wurde grundlegend auf ein lokales Bereitstellungsmodell umgestellt:

- **Wechsel zu Full/Offline-ISOs:** Es wurden die vollständigen DVD-Images (Full-ISOs) der Betriebssysteme bezogen, die bereits alle notwendigen Softwarepakete enthalten.
- **Aufbau lokaler Repositories:** Diese ISOs wurden vollständig entpackt und über den **Nginx-Webserver** als lokale Paketquellen zur Verfügung gestellt.
- **Konfigurationsanpassung:** In den Antwortdateien (Preseed/Kickstart) wurde die Suche nach Online-Spiegelservern deaktiviert und stattdessen die IP-Adresse des eigenen Servers (**192.168.1.10**) als exklusive Quelle definiert.

Ergebnis: Durch den Einsatz von vollständigen Offline-Medien wurde die Installation zu 100 % unabhängig von einer externen Internetverbindung. Dies garantiert nicht nur die Funktionalität im isolierten Netzwerk, sondern erhöht auch die Installationsgeschwindigkeit massiv, da die Datenübertragung mit lokaler Netzwerkgeschwindigkeit (LAN) stattfindet.

6. Absicherung und Einsatzbereiche des PXE-Boot-Servers

6.1 Absicherung des PXE-Boot-Servers

6.1.1 Bedrohungsmodell und Sicherheitsrisiken

Ein Preboot eXecution Environment (PXE) dient der Netzwerk-Bereitstellung von Bootimages zur automatisierten Installation oder dem Start von Betriebssystemen. Der Standard selbst enthält keine Mechanismen für Authentifizierung oder Verschlüsselung, da er auf grundlegenden Protokollen wie DHCP und TFTP basiert, die ursprünglich nicht für sichere Kommunikation entwickelt wurden. Die fehlende kryptografische Absicherung eröffnet potenzielle Angriffsflächen, etwa *Man-in-the-Middle*-Angriffe oder Manipulationen von Boot-Images durch unautorisierte Systeme im gleichen Netzwerksegment.

Das Dynamic Host Configuration Protocol (DHCP), das hierbei zur Übermittlung von Konfigurationsparametern eingesetzt wird, besitzt keine eingebaute Authentifizierung zwischen Client und Server. Dadurch können sogenannte *Rogue DHCP Server* falsche Informationen an Clients aussenden, darunter fehlerhafte Boot-Server-Adressen oder manipulierte Boot-Optionen, was Clients dazu verleiten kann, bösartige Images zu laden. Dies ist eine bekannte DHCP-Schwachstelle, die ebenfalls in der akademischen DHCP-Sicherheitsliteratur beschrieben wird.

Weiterhin existieren direktere Protokollrisiken: TFTP überträgt Dateien im Klartext und verwendet das verbindungslose UDP, was bedeutet, dass Boot-Images und Konfigurationsdaten keiner Integritätsprüfung unterliegen und ohne Verschlüsselung übertragen werden. Dies ermöglicht es einem Angreifer, während der Übertragung Dateien abzufangen oder zu manipulieren, solange er Zugriff auf das gleiche Netzwerksegment besitzt.

Solche Schwachstellen führen in der Praxis dazu, dass ein kompromittierter Boot-Server, ein manipuliertes Boot-Image oder ein nicht isoliertes Netzwerksegment zur Kontrolle über Clients schon vor dem Start des Betriebssystems genutzt werden kann. Diese Risiken sind im Kontext von automatisierten Installationen besonders gravierend, da durch einen kompromittierten Bootprozess Malware oder unerwünschte Systemkonfigurationen bereits vor dem eigentlichen OS-Laden etabliert werden können.

