

Zero Trust in heterogenen IT-/OT-Netzwerken

Bachelorarbeit Frühjahrssemester 2026

Baruyr Bousnaian & Ivo Cavelti

Konzeption, Umsetzung und Evaluation im Vergleich zu perimeterbasierter
Sicherheit



Studiengang Informatik
Ostschweizer Fachhochschule
Campus Rapperswil-Jona

11. Juni 2026

Betreuer: Prof. Dr. Daniel Patrick Politze
Experte: Ramon Schildknecht
Gegenleserin: Prof. Dr. Mitra Purandare

Zusammenfassung

Heterogene Netzwerkkumgebungen aus Geräten der Informationstechnik (IT) und der Operational Technology (OT) werden traditionell über perimeterbasierte Sicherheitsarchitekturen geschützt, in denen interne Netze als vertrauenswürdig gelten. Mit Cloud-Diensten, mobilen Arbeitsformen und zunehmend vernetzten OT-Komponenten stösst dieses Modell an Grenzen. Diese Bachelorarbeit untersucht deshalb die Forschungsfrage, welche Vorteile eine Zero-Trust-Architektur in einer heterogenen Netzwerkkumgebung mit IT- und OT-Geräten gegenüber einer perimeterbasierten Sicherheitsarchitektur bietet. Dazu wurden zehn Hypothesen formuliert und in einer eigens konzipierten Laborumgebung mit IT-Diensten, OT-nahen Komponenten und externen Wartungszugriffen experimentell evaluiert.

Die Zero-Trust-Umsetzung basiert auf OpenZiti, das Netzwerkzugriffe nicht anhand von Netzwerkpositionen oder IP-Adressen, sondern anhand von Identitäten von Benutzern und Diensten kontrolliert, sowie auf Keycloak zur Authentifizierung. oauth2-proxy und Envoy setzen Zugriffsentscheide im Datenpfad durch, Open Policy Agent und Wazuh dienen der Policy-Auswertung, Protokollierung und automatisierten Eindämmung. Als Vergleich dient eine perimeterbasierte Architektur mit WireGuard-VPN, OPNsense-Firewall und veröffentlichten Diensten. Die Evaluation erfolgte anhand vier kontrollierter Angriffsszenarien (kompromittierte Zugangsdaten, kompromittierter Endpoint, kompromittierter Diensthost, Missbrauch eines legitimen Wartungszugangs) sowie drei Wartungsszenarien (Hinzufügen eines neuen Clients, Cloud-Dienstes und OT-Sensors). Alle Szenarien wurden als reproduzierbare Skripte umgesetzt und in beiden Architekturen ausgeführt.

Die Ergebnisse zeigen unter Laborbedingungen, dass kompromittierte oder missbräuchlich verwendete Identitäten in der Zero-Trust-Architektur ausschliesslich die explizit freigegebenen Dienste sehen und erreichen. Ein kompromittierter Diensthost konnte mangels Dial-Policies keine laterale Bewegung durchführen, während ein kompromittierter VPN-Peer in der perimeterbasierten Architektur Zugriff auf mehrere vollständige Subnetze erhielt. Eindämmungsmassnahmen wirkten identitätsgenau, dauerten ab Eingang der Meldung rund eine Sekunde und beeinträchtigten keine anderen Identitäten oder Dienste. Die erzeugten Protokolle erlauben zudem eine durchgängige Korrelation von Authentifizierung, Dienstsichtbarkeit, Policy-Entscheid und Eindämmung entlang einer gemeinsamen Identitätskette. Demgegenüber erforderte Zero Trust mehr initiale Konfiguration und laufende Pflege, da Identitäten, Services sowie Bind- und Dial-Policies explizit modelliert werden müssen. Ein Vorteil bei der Erkennungsgeschwindigkeit war nicht nachweisbar, da die Erkennung in beiden Umgebungen primär vom eingesetzten Endpoint-Schutz abhing.

Insgesamt bestätigen die Laborergebnisse, dass die untersuchte Zero-Trust-Architektur strukturelle Sicherheitsvorteile bei der Begrenzung der Angriffsfläche, der gezielten Eindämmung, der Nachvollziehbarkeit von Angriffspfaden und der kontrollierten Anpassbarkeit bietet, dieser Gewinn jedoch mit höherem Konfigurations- und Betriebsaufwand einhergeht. Die Aussagekraft ist durch den Laborcharakter und die kontrolliert simulierten Angriffe begrenzt.

Management Summary

Ausgangslage

Unternehmen betreiben heute zunehmend verteilte und heterogene Netzwerkumgebungen. Neben klassischen IT-Diensten kommen Cloud- beziehungsweise Virtual-Private-Server-Systeme (VPS), mobile Arbeitsformen, On-Premises-Komponenten und Geräte der Operational Technology (OT) wie Sensoren oder steuernde Komponenten hinzu. Dadurch entstehen mehr Kommunikationsbeziehungen und mehr potenzielle Angriffspunkte.

Traditionelle Sicherheitsarchitekturen schützen solche Umgebungen häufig über einen Netzwerk-perimeter. Dabei gelten interne Netze oder VPN-Zugänge stärker als vertrauenswürdig, während externe Zugriffe durch Firewalls, Routen und veröffentlichte Ports begrenzt werden. Dieses Modell bleibt in vielen Organisationen verbreitet, stösst aber bei Cloud-Diensten, Homeoffice, IoT und gemischten IT-/OT-Umgebungen an Grenzen. Wenn ein Angreifer gültige Zugangsdaten, einen Endpoint oder einen Server kompromittiert, kann der weitere Schaden stark davon abhängen, welche Netze und Dienste aus dieser Position erreichbar sind.

Die Arbeit untersucht deshalb, welche Vorteile eine Zero-Trust-Architektur in einer heterogenen Netzwerkumgebung mit IT- und OT-Geräten gegenüber einer perimeterbasierten Sicherheitsarchitektur bietet. Im Zentrum steht nicht nur die theoretische Betrachtung, sondern ein praktischer Vergleich in einer Laborumgebung.

Ziel der Arbeit

Ziel der Arbeit war der Aufbau einer vergleichbaren Laborumgebung, in der beide Sicherheitsansätze anhand derselben fachlichen Ressourcen, Benutzerrollen und Szenarien untersucht werden können. Die Umgebung umfasst Cloud- beziehungsweise VPS-Dienste, einen On-Premises-Server, Benutzerrollen, Monitoring, Raspberry-Pi-basierte Sensorik und eine MQTT-gesteuerte Lüfter-Simulation. Der physische Laboraufbau ist in Abbildung 1 dargestellt.



Abbildung 1.: Physischer Laboraufbau mit den Client-Arbeitsplätzen der Benutzerrollen Alice und Bob sowie der Lüfter-Simulation auf einer LED-Matrix

Vorgehen und Technologien

Die Zero-Trust-Variante wurde ausschliesslich mit Open-Source-Komponenten umgesetzt. Dadurch bleiben Architektur, Konfigurationen und Schnittstellen nachvollziehbar und reproduzierbar.

Keycloak übernimmt die zentrale Authentifizierung und Rollenverwaltung. OpenZiti stellt den identitätsbasierten Netzwerkzugriff bereit. oauth2-proxy verbindet HTTP-Anwendungen mit der Anmeldung über Keycloak. Envoy setzt Zugriffsentscheide im Datenpfad durch. Open Policy Agent bewertet Anfragen anhand definierter Policies. Wazuh ergänzt die Umgebung um Sicherheitsüberwachung und bildet zusammen mit der Wazuh-OPA-Bridge die Grundlage für dynamische Zugriffseinschränkungen.

Der wesentliche Unterschied zur perimeterbasierten Variante liegt im Vertrauensmodell. In der Zero-Trust-Architektur entsteht Zugriff nicht durch die Zugehörigkeit zu einem internen Netz oder VPN, sondern durch explizite Berechtigungen für eine konkrete Identität und einen konkreten Dienst. Abbildung 2 zeigt vereinfacht, wie Netzwerkzugriff, Benutzeranmeldung und Policy-Entscheid in einem mehrstufigen Zugriffspfad zusammenwirken: Erst wenn alle Stufen zustimmen, erreicht eine Anfrage die eigentliche Anwendung.

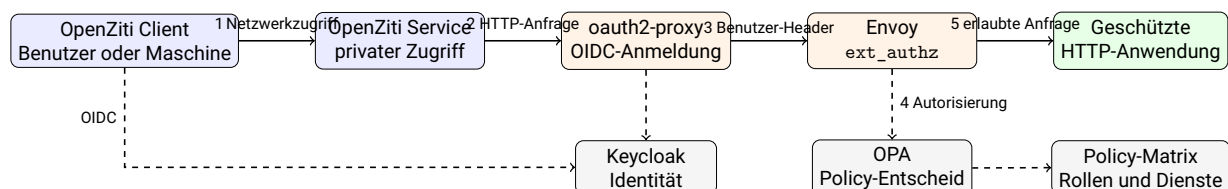


Abbildung 2.: Vereinfachter Zugriffspfad der Zero-Trust-Architektur: Eine Anfrage durchläuft den privaten Netzwerkzugriff (OpenZiti), die Benutzeranmeldung (Keycloak via oauth2-proxy) und den Richtlinienentscheid (OPA), bevor sie die Anwendung erreicht

Als Vergleich diene eine perimeterbasierte WireGuard-/VPN-Architektur mit veröffentlichten Diensten. In dieser Variante werden Zugriffe über WireGuard-Peers, AllowedIPs, Routingtabellen, Firewallzustand und Host-Ports beschrieben. Für weitere Perimeterkomponenten wie dedizierte Firewall-Appliances, VLANs oder DMZ-Segmente lagen in der dokumentierten Umsetzung keine konkreten Implementierungsartefakte vor.

Umsetzung

Die praktische Umgebung wurde so aufgebaut, dass klassische IT-Dienste und OT-nahe Komponenten gemeinsam betrachtet werden können. Zu den IT-Diensten gehören unter anderem Intranet, Accounting und Uptime Kuma. Die OT-nahe Umgebung umfasst ein Dashboard, MQTT-Kommunikation, einen DHT11-/Modbus-Sensor und die Lüfter-Simulation. Abbildung 3 zeigt das OT-Dashboard, über das Sensordaten angezeigt und der simulierte Lüfter gesteuert werden.