6.1.2 Netzwerkssegmentierung des PXE-Servers

Die Segmentierung des Netzwerkes, in dem der PXE-Boot-Server arbeitet, stellt eine Kernmaßnahme zur Risikoreduzierung dar. Durch die physische oder logische Trennung des PXE-Bereichs vom restlichen Unternehmensnetzwerk wird verhindert, dass unautorisierte Systeme auf die Bootdienste zugreifen oder diese anbieten können. Ein isoliertes VLAN oder Subnetz, in dem lediglich administrierte Clients PXE-Bootanfragen stellen dürfen, senkt die Wahrscheinlichkeit eines Angreifers erheblich, der Zugang zum PXE-Dienst erlangt.

Die Firewall-Konfiguration kann zusätzlich so gestaltet werden, dass nur die notwendigen Portnummern für DHCP (UDP 67/68) und TFTP (UDP 69) innerhalb des isolierten Netzsegments geöffnet werden, wodurch allgemeine Broadcast- oder Portscanning-Angriffe aus äußeren Netzwerken unterbunden werden.

6.1.3 Absicherung der DHCP- und TFTP-Dienste

Die Absicherung der zentralen Dienste DHCP und TFTP ist ein entscheidender Bestandteil beim Betrieb eines PXE-Boot-Servers, da beide Protokolle im Standardbetrieb keine integrierten Authentifizierungs- oder Sicherheitsmechanismen besitzen und somit potenziell missbraucht werden können.

DHCP Snooping als Netzwerkmechanismus stellt eine etablierte Schutzmaßnahme dar, um *Rogue-DHCP-Server* zu verhindern. Durch DHCP Snooping wird der DHCP-Verkehr in einem Layer-2-Switch überwacht und nur Paketen von autorisierten (vertrauenswürdigen) Ports die Weiterleitung erlaubt. Pakete von nicht vertrauenswürdigen Ports werden blockiert, wodurch verhindert wird, dass unautorisierte Geräte falsche DHCP-Antworten versenden und Clients manipulierte Netzwerkparameter zuweisen. DHCP Snooping speichert zudem gültige DHCP-Zuordnungen in einer "Binding Database", die zur Überprüfung späterer Netzwerkaktivitäten genutzt werden kann. Diese Technik wird in der Netzwerksicherheitspraxis eingesetzt, um DHCP-Spoofing und -Attacken zu reduzieren.

Auf Netzwerkhardware wie Switches und Routern kann DHCP Snooping konfiguriert werden, indem zunächst alle Ports als *untrusted* definiert werden und anschließend die Ports, an denen sich der legitime DHCP-Server befindet, als *trusted* gekennzeichnet werden. Dies bedeutet, dass nur definierte Geräte IP-Adressvergabe-Pakete aussenden dürfen. Diese Maßnahme reduziert nicht nur das Risiko von Rogue-DHCP-Servern, sondern dient auch als Grundlage für weitere Schutzmechanismen wie *Dynamic ARP Inspection (DAI)* oder *IP Source Guard*, die auf den DHCP Snooping-Bindungen aufbauen können, um zusätzliche Angriffsflächen zu minimieren.

Während DHCP Snooping die Authentizität der DHCP-Kommunikation verbessert, bleibt beim TFTP-Dienst ein weiteres Risiko bestehen: das Protokoll ist von Haus aus einfach und unsicher. TFTP wurde ursprünglich für einfache Dateiübertragungen entwickelt und enthält keine eingebaute Authentifizierung oder Verschlüsselung, wodurch Server und Clients im Netzwerkverkehr für Manipulation oder unbefugten Zugriff anfällig sind. Zudem verwendet TFTP UDP, was keine zuverlässige Transportkontrolle oder Session-Authentifizierung bietet, wodurch Angreifer durch Netzwerzugriff Dateien abfangen oder modifizieren könnten.

Darüber hinaus lassen sich Dateizugriffsrechte des TFTP-Servers restriktiv setzen, indem nur die minimal notwendigen Boot-Images bereitgestellt und Schreibrechte für nicht vertrauenswürdige Prozesse entzogen werden. Eine Protokollierung der TFTP-Zugriffe kann zusätzlich dazu beitragen, unerwartete Zugriffe oder Fehlverhalten frühzeitig zu erkennen und darauf reagieren zu können.