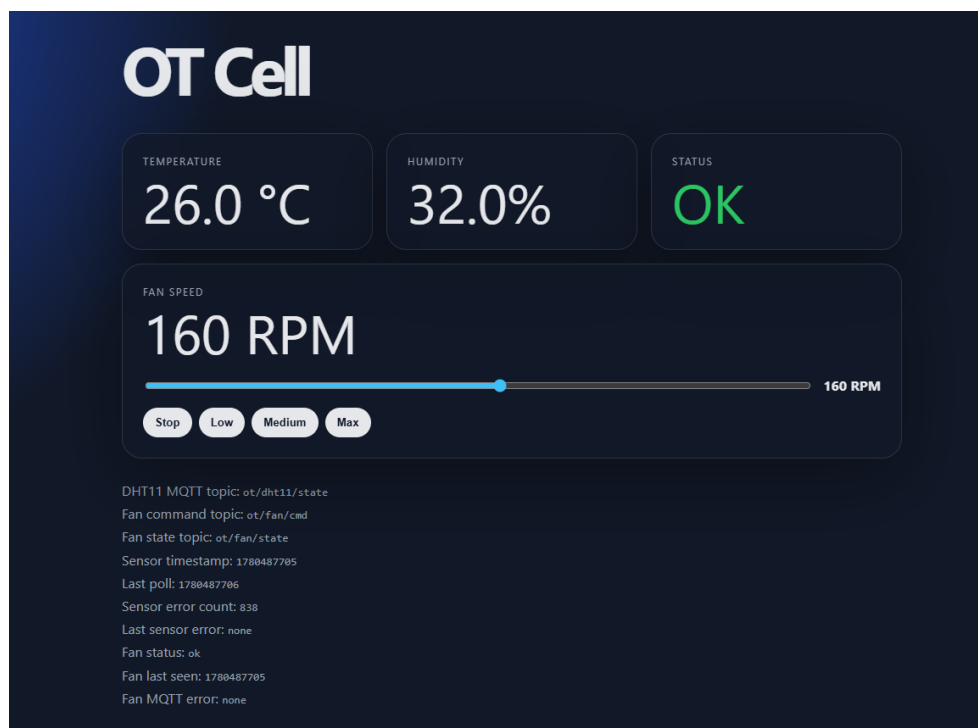


Abbildung 3.: OT-Dashboard der Laborumgebung zur Anzeige der Sensordaten (Temperatur, Luftfeuchtigkeit) und zur Steuerung der Lüfter-Simulation

Für die Zero-Trust-Architektur wurden Dienste nicht allgemein im Netzwerk exponiert. Stattdessen wurden sie als OpenZiti-Services mit passenden Identitäten, Rollen und Policies modelliert. Benutzer wie `bob`, `service-technician` oder `ops-admin` erhalten nur die Dienste, die ihrer Rolle ausdrücklich zugewiesen sind. Für HTTP-Dienste kommt zusätzlich eine anwendungsbezogene Prüfung über `oauth2-proxy`, `Envoy` und `OPA` hinzu. Nicht-HTTP-Dienste wie MQTT, Modbus, RDP oder SSH werden ebenfalls über OpenZiti-Services abgebildet.

Die perimeterbasierte Vergleichsarchitektur wurde als WireGuard-basierte Variante umgesetzt. Dort stehen VPN-Peers, IP-Bereiche, Routen, veröffentlichte Ports und Hostzustände im Vordergrund. Dadurch entsteht ein anderes Betriebsmodell: Änderungen können technisch einfacher sein, sind aber stärker an Netzwerk- und Portfreigaben gebunden.

Ergebnisse

Die Evaluation umfasste vier Angriffsszenarien und drei Wartungsszenarien. Die Angriffsszenarien betrachteten kompromittierte Zugangsdaten, einen kompromittierten Endpoint, einen kompromittierten Diensthost und den Missbrauch eines legitimen Wartungszugangs. Die Wartungsszenarien prüften das Hinzufügen eines neuen Clients, eines neuen Cloud-Dienstes und eines neuen OT-Sensors.

Das zentrale Ergebnis ist, dass die Zero-Trust-Architektur die erreichbare Angriffsfläche in allen untersuchten Angriffsszenarien auf explizit freigegebene Dienste begrenzte. Die kompromittierte Identität `bob` sah im Phishing-Szenario (A1) und im Endpoint-Szenario (A2) nur die Dienste `intranet`, `ot-dashboard` und `uptime-kuma-direct`. Nicht zugewiesene Dienste wie `accounting`, `mqtt-broker`, `ot-dht11-modbus`, `guacamole-svc`, `win-rdp` und die SSH-Services waren nicht erreichbar. Abbildung 4 veranschaulicht dieses Prinzip aus Benutzersicht: Der Client zeigt ausschliesslich die für die angemeldete Identität freigegebenen privaten Dienste an.

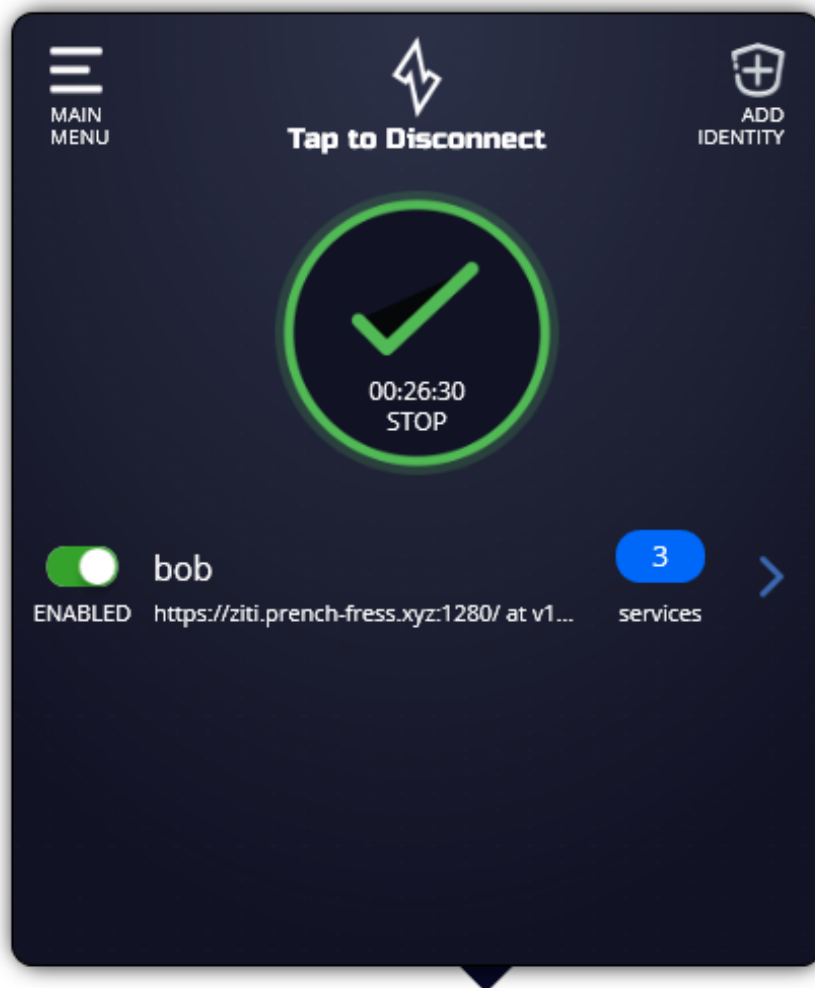


Abbildung 4.: OpenZiti Desktop Edge Client aus Benutzersicht: Sichtbar sind ausschliesslich die der angemeldeten Identität explizit freigegebenen privaten Dienste

Beim Szenario mit kompromittiertem Diensthost zeigte sich ebenfalls eine klare Begrenzung. Die Host-Identität `intranet-host` besass keine Dial-Policies und konnte über OpenZiti keine weite-

ren Dienste erreichen. Ein Angreifer konnte damit über diese Identität keine laterale Bewegung zu anderen Ziti-Diensten durchführen. Beim Szenario mit einem legitimen, aber missbräuchlich verwendeten Service-Techniker waren exakt die vier der Rolle zugewiesenen OT-nahen Dienste sichtbar: `guacamole-svc`, `mqtt-broker`, `ot-dashboard` und `ot-dht11-modbus`. IT- und Administrationsdienste erschienen nicht.

Auch die Eindämmung war in der Zero-Trust-Variante gezielt möglich. Benutzer- oder Host-Identitäten konnten selektiv blockiert werden, ohne ganze Netzbereiche oder grössere Dienstgruppen abzuschalten. In den dokumentierten Läufen dauerte die Eindämmung ab Eingang der Meldung ungefähr eine Sekunde. Die Restore-Schritte entfernten die Blockierungen wieder vollständig, und andere Identitäten wurden nicht beeinträchtigt. Kein nachweisbarer Vorteil ergab sich dagegen bei der Erkennungsgeschwindigkeit: Die initiale Erkennung hing in beiden Umgebungen vom eingesetzten Endpoint-Schutz ab und nicht von der Netzwerkarchitektur.

Die perimeterbasierten Vergleichsläufe zeigten demgegenüber ein gröberes Modell. Der Zugriff wurde über WireGuard-Peers, AllowedIPs, Routen, veröffentlichte Ports und Hostzustände beschrieben. Ein dokumentierter Peer hatte AllowedIPs für mehrere private /24-Netze. Eine servicegenaue Eindämmung einzelner Identitäten wurde in dieser Architektur nicht festgestellt. Mögliche Massnahmen liegen eher bei Peer-Entfernung, Routenänderung, Portschliessung, Containerstopp oder Firewall-Regeländerung.

Bei den Wartungsszenarien zeigte sich ein Zielkonflikt. Zero Trust erforderte mehr explizite Verwaltungsobjekte, beispielsweise Identitäten, Enrollment-Dateien, Services, Configs sowie Bind- und Dial-Policies. Die Änderungen dauerten in den dokumentierten Läufen rund zwei bis drei Sekunden und konnten vollständig zurückgerollt werden. Gleichzeitig blieb der Zugriff präzise begrenzt: Neue Clients oder Dienste erweiterten bestehende Benutzerrollen nicht unbeabsichtigt. Die perimeterbasierten Wartungsänderungen waren technisch einfacher, gingen jedoch mit gröberer Zugriffskontrolle einher, weil Freigaben über Peer-, IP- oder Portkontext und nicht über einzelne dienstbezogene Identitäten modelliert wurden.

Bedeutung für die Praxis

Für Organisationen mit gemischten IT-/OT-Umgebungen zeigen die Ergebnisse, dass eine Zero-Trust-Architektur den möglichen Schaden nach einer Kompromittierung strukturell begrenzen kann: Gestohlene Zugangsdaten, kompromittierte Endgeräte oder missbrauchte Wartungszugänge führen nicht automatisch zu weitreichendem Netzwerkzugriff. Zudem lassen sich Vorfälle gezielt und schnell eindämmen, ohne den übrigen Betrieb zu unterbrechen, und die identitätsbezogene Protokollierung erleichtert die Aufarbeitung von Vorfällen. Die Umsetzung mit Open-Source-Komponenten zeigt darüber hinaus, dass eine solche Architektur ohne kostenintensive proprietäre Enterprise-Produkte realisierbar ist.

Aus den Ergebnissen lassen sich, im Rahmen der Laborbedingungen, folgende praktische Hinweise ableiten:

- Zero Trust lohnt sich insbesondere dort, wo granulare Zugriffskontrolle, externe Wartungszugänge, kritische OT-Komponenten oder hohe Anforderungen an Nachvollziehbarkeit bestehen.
- Der höhere Einrichtungs- und Pflegeaufwand ist einzuplanen: Identitäten, Dienste und Policies müssen explizit modelliert und laufend gepflegt werden. Dieser Aufwand ist zugleich die Grundlage der beobachteten Sicherheitsvorteile.

- Zero Trust ersetzt weder Endpoint-Schutz noch sichere Anwendungen. Die Erkennungsgeschwindigkeit hing in der Evaluation vom Endpoint-Schutz ab, nicht von der Netzwerkarchitektur.
- Perimeterbasierte Architekturen können viele der beobachteten Schutzwirkungen mit zusätzlichen Massnahmen wie Segmentierung, Host-Firewalls oder Network Access Control nachrüsten. Bei Zero Trust ist die restriktive Grundhaltung dagegen Standardverhalten.

Einordnung und Ausblick

Die Arbeit bestätigt für die untersuchte Laborumgebung vor allem Vorteile von Zero Trust bei der Reduktion der Angriffsfläche, der Begrenzung lateraler Bewegung nach einer Erstkompromittierung, der gezielten Eindämmung, der Nachvollziehbarkeit von Angriffspfaden und der kontrollierten Anpassbarkeit. Kein nachweisbarer Vorsprung zeigte sich bei der Erkennungsgeschwindigkeit sowie beim Schutz kritischer OT-Ressourcen, wo auch die perimeterbasierte Variante mit Netzsegmentierung eine wirksame, wenn auch gröbere Schutzwirkung erzielte. Der perimeterbasierte Ansatz bleibt bei einfachen technischen Änderungen weniger aufwendig, bietet aber eine gröbere Zugriffskontrolle und hängt stärker von Netzsegmenten, Routen und Ports ab.

Die Ergebnisse sind durch den Laborcharakter begrenzt. Die Szenarien wurden kontrolliert und nicht destruktiv durchgeführt. Es wurden keine realen Phishing-Kampagnen, Malware-Infektionen oder produktiven OT-Manipulationen ausgeführt. Zudem wurde keine vollständige automatische Time to Detect über ein produktives Alarmereignis gemessen. Die Aussagen zur Time to Contain sind deshalb belastbarer als Aussagen zur automatischen Erkennungszeit.

Für weiterführende Arbeiten bieten sich eine grössere OT-Umgebung, zusätzliche industrielle Protokolle, produktionsnähere Erkennungsszenarien und eine weitergehende dynamische Policy-Anpassung an. Ebenfalls sinnvoll wäre ein breiterer Vergleich mit weiteren Perimeter-Varianten, etwa mit dedizierter Firewall-Appliance, VLANs, DMZ-Segmenten oder industrieller Zonierung.

Inhaltsverzeichnis

Management Summary	ii
1. Einleitung	1
1.1. Ausgangslage und Motivation	1
1.2. Forschungsfrage	2
1.3. Hypothesen	3
1.4. Evaluationskonzept	7
1.5. Bisherige Forschung	10
1.6. Grundlagen von Zero Trust	11
1.7. Perimeterbasierte Sicherheitsansätze	13
2. Konzeption	15
2.1. Zielsetzung der Laborumgebung	15
2.2. Eingesetzte Technologien	18
2.3. Gemeinsame IT/OT-Umgebung	22
2.4. Zero-Trust-Architektur	26
2.5. Perimeterbasierte Architektur	35
3. Umsetzung und Evaluation	39
3.1. OT-Implementierung	39
3.2. Infrastructure as Code und Reproduzierbarkeit	42
3.3. Durchführung der Evaluation	47
3.4. Bewertung der Hypothesen	54
4. Fazit	68
4.1. Beantwortung der Forschungsfrage	68
4.2. Limitationen	70
4.3. Ausblick	71
Literatur	72
Glossar	74
Hilfsmittelverzeichnis	77
A. Projektplan	78
A.1. Projektmanagement	78
A.2. Risikoanalyse	80
B. Selbstreflexion	83
B.1. Selbstreflexion Ivo Cavelti	83
B.2. Selbstreflexion Baruyr Bousnaian	84

C. Evaluationsprotokolle	85
C.1. Protokollvorlage	85
C.2. Protokoll A1 – Phishing und Credential Theft	85
C.3. Protokoll A1 – Perimeterbasierte Architektur	87
C.4. Protokoll A2 – Perimeterbasierte Architektur	88
C.5. Protokoll A3 – Perimeterbasierte Architektur	89
C.6. Protokoll A4 – Perimeterbasierte Architektur	90
C.7. Protokoll B1 – Perimeterbasierte Architektur	91
C.8. Protokoll B2 – Perimeterbasierte Architektur	92
C.9. Protokoll B3 – Perimeterbasierte Architektur	93
C.10. Protokoll A2 – Kompromittierter Endpoint	94
C.11. Protokoll A3 – Exponierter und verwundbarer Cloud-Dienst	97
C.12. Protokoll B1 – Hinzufügen eines neuen Clients	98
C.13. Protokoll B2 – Hinzufügen eines neuen Cloud-Dienstes	99
C.14. Protokoll B3 – Hinzufügen eines neuen Sensors	100
C.15. Protokoll A4 – Insider-Threat durch externen Service-Techniker	101
D. README der Infrastruktur	103
E. Playbook zur Erkundung der Laborumgebung	113
E.1. Allgemeine Voraussetzungen	113
E.2. OpenZiti Desktop Edge Client verwenden	114
E.3. Rolle ops-admin	115
E.4. Rolle alice	116
E.5. Rolle bob	118
E.6. Rolle service-technician	120
E.7. Use Case: Malware-Test mit EICAR-Datei	120
E.8. Zusammenfassende Feature-Tests	122
E.9. Troubleshooting	123

1. Einleitung

Dieses Kapitel führt in die Problemstellung der Arbeit ein und begründet die Relevanz von Zero Trust für heterogene Umgebungen aus Informationstechnik (IT) und Operational Technology (OT). Es formuliert die Forschungsfrage, leitet die Hypothesen ab und beschreibt das Evaluationskonzept. Anschliessend werden der Stand der Forschung sowie die Grundlagen von Zero Trust und perimeterbasierten Sicherheitsarchitekturen dargestellt.

1.1. Ausgangslage und Motivation

Cybersicherheit gehört für Unternehmen unabhängig von ihrem Sektor zu den zentralen Prioritäten. Der Bericht *PwC Global Digital Trust Insights 2025* zeigt, dass im Jahr 2025 rund 67 % der Schweizer Unternehmen planen, ihr Budget für Cybersicherheit zu erhöhen [17]. Gleichzeitig steigen auch die Anforderungen an IT-Systeme und Netzwerkumgebungen. Gemäss Daten des Bundesamts für Statistik (BFS) stieg der Anteil der Erwerbstätigen in der Schweiz, die zumindest gelegentlich im Homeoffice arbeiten, von 27.5 % im Jahr 2017 auf 38.4 % im Jahr 2023 [6]. Dies führt dazu, dass Mitarbeitenden auch ausserhalb der eigenen Unternehmensnetzwerke eine sichere und zuverlässige Arbeitsumgebung zur Verfügung gestellt werden muss.

Auch in der Fertigungsindustrie nimmt die Komplexität von Netzwerken durch die zunehmende Verbreitung von Internet-of-Things-(IoT)-Technologien zu. Gemäss einer von MSM Research durchgeführten und von Swisscom publizierten Studie setzen bereits 18 % der befragten Schweizer Unternehmen IoT-Lösungen produktiv ein. Dabei betreffen viele Anwendungen den Bereich Industrie 4.0 und Produktionsautomatisierung (34 %) sowie Predictive Maintenance (24 %) [22]. Durch den Einsatz von IoT werden Maschinen, Sensoren und Systeme miteinander vernetzt, sodass kontinuierlich Daten ausgetauscht und Prozesse automatisiert werden können. Dadurch steigt die Anzahl vernetzter Geräte und Kommunikationsbeziehungen innerhalb industrieller Umgebungen deutlich an. Infolgedessen entwickeln sich Produktionsnetzwerke zunehmend zu komplexen, hochvernetzten Infrastrukturen, ähnlich wie dies bereits in klassischen IT-Umgebungen im Office-Bereich zu beobachten ist.

Traditionell basierte die Netzwerksicherheit in vielen Organisationen auf einem perimeterbasierten Sicherheitsmodell. In diesem Ansatz wird das interne Netzwerk als vertrauenswürdig betrachtet, während externe Netzwerke durch Schutzmechanismen wie Firewalls oder VPN-Gateways abgesichert werden. In der Praxisliteratur wird dieses Modell häufig als «Castle-and-Moat»-Ansatz bezeichnet. NIST beschreibt entsprechend, dass bei perimeterzentrierten Architekturen starke Schutzmechanismen an der Netzgrenze eingesetzt werden, während innerhalb des Netzwerks ein hohes implizites Vertrauen besteht [13]. Mit der zunehmenden Digitalisierung, der Nutzung von Cloud-Diensten sowie mobilen Arbeitsformen stossen solche perimeterzentrierten Sicherheitsarchitekturen jedoch zunehmend an ihre Grenzen [13, 16]. Obwohl moderne Sicherheitskonzepte wie Zero Trust zunehmend an Bedeutung gewinnen, befinden sich viele Organisationen noch in einer Übergangsphase. Eine Studie von Cisco zeigt, dass zwar ein grosser Teil der Unternehmen bereits mit der Einführung von Zero-Trust-Strategien begonnen hat, jedoch nur ein kleiner Anteil

eine vollständige Umsetzung erreicht hat [2]. Dies deutet darauf hin, dass perimeterbasierte Sicherheitsmodelle weiterhin weit verbreitet sind und in vielen Organisationen nach wie vor einen zentralen Bestandteil der Netzwerksicherheitsarchitektur darstellen.

Vor diesem Hintergrund untersucht diese Arbeit, wie Zero Trust in einer heterogenen Umgebung umgesetzt werden kann, die aus Cloud-Diensten, IoT-Geräten, Clients und lokalen Diensten besteht. In der Gegenüberstellung mit einem perimeterbasierten Ansatz soll analysiert werden, welche Vorteile Zero Trust in einem solchen Umfeld bietet und welche potenziellen Schwächen bestehen können. Zu diesem Zweck wird eine Laborumgebung aufgebaut, in der beide Ansätze implementiert und miteinander verglichen werden. Ziel ist es, die Unterschiede praxisnah aufzuzeigen und die technologischen Grundlagen beider Sicherheitsansätze zu evaluieren. Darüber hinaus soll die Arbeit als Orientierung dienen und Organisationen bei der Umstellung auf eine Zero-Trust-Architektur unterstützen.