Diese Maßnahmen (DHCP Snooping, gezielte Port-Konfiguration, ACLs und restriktive Dateizugriffsrechte) sind praxisnahe Verfahren, um die inhärenten Sicherheitsdefizite von DHCP und TFTP zu kompensieren und ein sicheres, netzwerkgestütztes Boot-Umfeld zu etablieren.

6.1.4 Schutz der Boot-Images und Installationsdateien

Ein zentraler Punkt der PXE-Sicherheit ist die Gewährleistung der **Integrität der bereitgestellten Boot-Images**. Dies lässt sich erreichen durch digitale Signaturen oder Prüfsummen, die vor der Ausführung auf dem Client überprüft werden. Obwohl PXE keine eingebauten Sicherheitsmechanismen hierfür besitzt, kann eine zusätzliche Signaturlogik implementiert werden, die sicherstellt, dass Boot-Images nicht manipuliert wurden.

Eine weitere Absicherung besteht darin, den TFTP-Server selbst von sensiblen Daten fernzuhalten und nur die notwendigsten Boot-Images zu hosten. Dies minimiert den Schaden eines möglichen Datenabflusses oder einer unbeabsichtigten Bereitstellung von vertraulichen Informationen über das Netzwerk.

6.1.5 Erweiterte Absicherung: Secure Boot und Vertrauenskette

Über die Netzwerk- und Protokollebene hinaus lässt sich die Boot-Sicherheit durch **Secure Boot-Mechanismen auf Clientseite** stärken. Secure Boot ist eine UEFI-Funktion, bei der vor dem Laden jedes Boot-Elements eine **Signaturprüfung durchgeführt wird**. Dadurch wird verhindert, dass nicht authentifizierte Bootloader oder Kernel ausgeführt werden, selbst wenn ein Angreifer unautorisiert Zugang zur PXE-Infrastruktur erhalten hat.

Secure Boot baut eine **Vertrauenskette vom Firmware-Start bis zum Betriebssystemkernel** auf und reduziert so die Gefahr, dass präparierte Bootkosten oder manipulierte Images erfolgreich ausgeführt werden können. Dabei ist zu beachten, dass Secure Boot zwar die Integrität des Bootprozesses schützt, jedoch **nicht automatisch die Authentizität des Servers oder die Netzwerkkommunikation absichert**, weshalb es zusammen mit den anderen Sicherheitsmaßnahmen eingesetzt werden sollte.

6.2 Einsatzbereiche von PXE-Boot

Die Implementierung des PXE-Standards (Preboot Execution Environment) stellt eine fundamentale Methode zur Fernsteuerung von Boot-Vorgängen in rechnergestützten Netzwerken dar. Durch die Entkoppelung des Boot-Prozesses von lokalen Speichermedien ergeben sich spezialisierte Einsatzgebiete, die vor allem in skalierten Infrastrukturen von Bedeutung sind.

6.2.1 Automatisierte Betriebssystembereitstellung

Die automatisierte Bereitstellung (Deployment) von Betriebssystemen ist das primäre Einsatzgebiet von PXE. Hierbei kommen Mechanismen der **unbeaufsichtigten Installation** zum Einsatz, wie etwa **Debian Preseed**, **Red Hat Kickstart** oder **Ubuntu Autoinstall**.

Diese Verfahren ermöglichen es, Konfigurationsparameter (Partitionierung, Paketauswahl, Benutzerkonten) in einer Antwortdatei zu definieren. PXE dient hierbei als Initialzündung: Das System lädt über das Netzwerk den Kernel und die Konfigurationsdatei, wodurch manuelle Eingriffe durch Administratoren vollständig eliminiert werden. Dies führt zu einer signifikanten Reduktion der Fehlerquote und einer massiven Zeitersparnis bei der Ersteinrichtung.

6.2.2 Nutzung in Unternehmens- und Rechenzentrumsumgebungen

In modernen Rechenzentren und großen Unternehmensnetzwerken ist PXE-Boot ein grundlegender Bestandteil von Provisionierungsprozessen. Durch die zentrale Steuerung über PXE-Server lassen sich Betriebssysteme oder System-Images schnell und wiederholbar auf Bare-Metal-Servern oder virtuellen Knoten verteilen. Viele kommerzielle und Open-Source-Orchestrierungs-Frameworks integrieren PXE-Boot als Startpunkt für Server-Provisioning, bevor weiterführende Automatisierungswerkzeuge wie *kickstart* (Red Hat) oder *FAI* (Fully Automatic Installation) greifen.

Der Vorteil dieses Ansatzes liegt nicht nur in der Geschwindigkeit, sondern auch in der Standardisierung von Hardware-Konfigurationen, Sicherheitspolicies und Software-Stacks. Durch die zentrale Verwaltung wird sichergestellt, dass alle Serverknoten ein identisches Basisimage besitzen, was Fehlerquellen minimiert und die Wartbarkeit erhöht. Insbesondere bei Perioden mit hoher Serverdichte, wie etwa bei Cluster-Aufbauten oder Cloud-Infrastrukturen, zeigt sich die Leistungsfähigkeit netzwerkbasierter Boot- und Deployment-Prozesse.

6.2.3 Einsatz in Schulungs-, Test- und Laborumgebungen

In Umgebungen mit hoher Fluktuation, wie Computerlaboren oder Testzentren, bietet PXE eine **hohe Flexibilität**. Da Systeme nach jeder Nutzung auf einen definierten Ausgangszustand zurückgesetzt werden müssen, erlaubt PXE die schnelle Bereitstellung temporärer Betriebssysteme. Auch der Wechsel zwischen verschiedenen Architekturen oder OS-Versionen zu Testzwecken lässt sich zentral steuern, ohne dass physischer Zugriff auf die Client-Hardware notwendig ist.

6.2.4 Unterstützung von Diskless- bzw. Thin-Client-Umgebungen

Ein weiteres Einsatzfeld von PXE-Boot findet sich in Diskless- oder Thin-Client-Architekturen, wo Clients keinen eigenen lokalen Speicher besitzen und stattdessen das Betriebssystem und Anwendungen direkt vom Netzwerkservier laden. Dies wird beispielsweise in Virtual Desktop Infrastructure (VDI)-Lösungen oder in Spezialkonfigurationen wie wissenschaftlichen Clustern und öffentlichen Computerlaboren genutzt. PXE-Boot ermöglicht dabei den Netzwerkstart und die verbindliche Steuerung der Clients durch zentrale Server, was insbesondere bei der Verwaltung heterogener Rechnerlandschaften von Vorteil ist.

6.3 Vergleich: Traditionelles PXE (TFTP) vs. Modernes HTTP-Boot

Im Rahmen dieses Projekts wurde ein hybrider Ansatz verwendet: Der initiale Boot erfolgt über PXE (TFTP), während die schweren Datenpakete über HTTP (Nginx) übertragen werden. Ein direkter Vergleich dieser Technologien verdeutlicht die Relevanz moderner Übertragungsprotokolle für das Deployment.

Merkmal	Traditionelles PXE (Legacy/TFTP)	Modernes HTTP-Boot (UEFI)
Transportprotokoll	UDP (verbindungslos)	TCP (verbindungsorientiert)
Übertragungsdienst	TFTP (Trivial File Transfer Protocol)	HTTP / HTTPS
Fehlerkorrektur	Minimal (Timeouts, Block-Ebene)	Hoch (TCP-Handshake & Retransmission)
Geschwindigkeit	Gering (hoher Overhead durch ACKs)	Hoch (optimierte TCP-Fenstergröße)
Sicherheit	Keine native Verschlüsselung	TLS/SSL Unterstützung möglich
Infrastruktur	Erfordert DHCP-Optionen 66/67	Nutzt Standard-Web-Infrastruktur