1.2. Forschungsfrage

In diesem Abschnitt werden die zentrale Forschungsfrage der Arbeit sowie die daraus abgeleiteten Teilfragen formuliert, die den inhaltlichen und methodischen Rahmen der Arbeit bestimmen.

1.2.1. Zentrale Forschungsfrage

Die zentrale Forschungsfrage dieser Arbeit lautet:

Welche Vorteile bietet eine Zero-Trust-Architektur in einer heterogenen Netzwerkumgebung mit IT- und OT-Geräten im Vergleich zu einer perimeterbasierten Sicherheitsarchitektur?

1.2.2. Abgeleitete Teilfragen

Zur systematischen Beantwortung der zentralen Forschungsfrage werden folgende Teilfragen untersucht:

1. Theoretische Grundlagen

- 1.1** Welche konzeptionellen Prinzipien und Kernkomponenten kennzeichnen eine Zero-Trust-Architektur?
- 1.2** In welchen wesentlichen Aspekten unterscheidet sich eine Zero-Trust-Architektur von klassischen perimeterbasierten Sicherheitsarchitekturen?

2. Stand der Technik

- 2.1** Welche Referenzmodelle, Frameworks und Architekturansätze für Zero Trust sind derzeit etabliert?
- 2.2** Welche Technologien und Werkzeuge werden aktuell zur Umsetzung von Zero-Trust-Architekturen eingesetzt?

3. Konzeption für ein heterogenes Netzwerk

3.1 Welche besonderen sicherheitstechnischen Anforderungen ergeben sich bei der Absicherung eines heterogenen Netzwerks aus IT- und OT-Geräten?

3.2 Wie kann auf Grundlage dieser Anforderungen eine geeignete Zero-Trust-Architektur modelliert werden?

4. Praktische Umsetzung im Laborszenario

4.1 Wie lässt sich die konzipierte Zero-Trust-Architektur in einer Laborumgebung technisch implementieren?

4.2 Welche Herausforderungen treten bei der praktischen Umsetzung auf?

5. Evaluation und Vergleich

5.1 Anhand welcher Kriterien kann die Wirksamkeit der implementierten Zero-Trust-Architektur bewertet werden?

5.2 Welche sicherheitstechnischen Vor- und Nachteile zeigen sich im Vergleich zu einer perimeterbasierten Architektur im identischen Laborszenario?

1.3. Hypothesen

Um einen strukturierten Vergleich zwischen einer Zero-Trust-Architektur und einer perimeterbasierten Sicherheitsarchitektur zu ermöglichen, werden im Folgenden zehn Hypothesen formuliert. Diese dienen als Grundlage für die Bewertung und den Vergleich der beiden Ansätze im weiteren Verlauf der Arbeit. Der Abschnitt leistet damit einen Beitrag zur Beantwortung von Teilfrage 5.1, indem er die später verwendeten Bewertungskriterien vorbereitet. Die in den Bewertungskriterien genannten technischen Artefakte, etwa OpenZiti-Services, Dial-Policies oder WireGuard-AllowedIPs, beziehen sich auf die in Kapitel 2 eingeführten Technologien und Architekturen.

1.3.1. Sicherheitsbezogene Hypothesen

1.3.1.1. H1 – Verringerung der Angriffsfläche

Beschreibung: Eine Zero-Trust-Architektur reduziert die effektiv erreichbare Angriffsfläche in einer heterogenen Netzwerkumgebung stärker als ein perimeterbasierter Ansatz, da Dienste, Ports und Kommunikationsbeziehungen restriktiver freigegeben werden.

Bewertungskriterium:

- Anzahl und Art der aus Angreiferperspektive sichtbaren beziehungsweise erreichbaren Dienste
- Ziti-Service-Sichtbarkeit und vorhandene Dial-Policies in der Zero-Trust-Architektur
- veröffentlichte Ports, WireGuard-AllowedIPs, Routingtabellen und TCP-Erreichbarkeit in der perimeterbasierten Architektur

Erwartung: Unter Zero Trust sind weniger Systeme und Dienste erreichbar.

1.3.1.2. H2 – Schnellere Erkennung von Angriffsaktivitäten

Beschreibung: Eine Zero-Trust-Architektur ermöglicht die schnellere Erkennung verdächtiger Aktivitäten, da Zugriffe kontinuierlich überprüft, protokolliert und kontextbezogen bewertet werden.

Bewertungskriterium:

- vorhandene Identitäts-, Zugriffs- und Bridge-Artefakte zu verdächtigen Aktivitäten
- Zeitpunkt einer ersten Alarmierung beziehungsweise eines auswertbaren Alarmereignisses, falls vorhanden
- Vergleich der verfügbaren Identitätsdetektion zwischen Zero Trust und perimeterbasierter Architektur

Erwartung: Die Erkennungszeit ist unter Zero Trust kürzer.

1.3.1.3. H3 – Schnellere und gezieltere Eindämmung

Beschreibung: Angriffe können schneller und gezielter eingedämmt werden, da kompromittierte Identitäten oder Systeme selektiv isoliert werden können.

Bewertungskriterium:

- *Time to Contain* (TTC) zwischen Containment-Start und bestätigtem Block beziehungsweise deaktivierter Identität
- Anzahl und Art der betroffenen Identitäten, Server oder Dienste
- Granularität der Eindämmungsmassnahme, insbesondere Identitäts-/Service-Block gegenüber Peer-, Port-, Routen- oder Firewall-Massnahmen
- Kollateraleffekt auf nicht betroffene Systeme und legitime Zugriffe

Erwartung: Die Eindämmung erfolgt schneller und mit geringerem Kollateraleffekt.

1.3.1.4. H4 – Höhere Nachvollziehbarkeit des Angriffspfads

Beschreibung: Angriffsaktivitäten lassen sich in einer Zero-Trust-Architektur besser rekonstruieren, da Zugriffe detailliert und identitätsbezogen protokolliert werden.

Bewertungskriterium:

- Vollständigkeit der dokumentierten Authentifizierungs-, Session-, Blockierungs- und Restore-Artefakte
- Zuordenbarkeit zu Benutzer, Identität, Dienst, Policy-Entscheidung und Zeitpunkt
- Vergleich der verfügbaren Zero-Trust-Artefakte mit Netz-, Host-, Port- und Routingartefakten der perimeterbasierten Architektur

Erwartung: Der Angriffspfad ist unter Zero Trust vollständiger nachvollziehbar.

1.3.1.5. H5 – Keine unverhältnismässige Einschränkung des legitimen Betriebs

Beschreibung: Die Sicherheit wird erhöht, ohne den legitimen Betrieb unverhältnismässig einzuschränken.

Bewertungskriterium:

- durch Containment betroffene Dienste, Identitäten und Kommunikationsbeziehungen
- unveränderte Sichtbarkeit und Erreichbarkeit nicht betroffener legitimer Dienste
- Vollständigkeit der Wiederherstellung nach dem Restore-Schritt
- qualitative Bewertung möglicher Kollateraleffekte in der perimeterbasierten Architektur

Erwartung: Keine unverhältnismässige Einschränkung des Betriebs.

1.3.1.6. H6 – Begrenzung des Angriffserfolgs nach Erstkompromittierung

Beschreibung: Nach einem initialen Eindringen ist der weitere Angriffserfolg unter Zero Trust geringer, da kompromittierte Identitäten nur auf explizit freigegebene Dienste zugreifen können.

Bewertungskriterium:

- Anzahl und Art der nach Erstkompromittierung sichtbaren beziehungsweise erreichbaren Dienste
- Umfang der für die kompromittierte Identität oder Host-Identität vorhandenen Dial-Policies
- vorhandenes Pivot-Potenzial zu weiteren Diensten oder Netzbereichen
- Vergleich mit perimeterbasierten Netzpfaden, AllowedIPs, Ports und tatsächlich erreichbaren Zielsystemen

Erwartung: Der Angriffserfolg ist unter Zero Trust geringer.

1.3.1.7. H7 – Schutz kritischer OT-Ressourcen

Beschreibung: OT-Ressourcen (Sensoren, Maschinen etc.) werden wirksamer vor unautorisierten Zugriffen geschützt.

Bewertungskriterium:

- Sichtbarkeit und Erreichbarkeit von OT-Diensten für unautorisierte oder rollenbeschränkte Identitäten
- Übereinstimmung der sichtbaren OT-Dienste mit den explizit vorgesehenen Rollen- und Service-Policies
- Ausschluss von IT- oder Administrationsdiensten aus OT-Wartungskontexten
- perimeterbasierte Exposition von OT-Ressourcen über Netze, Ports, Firewallzustand und lokale Erreichbarkeit

Erwartung: Zero Trust bietet höheren Schutz kritischer Systeme.

1.3.2. Betriebsbezogene Hypothesen

1.3.2.1. H8 – Höherer Einrichtungsaufwand bei Zero Trust

Beschreibung: Die Implementierung einer Zero-Trust-Architektur erfordert einen höheren initialen Aufwand als ein perimeterbasierter Ansatz, da Identitäten, Dienste und Policies explizit modelliert werden müssen.

Bewertungskriterium:

- Zeit zwischen change-start und change-end
- Anzahl und Art temporär erzeugter Verwaltungsobjekte, insbesondere Identitäten, Services, Configs sowie Bind- und Dial-Policies
- Umfang der Konfigurations- beziehungsweise Laufzeitzustandsänderung
- Vergleich mit perimeterbasierten Änderungen an WireGuard-Peers, AllowedIPs, veröffentlichten Ports und Containern

Erwartung: Der Einrichtungsaufwand ist höher.

1.3.2.2. H9 – Höherer Betriebs- und Pflegeaufwand

Beschreibung: Der laufende Betrieb einer Zero-Trust-Architektur ist aufwendiger, da zusätzliche Verwaltungsobjekte wie Identitäten, Policies, Services und Konfigurationen kontinuierlich gepflegt werden müssen.

Bewertungskriterium:

- Anzahl zusätzlicher Identitäten, Policies, Services und Configs, die im Betrieb gepflegt werden müssen
- Aufwand für Policy-, Benutzer-, Geräte- und Service-Anpassungen
- Nachvollziehbarkeit und Reversibilität der Änderungen über Baseline-, Verify- und Post-Restore-Artefakte
- perimeterbasierter Pflegeaufwand für Peer-, IP-, Port-, Firewall- und Routendokumentation

Erwartung: Der Betriebsaufwand ist höher.

1.3.2.3. H10 – Höhere Anpassungsfähigkeit trotz Komplexität

Beschreibung: Zero Trust ermöglicht präzisere Anpassungen trotz höherer Komplexität.

Bewertungskriterium:

- Granularität der Änderung auf Identitäts-, Dienst- und Policy-Ebene
- Auswirkung neuer Clients, Dienste oder OT-Sensoren auf bestehende Dienstsichtbarkeit und Kommunikationsbeziehungen
- Anzahl unbeabsichtigter Zugriffserweiterungen oder Nebeneffekte
- Vollständigkeit des Restore-Schritts und Entfernung temporärer Objekte

- Vergleich mit perimeterbasierten Änderungen über Peer-, IP- und Portfreigaben

Erwartung: Höhere Flexibilität bei kontrollierter Anpassbarkeit.

1.4. Evaluationskonzept

In diesem Abschnitt wird das methodische Vorgehen zur Evaluation der in Abschnitt 1.3 formulierten Hypothesen beschrieben. Ziel ist es, die Wirksamkeit einer Zero-Trust-Architektur im Vergleich zu einer klassischen perimeterbasierten Sicherheitsarchitektur systematisch und nachvollziehbar zu untersuchen. Der Abschnitt adressiert Teilfrage 5.1, indem er die Kriterien und Szenarien zur Bewertung der implementierten Architektur festlegt.

Die Evaluation erfolgt aus zwei komplementären Perspektiven:

- sicherheitsbezogene Bewertung anhand definierter Angriffsszenarien
- betriebliche Bewertung anhand typischer Wartungs- und Änderungsprozesse

Für beide Perspektiven werden identische Szenarien in vergleichbaren Umgebungen durchgeführt und anhand der zuvor definierten Bewertungskriterien analysiert. Die in diesem Abschnitt definierten Szenarien bilden die verbindliche Grundlage für die praktische Durchführung. Deren konkrete Umsetzung in beiden Architekturen wird in Abschnitt 3.3 dokumentiert.

1.4.1. Evaluationsmethodik

Jedes Szenario wird sowohl innerhalb der Zero-Trust-Architektur als auch in der perimeterbasierten Architektur durchgeführt. Die Durchführung erfolgt unter möglichst identischen Rahmenbedingungen, um eine hohe Vergleichbarkeit der Ergebnisse sicherzustellen. Eine bekannte Asymmetrie besteht darin, dass nur die Zero-Trust-Umgebung über ein SIEM (Security Information and Event Management) verfügt; diese Einschränkung wird bei der Bewertung der Erkennungshypothese H2 berücksichtigt.

Die Auswertung orientiert sich an den in Abschnitt 1.3 definierten Kriterien. Dabei werden insbesondere folgende Aspekte berücksichtigt:

- quantitative Metriken (z. B. Zeitaufwand, Anzahl beteiligter Systeme, Anzahl detektierter Ereignisse)
- qualitative Beobachtungen (z. B. Nachvollziehbarkeit von Zugriffen, Komplexität der Umsetzung)

Die erhobenen Ergebnisse werden anschliessend systematisch gegenübergestellt und zur Bewertung der jeweiligen Hypothesen herangezogen.

1.4.2. Angriffsszenarien (Sicherheitsbewertung)

Zur Evaluation der sicherheitsbezogenen Hypothesen H1 bis H7 werden vier praxisnahe Angriffsszenarien definiert. Die Auswahl basiert auf empirischen Untersuchungen zu Cyberangriffen. Studien wie der *Verizon Data Breach Investigations Report* sowie Analysen von ENISA und Mandiant zeigen, dass erfolgreiche Angriffe typischerweise auf wenige zentrale Eintrittsvektoren zurückzuführen sind [23, 5, 9]. Die vier Szenarien greifen diese Eintrittsvektoren auf: gestohlene

Zugangsdaten und Phishing (A1), kompromittierte Endgeräte (A2), die Ausnutzung exponierter Anwendungen (A3) sowie der Missbrauch legitimer Zugriffsrechte (A4).

Die gewählten Szenarien repräsentieren somit unterschiedliche Angreiferperspektiven:

- Angriffe auf Benutzeridentitäten
- Angriffe auf Endgeräte
- Angriffe auf öffentlich erreichbare Systeme
- Missbrauch legitimer Zugriffsrechte

Diese Kombination ermöglicht eine breit angelegte und praxisorientierte Bewertung der Sicherheitsmechanismen beider Architekturen.

1.4.2.1. Szenario A1 – Phishing und Credential Theft

Beschreibung: Ein Mitarbeitender wird im Rahmen einer Phishing-Kampagne zur Eingabe seiner Zugangsdaten auf einer manipulierten Webseite verleitet. Der Angreifer verwendet die durch Credential Theft erlangten Zugangsdaten, um sich legitim wirkend an zentralen Systemen (z. B. Cloud-Diensten oder VPN) anzumelden.

Ziel des Angreifers:

- Übernahme von Benutzerkonten (Account Takeover)
- Zugriff auf Cloud-Dienste und interne Systeme
- Umgehung klassischer Perimeter-Schutzmechanismen

1.4.2.2. Szenario A2 – Kompromittierter Endpoint

Beschreibung: Ein Endpoint eines Mitarbeitenden wird durch Schadsoftware kompromittiert, beispielsweise durch das Öffnen eines präparierten Dokuments oder den Download einer manipulierten Datei. Der Angreifer erhält dadurch Kontrolle über das System im Kontext des Benutzers.

Ziel des Angreifers:

- Etablierung von Persistenz auf dem Endpoint
- Auslesen lokaler Zugangsdaten
- Laterale Bewegung im Netzwerk
- Eskalation von Berechtigungen

1.4.2.3. Szenario A3 – Exponierter und verwundbarer Cloud-Dienst

Beschreibung: Ein öffentlich erreichbarer Cloud-Dienst (z. B. eine Webanwendung oder API) weist eine Sicherheitslücke oder Fehlkonfiguration auf. Ein Angreifer identifiziert diese Schwachstelle und nutzt sie gezielt aus, um Zugriff auf interne Ressourcen oder sensible Daten zu erlangen.

Ziel des Angreifers:

- Ausnutzung einer technischen Schwachstelle (z. B. Remote Code Execution)

- Zugriff auf Backend-Systeme oder Datenbanken
- Etablierung von Persistenz innerhalb der Cloud-Umgebung
- Weitergehender Zugriff auf interne Netzbereiche

1.4.2.4. Szenario A4 – Insider Threat durch externen Service-Techniker

Beschreibung: Ein externer Service-Techniker erhält im Rahmen von Wartungsarbeiten legitimen Zugriff auf Teile der Infrastruktur (z. B. über VPN oder vor Ort). Dieser Zugriff wird jedoch missbräuchlich genutzt, um auf Systeme ausserhalb des vorgesehenen Aufgabenbereichs zuzugreifen.

Ziel des Angreifers:

- Zugriff auf nicht autorisierte Systeme
- Ausweitung bestehender Berechtigungen
- Manipulation oder Exfiltration von Daten
- Etablierung eines dauerhaften Zugangs (Backdoor)

1.4.3. Wartungsszenarien (Betriebsbewertung)

Zur Evaluation der betriebsbezogenen Hypothesen H8 bis H10 werden typische Änderungs- und Erweiterungsszenarien betrachtet, wie sie in realen IT- und OT-Umgebungen regelmässig auftreten.

1.4.3.1. Szenario B1 – Hinzufügen eines neuen Clients

Beschreibung: Ein neuer Client wird in die bestehende Netzwerkumgebung integriert und für den regulären Betrieb konfiguriert.

Ziel der Evaluation:

- Bewertung des Einrichtungsaufwands
- Analyse der erforderlichen Konfigurationsschritte
- Untersuchung möglicher Auswirkungen auf bestehende Systeme

Relevante Hypothesen: H8, H9, H10

1.4.3.2. Szenario B2 – Hinzufügen eines neuen Cloud-Dienstes

Beschreibung: Ein zusätzlicher Cloud-Dienst wird in die bestehende Architektur integriert und mit internen Systemen verbunden.

Ziel der Evaluation:

- Bewertung der Integrationskomplexität
- Analyse der erforderlichen Sicherheitskonfigurationen

- Untersuchung der Auswirkungen auf bestehende Kommunikationsbeziehungen

Relevante Hypothesen: H8, H9, H10

1.4.3.3. Szenario B3 – Hinzufügen eines neuen Sensors (OT-Komponente)

Beschreibung: Ein neuer Sensor wird in die OT-Umgebung integriert und in bestehende Kommunikationsstrukturen eingebunden.

Ziel der Evaluation:

- Bewertung des Integrationsaufwands in einer heterogenen Umgebung
- Analyse der erforderlichen Sicherheitsfreigaben
- Untersuchung potenzieller Auswirkungen auf bestehende OT-Komponenten

Relevante Hypothesen: H8, H9, H10

1.4.4. Auswertung und Vergleich

Die Ergebnisse der einzelnen Szenarien werden systematisch erfasst und den jeweiligen Hypothesen zugeordnet. Für jede Hypothese erfolgt eine strukturierte Bewertung auf Basis der definierten Kriterien.

Anschliessend werden die Resultate beider Architekturen gegenübergestellt, um:

- Unterschiede in der Sicherheitswirkung zu identifizieren
- betriebliche Auswirkungen zu bewerten
- Stärken und Schwächen der jeweiligen Ansätze herauszuarbeiten

Die abschliessende Analyse dient dazu, die zentrale Forschungsfrage dieser Arbeit fundiert zu beantworten.

1.5. Bisherige Forschung

In der aktuellen Forschung existieren verschiedene Ansätze, um den steigenden Sicherheitsanforderungen in industriellen Netzwerken mithilfe von Zero-Trust-Architekturen zu begegnen. Trotz unterschiedlicher technischer Umsetzungen verfolgen diese Arbeiten ein gemeinsames Ziel: die konsequente Anwendung der zentralen Zero-Trust-Prinzipien, insbesondere den vollständigen Verzicht auf implizites Vertrauen innerhalb eines Netzwerks. Ein besonderer Schwerpunkt liegt dabei häufig auf dem Schutz von Legacy-Systemen, die ursprünglich nicht für vernetzte Umgebungen entwickelt wurden. Der Abschnitt adressiert Teilfrage 2.1, indem er etablierte Architekturansätze und Referenzbezüge aus der Forschung einordnet.

Die Unterschiede zwischen den Ansätzen zeigen sich vor allem in der konkreten Architektur sowie im Grad der Netzwerksegmentierung.

Zanasi et al. argumentieren, dass klassische Mikrosegmentierung für industrielle Umgebungen nicht ausreicht, da dabei weiterhin kleine Netzwerkbereiche mit implizitem Vertrauen entstehen. Stattdessen schlagen sie eine gerätespezifische Segmentierung («per-device segmentation»)

vor. In diesem Ansatz wird jede Kommunikation zwischen einzelnen Geräten durch eine Next Generation Firewall überwacht und autorisiert [25].

Federici et al. verfolgen hingegen einen Ansatz mit Fokus auf hoher Skalierbarkeit. Sie unterteilen die Zero-Trust-Architektur in zwei Abstraktionsebenen: eine Netzwerk-Ebene und eine Edge-Ebene. Dieses Modell entspricht einer zweistufigen Autorisierung. Während der Zugriff auf der Netzwerkebene durch Software-Defined Networking (SDN) verwaltet wird, kommen bei den Edge-Geräten hardwarebasierte Trusted Execution Environments (TEE) zum Einsatz. Dadurch wird eine sehr feingranulare Zugriffskontrolle ermöglicht, bei der beispielsweise einzelne Speicherbereiche oder Peripheriegeräte gezielt abgesichert werden können [7].