Warum dieser Vergleich wichtig ist:

- Zuverlässigkeit:** TFTP basiert auf UDP und hat keine native Staukontrolle. In großen Netzwerken oder bei hoher Last können Pakete verloren gehen, was den Boot-Vorgang abbricht. HTTP nutzt TCP, was eine garantierter Zustellung der Daten ermöglicht.
- Skalierbarkeit:** Ein Nginx-Webserver kann hunderte Clients gleichzeitig mit ISO-Images bedienen, während TFTP-Server bei parallelen Zugriffen oft an ihre Leistungsgrenzen stoßen.
- Firewall-Kompatibilität:** HTTP-Traffic (Port 80/443) wird in modernen Unternehmensnetzwerken problemlos geroutet, während TFTP oft durch Sicherheitsrichtlinien blockiert wird.

Fazit für das Projekt: Obwohl die Testumgebung auf klassischem PXE basiert, wurde durch die Auslagerung der Installationsmedien auf den **Nginx-Webserver** (Schicht 3) die Stabilität eines modernen HTTP-Boot-Verfahrens simuliert. Dies kombiniert die universelle Kompatibilität von PXE mit der Performanz von HTTP.

7. Fazit und Ausblick

7.1 Zusammenfassung der Ergebnisse

Das primäre Ziel dieses Projekts, die Entwicklung und Implementierung eines vollautomatisierten PXE-Boot-Servers in einer isolierten Laborumgebung, wurde erfolgreich realisiert. Durch die Kombination aus **dnsmasq** (DHCP/TFTP) und **Nginx** (HTTP) konnte eine hybride Infrastruktur geschaffen werden, die sowohl stabil als auch performant ist.

Die Automatisierung für fünf verschiedene Distributionen (Debian, Kali, Fedora, Rocky Linux und CachyOS) wurde durch den Einsatz von Preseed, Kickstart und JSON-Konfigurationen vollständig umgesetzt. Besonders hervorzuheben ist die erfolgreiche Lösung von Hardware-Erkennungsproblemen (CD-ROM-Bypass) und die Optimierung des Ressourcenmanagements durch die Integration von zusätzlichem Speicherplatz.

Ergänzend kann der Einsatz von Secure-Boot-Mechanismen das Vertrauen in die Boot-Kette weiter erhöhen, auch wenn dies mit zusätzlichen administrativen Aufwand verbunden ist. Die dargestellten Einsatzbereiche verdeutlichen, dass PXE-Boot vor allem in kontrollierten Umgebungen wie Unternehmensnetzwerken oder Rechenzentren seine Stärken ausspielt. Unter Berücksichtigung angemessener Sicherheitsmaßnahmen stellt PXE-Boot somit eine leistungsfähige und praxisnahe Lösung zur automatisierten Systembereitstellung dar.

7.2 Reflexion und Mehrwert

Der Aufbau zeigt, dass moderne IT-Infrastrukturen durch "Zero-Touch"-Installationen massiv an Effizienz gewinnen. Der Mehrwert dieses Projekts liegt in der:

- **Zeitersparnis:** Die parallele Installation mehrerer Clients ist nun ohne manuellen Eingriff möglich.
- **Standardisierung:** Jeder installierte Client verfügt über exakt die gleiche Konfiguration, was Fehlerquellen minimiert.
- **Unabhängigkeit:** Durch die Bereitstellung lokaler Repositories ist das System autark und benötigt keinen Internetzugang.

7.3 Ausblick

Obwohl das System in seiner jetzigen Form voll funktionsfähig ist, bietet es Raum für zukünftige Erweiterungen:

1. **Windows-Integration:** Die Einbindung von Windows PE (WinPE) und dem Microsoft Deployment Toolkit (MDT) in das bestehende PXE-Menü.
2. **Sicherheit:** Implementierung von HTTPS für die Übertragung der Konfigurationsdaten, um die Integrität im Netzwerk weiter zu erhöhen.
3. **Monitoring:** Integration eines Dashboards zur Überwachung der laufenden Installationsvorgänge in Echtzeit.