Raich et al. präsentieren einen stärker integrativen Ansatz, bei dem Zero-Trust-Prinzipien mit etablierten industriellen Architekturmodellen kombiniert werden. Konkret wird Zero Trust mit dem Purdue-Modell (Purdue Enterprise Reference Architecture, PERA) verbunden, welches weiterhin weithin als Grundlage für industrielle Kontrollsysteme (ICS) dient. In diesem Ansatz bleibt das Prinzip der «Defense-in-Depth» durch klare Zonierung und den Einsatz von Demilitarisierten Zonen (DMZ) erhalten. Gleichzeitig wird die Architektur durch Zero-Trust-Richtlinien und kontinuierliche Überwachung erweitert, um die zunehmende Konvergenz von IT- und OT-Systemen abzusichern [18].

Die genannten Arbeiten zeigen, dass Zero Trust in industriellen Umgebungen nicht allein als Ersatz einzelner Netzwerkkomponenten verstanden werden kann. Entscheidend ist vielmehr die Kombination aus expliziten Kommunikationsbeziehungen, kontinuierlicher Autorisierung und einer nachvollziehbaren Einbettung in bestehende IT-/OT-Strukturen. Für diese Arbeit ist insbesondere relevant, dass klassische Segmentierung weiterhin implizite Vertrauensbereiche erzeugen kann. Die Laborumgebung untersucht deshalb nicht nur Netzwerkzonen, sondern konkrete Dienste, Identitäten und Rollen. Damit knüpft die Arbeit an die Forschung zu feingranularer Segmentierung und IT-/OT-Konvergenz an, bewertet die Ansätze jedoch anhand einer praktisch implementierten Vergleichsumgebung mit Zero-Trust- und perimeterbasierter Architektur.

1.6. Grundlagen von Zero Trust

Zero Trust ist ein Sicherheitsmodell, das auf einem grundlegenden Paradigmenwechsel basiert: Kein Akteur, kein Gerät und keine Netzwerkverbindung wird automatisch als vertrauenswürdig eingestuft, unabhängig davon, ob sich diese innerhalb oder ausserhalb des Unternehmensnetzwerks befinden. Das National Institute of Standards and Technology (NIST) definiert Zero Trust als einen Ansatz, bei dem der Schutz von Ressourcen ins Zentrum rückt und nicht mehr der Schutz von Netzwerksegmenten, wobei sämtliche Datenquellen und Rechendienste als Ressourcen betrachtet werden, denen kein implizites Vertrauen entgegengebracht wird [13]. Die zentrale Annahme lautet, dass Vertrauen niemals implizit gewährt, sondern stets explizit und kontinuierlich überprüft werden muss. Dieses Prinzip wird in der Praxis- und Herstellerliteratur häufig mit der Formel *Never Trust, Always Verify* zusammengefasst. Der Abschnitt adressiert Teilfrage 1.1 und leistet mit dem NIST-Modell zugleich einen Beitrag zu Teilfrage 2.1.

1.6.1. Zero Trust Prinzipien

Dem Zero-Trust-Ansatz liegen mehrere zentrale Prinzipien zugrunde, die gemeinsam das Sicherheitsmodell formen. Im Zentrum steht die identitätsbasierte Zugriffskontrolle: Jeder Zugriff auf eine Ressource setzt eine eindeutige Authentifizierung und Autorisierung voraus. Entscheidend

ist dabei nicht die Netzwerkposition eines Akteurs, sondern dessen verifizierte Identität sowie der Kontext der Anfrage [13]. Eng damit verbunden ist das Prinzip des Least Privilege. Benutzer und Systeme erhalten ausschliesslich die für ihre jeweilige Aufgabe notwendigen Zugriffsrechte. Dadurch wird die potenzielle Angriffsfläche reduziert und der Schaden im Falle einer Kompromittierung begrenzt.

Ein weiteres zentrales Element ist die Mikrosegmentierung. Anstatt das Netzwerk als eine einheitlich vertrauenswürdige Zone zu betrachten, wird es in kleinere, voneinander isolierte Segmente unterteilt. Der Zugriff zwischen diesen Segmenten wird individuell kontrolliert und überwacht. Dadurch kann sich ein Angreifer nach einem erfolgreichen Eindringen nicht ungehindert lateral durch das Netzwerk bewegen [13]. Die Einschränkung dieser Laterale Bewegungen stellt einen wesentlichen Vorteil gegenüber klassischen Architekturen dar, in denen ein überbrückter Perimeter häufig weitreichenden Zugriff ermöglicht.

Darüber hinaus spielt die kontinuierliche Verifikation eine zentrale Rolle. In einem Zero-Trust-Modell ist Authentifizierung kein einmaliger Vorgang, sondern ein fortlaufender Prozess. Zugriffsentscheidungen werden dynamisch und kontextabhängig getroffen, beispielsweise basierend auf Gerätezustand, Standort oder Nutzerverhalten. Verändern sich diese Parameter, kann ein zuvor gewährter Zugriff angepasst oder entzogen werden [13]. Dadurch wird sichergestellt, dass Vertrauen nicht statisch bestehen bleibt, sondern fortlaufend an die aktuelle Situation angepasst wird.

1.6.2. Architektur

NIST definiert ein abstraktes Architekturmodell, um die Prinzipien in der Praxis umzusetzen. Dieses Modell beschreibt die logischen Komponenten, die für die Umsetzung einer Zero-Trust-Strategie erforderlich sind, und legt fest, wie diese zusammenwirken. Im Kern trennt die Architektur zwei zentrale Aufgaben voneinander: die Entscheidung über einen Zugriff und deren Durchsetzung [13].

Die Entscheidung über Zugriffsanfragen wird vom sogenannten Policy Decision Point (PDP) getroffen. Dieser besteht aus zwei logischen Teilkomponenten: der Policy Engine (PE) und dem Policy Administrator (PA). Die Policy Engine ist für die eigentliche Bewertung einer Zugriffsanfrage zuständig. Sie prüft, ob ein angefragter Zugriff auf Grundlage der definierten Richtlinien und der verfügbaren Kontextinformationen gewährt oder abgelehnt werden soll. Der Policy Administrator setzt die Entscheidung der Policy Engine um, indem er die entsprechenden Anweisungen an die durchsetzende Komponente weitergibt. Er fungiert somit als Bindeglied zwischen Entscheidung und Durchsetzung [13].

Die tatsächliche Durchsetzung einer Zugriffsentscheidung erfolgt durch den Policy Enforcement Point (PEP). Dieser kontrolliert den Zugang zu den geschützten Ressourcen und stellt sicher, dass nur autorisierte Zugriffe stattfinden. Der PEP gewährt oder blockiert eine Verbindung ausschliesslich auf Basis der vom PDP erhaltenen Anweisungen. Ohne eine explizite Freigabe durch den PDP wird kein Zugriff zugelassen [13]. Durch diese Trennung von Entscheidung und Durchsetzung lassen sich Sicherheitsrichtlinien zentral verwalten und gleichzeitig an verschiedenen Stellen innerhalb der Infrastruktur durchsetzen.

Damit der PDP fundierte Zugriffsentscheidungen treffen kann, benötigt er eine breite Grundlage an Kontextinformationen. Dazu gehört zunächst die Identität des anfragenden Akteurs, die eindeutig verifiziert sein muss. Darüber hinaus fließt der Zustand des verwendeten Geräts in die Bewertung ein, etwa ob dieses den aktuellen Sicherheitsanforderungen entspricht. Auch der Standort sowie weitere kontextuelle Faktoren wie Zugriffszeit oder Verhaltensmuster können berücksichtigt

werden [13]. Diese Informationen werden aus verschiedenen Datenquellen innerhalb der Unternehmensinfrastruktur bezogen und der Policy Engine zur Verfügung gestellt. Je umfassender und aktueller die verfügbaren Kontextdaten sind, desto präziser kann die Zugriffsentscheidung ausfallen.

Der Ablauf einer typischen Zugriffsanfrage lässt sich vereinfacht wie folgt beschreiben, wie auch in der Abbildung 1.1 dargestellt: Ein Akteur, beispielsweise ein Benutzer oder ein System, stellt über den PEP eine Anfrage auf eine geschützte Ressource. Der PEP leitet diese Anfrage an den PDP weiter, ohne selbst eine Entscheidung zu treffen. Innerhalb des PDP bewertet die Policy Engine die Anfrage anhand der geltenden Richtlinien und der verfügbaren Kontextinformationen. Auf Basis dieser Bewertung trifft sie eine Entscheidung, die der Policy Administrator an den PEP übermittelt. Dieser gewährt oder verweigert den Zugriff entsprechend [13]. Entscheidend ist, dass dieser Vorgang nicht einmalig stattfindet. Wie im Abschnitt zur kontinuierlichen Verifikation beschrieben, wird die Zugriffsentscheidung fortlaufend überprüft und bei veränderten Bedingungen angepasst.

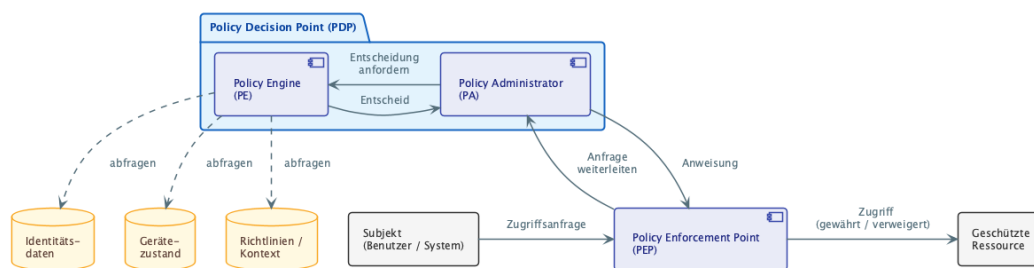


Abbildung 1.1.: Ablauf einer Zugriffsanfrage im NIST Architekturmodell

Das von NIST beschriebene Architekturmodell ist bewusst abstrakt gehalten und macht keine Vorgaben zur konkreten technischen Umsetzung. Es definiert vielmehr die logischen Rollen und deren Zusammenspiel, sodass Organisationen die Architektur an ihre spezifischen Anforderungen und bestehende Infrastruktur anpassen können [13].

1.7. Perimeterbasierte Sicherheitsansätze

Als Ausgangspunkt für den späteren Vergleich mit den in Abschnitt 1.6 beschriebenen Prinzipien von Zero Trust ist es sinnvoll, zunächst klassische Perimeterbasierte Sicherheitsarchitekturen einzuordnen. Diese Ansätze beruhen auf der Grundannahme, dass sich ein organisatorisch kontrolliertes internes Netzwerk von einem externen, nicht vertrauenswürdigen Netzwerk abgrenzen lässt. Sicherheit wird dabei in erster Linie über die Absicherung dieser Netzgrenze umgesetzt [19, 12]. Der Abschnitt adressiert Teilfrage 1.2, indem er die zentralen Unterschiede zum Zero-Trust-Modell vorbereitet.

Im Zentrum perimeterbasierter Sicherheitsarchitekturen steht somit der Netzwerkperimeter, also die Grenze zwischen dem internen Unternehmensnetz und externen Netzen wie dem Internet. An diesen Übergängen werden Sicherheitskomponenten wie Firewalls, Proxys oder VPN-Gateways eingesetzt, um den Datenverkehr zu kontrollieren und auf Grundlage definierter Regeln zuzulassen, einzuschränken oder zu blockieren. NIST beschreibt Firewalls in diesem Zusammenhang als Mechanismen zur Kontrolle des Netzwerkverkehrs zwischen Netzen oder Hosts mit unterschiedlicher Sicherheitslage [19].

Ein wesentliches Merkmal dieses Modells ist die starke Kopplung von Vertrauen an den Netzwerkstandort. Systeme und Benutzer innerhalb des internen Netzes gelten tendenziell als vertrauenswürdiger als externe Kommunikationspartner. Entsprechend wird der Schutzzschwerpunkt primär auf den ein- und ausgehenden Verkehr über die Netzgrenze gelegt. Innerhalb des Perimeters bestehen dagegen häufig grössere Kommunikationsfreiheiten, sofern diese nicht zusätzlich durch interne Segmentierung oder weitere Sicherheitszonen eingeschränkt werden [19, 12].

Zur praktischen Umsetzung perimeterbasierter Sicherheit werden Unternehmensnetze oft in unterschiedliche Zonen unterteilt. Häufig wird dabei eine DMZ eingesetzt, um öffentlich erreichbare Dienste logisch von internen Kernsystemen zu trennen. Ergänzend kommen Netzwerksegmentierung und zusätzliche interne Filtermechanismen zum Einsatz, um bestimmte Kommunikationsflüsse gezielt zu begrenzen. Auch wenn solche Massnahmen die Sicherheitsarchitektur verfeinern, bleibt der Perimeter als zentrales Schutzkonzept bestehen [19, 21].

Ein weiterer typischer Bestandteil perimeterbasierter Architekturen ist klassischer Remote Access über VPN-Technologien. Dabei authentisieren sich externe Benutzer zunächst gegenüber einem Zugangspunkt an der Netzgrenze und erhalten danach Zugriff auf Teile des internen Netzes. NIST führt aus, dass Remote-Access-Lösungen dazu dienen, externen Benutzern einen geschützten Zugang zu Unternehmensressourcen bereitzustellen. In perimeterbasierten Modellen wird damit der bestehende Sicherheitsansatz auf externe Zugriffe erweitert, ohne das Grundprinzip eines geschützten internen Netzes aufzugeben [20, 12].

Historisch waren perimeterbasierte Sicherheitsansätze besonders gut für IT-Umgebungen geeignet, in denen Anwendungen, Daten und Benutzer weitgehend innerhalb klar definierter Unternehmensnetze betrieben wurden. In solchen Umgebungen konnten Sicherheitskontrollen auf wenige zentrale Übergänge konzentriert werden, was die Verwaltung und Durchsetzung von Sicherheitsrichtlinien vereinfachte. Der Ansatz ist deshalb eng mit traditionellen Unternehmensnetzwerken und zentralisierten Rechenzentrumsstrukturen verbunden [19, 12].

Zusammenfassend lässt sich festhalten, dass perimeterbasierte Sicherheitsansätze Sicherheit primär als Schutz der Netzgrenze verstehen. Das zugrunde liegende Modell unterscheidet zwischen einem stärker geschützten internen Bereich und einem weniger vertrauenswürdigen externen Bereich. Die wichtigsten technischen Mittel sind dabei insbesondere Firewalls, Sicherheitszonen, DMZ-Konzepte, Segmentierung sowie kontrollierter Fernzugriff über zentrale Übergänge [19, 20, 12].

2. Konzeption

Dieses Kapitel beschreibt die fachliche und technische Konzeption der Laborumgebung, die eingesetzten Technologien sowie die beiden zu vergleichenden Sicherheitsarchitekturen.

2.1. Zielsetzung der Laborumgebung

Die Laborumgebung bildet die praktische Grundlage für den Vergleich zwischen der Zero-Trust-basierten Architektur und der perimeterbasierten Architektur. Sie stellt eine kontrollierte und reproduzierbare Umgebung bereit, in der beide Ansätze anhand derselben fachlichen Ressourcen, Benutzerrollen und Zugriffsszenarien untersucht werden können. Dadurch wird der Vergleich nicht auf abstrakte Architekturprinzipien beschränkt, sondern an einer konkreten gemischten IT/OT-Umgebung durchgeführt.

Die Umgebung ist so aufgebaut, dass sowohl klassische IT-Dienste als auch OT-nahe Komponenten berücksichtigt werden. Dies ist für die Forschungsfrage zentral, da Zero Trust nicht nur in einer reinen Office- oder Cloud-Umgebung bewertet werden soll, sondern in einem heterogenen Umfeld mit Benutzern, Anwendungen, Monitoring, Sensorik und steuernden Komponenten. Die Laborumgebung dient damit als gemeinsamer Referenzpunkt für die spätere Evaluation der Hypothesen.

2.1.1. Anforderungen aus der Forschungsfrage

Aus der Forschungsfrage ergibt sich die Anforderung, zwei unterschiedliche Sicherheitsarchitekturen unter möglichst vergleichbaren Bedingungen zu untersuchen. Die Laborumgebung muss daher Ressourcen enthalten, die in beiden Architekturen betrachtet werden können. Dazu gehören Benutzer mit unterschiedlichen Aufgaben, IT-Dienste mit unterschiedlichen Schutzbedarfen sowie OT-Komponenten, die sowohl lesende als auch steuernde Kommunikationsbeziehungen abbilden.

Die Dienste sind über drei Hosts verteilt. Diese Verteilung ist nicht zwingend erforderlich, um die einzelnen Dienste technisch betreiben zu können. Sie unterstützt jedoch die Zielsetzung des Labors, unterschiedliche Standorte und Betriebszonen abzubilden. Gerade im Zero-Trust-Kontext soll der Zugriff nicht primär vom Netzwerkstandort eines Geräts abhängen, sondern von Identität, Richtlinien und explizit freigegebenen Diensten. Die verteilte Architektur macht daher sichtbar, dass die Laborumgebung auch über mehrere Standorte hinweg betrieben und abgesichert werden kann.

Damit adressiert dieser Abschnitt Teilfrage 3.1. Aus der Kombination von IT- und OT-Komponenten ergeben sich insbesondere Anforderungen an die Trennung unterschiedlicher Schutzbedarfe, die Begrenzung lateraler Bewegungen zwischen IT- und OT-Diensten, die Unterscheidung zwischen lesenden Sensorzugriffen und steuernden OT-Zugriffen sowie die kontrollierte Einbindung externer

Wartungszugänge. Zusätzlich müssen Zugriffe nachvollziehbar einer Rolle, Identität oder Maschine zugeordnet werden können, da reine Netzzugehörigkeit in einer heterogenen Umgebung nicht ausreicht, um legitime und missbräuchliche Kommunikationsbeziehungen zuverlässig zu unterscheiden.

Die zentralen Plattformdienste wie OpenZiti Controller, Keycloak und Wazuh werden auf einem Cloud-Host betrieben. Dadurch sind sie unabhängig vom lokalen Labornetz erreichbar und können als gemeinsame Kontroll- und Managementebene für die übrigen Komponenten dienen. Zusätzlich war zu Beginn der Umsetzung noch nicht abschliessend bestimmbar, wie viele Ressourcen diese Dienste benötigen würden. Ein Cloud-Angebot bot deshalb die Möglichkeit, die Ressourcen bei Bedarf nachträglich einfacher zu skalieren.

Der erste virtuelle Server, VPS1, läuft beim Schweizer Cloud-Anbieter Infomaniak. Der zugrunde liegende Server befindet sich physisch in Plan-les-Ouates im Kanton Genf. Der zweite virtuelle Server, VPS2, läuft beim US-amerikanischen Cloud-Anbieter Linode; der verwendete Standort befindet sich physisch in Mailand, Italien. Beide Anbieter wurden gewählt, da bereits praktische Kenntnisse mit den jeweiligen Plattformen vorhanden waren und beide die für das Laborszenario benötigten Funktionen bereitstellen.

Beim On-Premises-Server handelt es sich um ein ausrangiertes ThinkPad T410 aus dem Jahr 2011. Die Wahl fiel primär auf dieses Gerät, da dadurch keine zusätzliche Hardware beschafft werden musste. Gleichzeitig zeigt der Einsatz, dass ältere Hardware in einem anderen Kontext, in diesem Fall als lokaler Server, weiterhin sinnvoll verwendet werden kann.

Für die finale Laborumgebung sind folgende Systeme vorgesehen:

- **VPS1** als Teil der serverseitigen Laborinfrastruktur,
- **VPS2** als weiterer serverseitiger Standort für ausgelagerte Dienste,
- **ein On-Premises-Server** zur Abbildung eines lokalen Unternehmens- oder Produktionsstandorts,
- **zwei Raspberry Pis** als OT-nahe Komponenten,
- **ein Client-System für Alice**, die eine Buchhaltungs-Rolle repräsentiert,
- **ein Client-System für Bob**, der einen technischen Mitarbeitenden des Unternehmens repräsentiert.

Die beiden Raspberry Pis bilden den OT-Anteil der Umgebung ab und berücksichtigen zwei unterschiedliche OT-Perspektiven: die Erfassung von Zustandsdaten über einen Sensor sowie die Steuerung einer technischen Komponente in Form eines simulierten Lüfters. Diese Unterscheidung ist für die spätere Bewertung relevant, weil reine Lesezugriffe und steuernde Zugriffe unterschiedliche sicherheitstechnische Anforderungen haben. Die technische Umsetzung der OT-Komponenten wird in Abschnitt 3.1 beschrieben.

Die beiden Client-Systeme dienen dazu, unterschiedliche Benutzerprofile in der Umgebung abzubilden. Diese Rollenverteilung unterstützt die Untersuchung von Least Privilege, da nicht alle Benutzer dieselben Ressourcen benötigen oder erhalten sollen. Die konkreten Rollen, Identitäten und Zugriffsbeziehungen von Alice und Bob werden in Abschnitt 2.3 im Detail beschrieben.