Abschließend lässt sich festhalten, dass das Projekt die Komplexität moderner Netzwerk-Bereitstellungen erfolgreich reduziert hat und eine solide Basis für weitere Automatisierungsschritte bietet.

Literaturverzeichnis

- DHCP | Dnsmasq:
<https://help.ubuntu.com/community/Dnsmasq>
- Nginx:
https://nginx.org/en/docs/beginners_guide.html
- Nginx | ubuntu
<https://ubuntu.com/tutorials/install-and-configure-nginx#1-overview>
- Deploying Kali over Network PXE Install | Kali Linux Documentation:
<https://www.kali.org/docs/installation/network-pxe/>
- Kickstart Documentation:
<https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html>
- Example-preseed:
<https://www.debian.org/releases/bullseye/example-preseed.txt>
- Linux PXE Boot Server Setup Guide | PDF | Network Architecture:
<https://de.scribd.com/document/349177590/how-to-configure-pxe-boot-server-in-linux>
- preseed partitioning failure (reddit):
https://www.reddit.com/r/debian/comments/1mi6s4s/preseed_partitioning_failure/
- DebianInstaller/Preseed - Debian Wiki:
<https://wiki.debian.org/DebianInstaller/Preseed>
- Debian-preseed (github):
<https://github.com/paullockaby/debian-preseed/blob/main/preseed.cfg>
- Kali Linux ISO:
<https://www.kali.org/get-kali/#kali-installer-images>
- Debian ISO:
<https://www.debian.org/distrib/netinst>
- Rocky Linux ISO:
<https://rockylinux.org/de-DE/download>
- Fedora ISO:
<https://www.fedoraproject.org/server/download>
- CachyOS ISO:
<https://cachyos.org/download/>
- Kali Linux netboot:
<https://http.kali.org/kali/dists/kali-rolling/main/installer-amd64/current/images/netboot/>
- Intel:
<https://www.intel.com/content/www/us/en/developer/articles/technical/network-boot-in-a-zero-trust-environment.html?utm.com>
- thegrenze.com:
https://thegrenze.com/pages/servej.php?association=GRENZE&fn=101_23.pdf&id=5148&issue=2&journal=GIJET&name=Strengthening+Network+Security+through+DHCP+RiskMitigation&volume=11&year=2025

- NDSS Symposium:
<https://www.ndss-symposium.org/wp-content/uploads/2025-330-paper.pdf>
- Casify:
<https://caasify.com/master-pxe-and-ipxe-setup-configure-dhcp-tftp-uefi-boot/>
- Cisco:
https://www.cisco.com/c/dam/global/en_ae/assets/ciscoexposaudi2008/assets/securing-places-on-the-network-nadhem-alfardan.pdf
- UserComp:
<https://usercomp.com/news/1400056/secure-tftp-server-on-ubuntu>
- CliffsNotes:
<https://www.cliffsnotes.com/study-notes/22846810>
- Acceis:
<https://www.acceis.fr/tackling-pxe-images/>
- Stallings (2016): Foundations of Modern Networking:
<https://ptgmedia.pearsoncmg.com/images/9780134175393/samplepages/9780134175393.pdf>
- Intel Corporate:
<https://dimitrije.website/files/pxespec.pdf>
- Irjet:
<https://www.irjet.net/archives/V10/I4/IRJET-V10I481.pdf>
- TechTarget:
<https://www.techtarget.com/searchnetworking/definition/Preboot-Execution-Environment>

Anhang

- GitHub: <https://github.com/HamidHekmatnezhad/PXE-Boot-Server>
- GitLab (THD): https://mygit.th-deg.de/aa02067/pxe_boot_server#