Die Laborumgebung wird physisch aufgebaut, um die Interaktion zwischen den Komponenten realitätsnah zu gestalten. Die Serverkomponenten laufen auf VPS-Instanzen, die über das Internet erreichbar sind, während die Clients und OT-Komponenten lokal im CN-Lab der Hochschule betrieben werden. Der physische Laboraufbau mit den Client-Arbeitsplätzen von Alice und Bob sowie der Lüfter-Simulation ist in Abbildung 2.1 dargestellt. Dadurch können sowohl interne als auch

externe Zugriffsszenarien untersucht werden, was für die Bewertung der Sicherheitsarchitekturen von zentraler Bedeutung ist.



Abbildung 2.1.: Physischer Laboraufbau mit den Client-Arbeitsplätzen von Alice und Bob sowie der Lüfter-Simulation

2.1.2. Abgrenzung des Laborszenarios

Die Laborumgebung ist keine vollständige Nachbildung eines produktiven Unternehmens- oder Produktionsnetzwerks. Sie bildet ausgewählte Eigenschaften einer gemischten IT/OT-Umgebung ab, die für den Vergleich der beiden Sicherheitsarchitekturen relevant sind. Dazu gehören ausgelagerte Dienste, unterschiedliche Benutzerrollen, interne und externe Zugriffspfade sowie OT-Komponenten mit Sensor- und Steuerfunktion.

Bewusst nicht im Vordergrund stehen Aspekte wie Hochverfügbarkeit, Skalierung auf eine grosse Anzahl von Benutzern oder Geräten, produktive Betriebsprozesse und vollständige industrielle Prozesssteuerung. Der Fokus liegt stattdessen auf einer nachvollziehbaren Laborumgebung, in der Zugriffsbeziehungen, Angriffsfläche, Überwachung und Betriebsaufwand vergleichbar untersucht werden können.

Der Infrastructure-as-Code-Ansatz unterstützt diese Zielsetzung, indem zentrale Dienste, Policies und Konfigurationen versioniert und reproduzierbar beschrieben werden. Die Grenzen dieses Ansatzes, etwa bei physischen Komponenten wie Raspberry Pis, Sensorik, Lüfter-Hardware und Client-Systemen, werden in Abschnitt 3.2.6 diskutiert.

Die finale Dienstverteilung trennt zentrale Plattformdienste (**VPS1**), ausgelagerte Dienste (**VPS2**) und lokale IT-/OT-nahe Dienste (**On-Premises-Server** und Raspberry Pis). Die fachlichen IT- und OT-Dienste dieser Umgebung werden in Abschnitt 2.3 beschrieben, die Zuordnung der Plattformkomponenten zur Zero-Trust-Architektur in Abschnitt 2.4. Die perimeterbasierte Vergleichsarchitektur ist als WireGuard-/VPN-Variante mit einer OPNsense-Firewall umgesetzt und wird in Abschnitt 2.5 dargestellt. Der Abschnitt leistet damit zugleich einen Beitrag zu Teilfrage 3.2, da er die fachliche und technische Grundlage für die spätere Modellierung der Zero-Trust-Architektur festlegt.

2.1.3. Vergleichbarkeit der Architekturen

Die Vergleichbarkeit der beiden Architekturen ergibt sich daraus, dass sie dieselbe fachliche Laborumgebung absichern sollen. Die relevanten Benutzer, IT-Dienste und OT-Komponenten bleiben als Vergleichsgegenstand gleich. Unterschiede sollen sich aus der jeweiligen Sicherheitsarchitektur ergeben, nicht aus unterschiedlichen Anwendungen oder Rollenmodellen.

In der Zero-Trust-basierten Architektur werden Zugriffe dienst- und identitätsbezogen modelliert. In der perimeterbasierten Architektur steht demgegenüber ein klassischer Netzwerkzugang im Vordergrund. Beide Ansätze müssen jedoch dieselben grundlegenden Aufgaben unterstützen: Alice und Bob sollen ihre vorgesehenen Tätigkeiten ausführen können, während nicht autorisierte oder missbräuchliche Zugriffe erkennbar und begrenzt sein sollen.

Für die Evaluation bedeutet dies, dass dieselben Szenarien aus Abschnitt 1.4 auf beide Architekturen angewendet werden. Die Laborumgebung muss daher so gestaltet sein, dass Angriffs- und Wartungsszenarien in beiden Varianten nachvollziehbar durchgeführt werden können. Nur so lassen sich Unterschiede hinsichtlich Angriffsfläche, Erkennung, Eindämmung, Auswirkungen auf legitime Zugriffe und Betriebsaufwand systematisch bewerten.

2.2. Eingesetzte Technologien

Die Umsetzung der Zero-Trust-Architektur basiert auf einem Stack von Open-Source-Technologien, welche die zentralen Rollen des NIST-Zero-Trust-Modells abbilden. Keycloak dient als Identity Provider, OpenZiti realisiert den identitätsbasierten Netzwerkzugriff, Envoy übernimmt die Rolle des Policy Enforcement Point, Open Policy Agent steuert die Richtlinienentscheidungen, und Wazuh ergänzt die Architektur um kontinuierliche Verifikation und Sicherheitsüberwachung. Dabei ist wichtig zu betonen, dass davon ausgegangen wird, dass die eingesetzten Komponenten keine Sicherheitslücken aufweisen. Daher wird in dieser Arbeit nicht die Sicherheit der einzelnen Technologien bewertet, sondern die Auswirkungen ihrer Integration in die Gesamtarchitektur und die daraus resultierenden Sicherheitsmerkmale im Vergleich zur perimeterbasierten Architektur untersucht. Der Abschnitt adressiert Teilfrage 2.2, indem er die für die Umsetzung verwendeten Werkzeuge und deren Rolle im Architekturmodell begründet.

2.2.1. Begründung für den Einsatz von Open-Source-Software

In dieser Arbeit wird die Zero-Trust-Architektur ausschliesslich mit Open-Source-Software umgesetzt. Diese Entscheidung dient der methodischen Nachvollziehbarkeit der Umsetzung. Da Zero Trust auf überprüfbaren Identitäten, expliziten Richtlinienentscheidungen, kontrollierten Zugriffspfaden und kontinuierlicher Überwachung basiert, müssen die eingesetzten Komponenten transparent analysierbar und reproduzierbar betreibbar sein. Open-Source-Software ermöglicht die Prüfung von Schnittstellen, Konfigurationen und Sicherheitsmechanismen und reduziert zugleich die Abhängigkeit von einzelnen Herstellern.

Der Einsatz offener Technologien bedeutet nicht, dass die verwendeten Komponenten per se sicher sind. Er schafft jedoch die Voraussetzung, sicherheitsrelevante Annahmen, Integrationspunkte und Konfigurationsentscheidungen offen zu bewerten. Damit unterstützt Open Source den wissenschaftlichen Anspruch dieser Arbeit, eine nachvollziehbare, erweiterbare und herstellerunabhängige Architekturgrundlage zu schaffen.

2.2.2. Abgrenzung zu proprietären Enterprise-Komponenten

Eine zusätzliche Schwierigkeit bei der Umsetzung bestand darin, dass Netzwerk- und Sicherheitskomponenten für Enterprise-Umgebungen häufig mit hohen Anschaffungs- und Betriebskosten verbunden sind. Dies betrifft insbesondere spezialisierte Appliances wie Firewalls, Proxies, Load Balancer oder WAN-Optimierer, die in klassischen Unternehmensnetzen als dedizierte Hardwarekomponenten betrieben werden. In der Literatur werden solche Middleboxes als kostenintensive und häufig proprietäre Systeme beschrieben, die sowohl die Investitionskosten als auch den Betriebsaufwand eines Netzwerks erheblich beeinflussen können [1].

Auch Arbeiten zu Network Functions Virtualisation (NFV) begründen die Virtualisierung von Netzwerkfunktionen unter anderem damit, proprietäre Spezialhardware durch softwarebasierte Funktionen auf standardisierter Infrastruktur zu ersetzen und dadurch Kapital- und Betriebskosten zu reduzieren [4, 26]. Diese Kostenproblematik ist für eine Laborumgebung besonders relevant, da der Erwerb kommerzieller Enterprise-Komponenten den finanziellen Rahmen einer wissenschaftlichen Arbeit deutlich überschreiten kann. Der Einsatz von Open-Source-Software ermöglicht es daher, zentrale Zero-Trust-Funktionen realitätsnah abzubilden, ohne die Architektur an kostenintensive proprietäre Produkte zu binden.

2.2.3. Auswahlkriterien

Die Wahl der Werkzeuge orientiert sich an vier Kriterien:

- **Transparenz:** Schnittstellen und Sicherheitsmechanismen sind vollständig dokumentiert und nachvollziehbar.
- **Reproduzierbarkeit:** Die Komponenten können in vergleichbaren Umgebungen zuverlässig erneut bereitgestellt werden.
- **Modularität:** Identität, Richtlinienentscheidung, Richtliniendurchsetzung und Überwachung bleiben klar getrennt.
- **Erweiterbarkeit:** Neue Dienste und Rollen können ohne grundlegenden Architekturwechsel ergänzt werden.

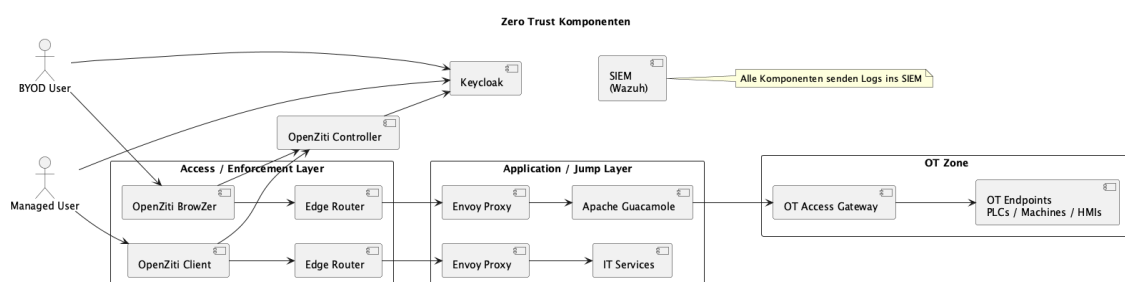


Abbildung 2.2.: Komponentenübersicht des eingesetzten Technologie-Stacks

2.2.4. Zuordnung der Technologien zu den Zero-Trust-Rollen

Die in Abschnitt 1.6 beschriebenen logischen Rollen werden mit den in Tabelle 2.1 aufgeführten Komponenten umgesetzt:

Zero-Trust-Rolle	Technologie
Identity Provider	Keycloak
Identitätsbasierter Netzwerkzugriff	OpenZiti
Policy Enforcement Point	Envoy
Policy Engine	Open Policy Agent
Kontinuierliche Verifikation / SIEM	Wazuh

Tabelle 2.1.: Zuordnung der Zero-Trust-Rollen zu den eingesetzten Technologien

2.2.5. Keycloak

Keycloak stellt das zentrale Identitäts- und Zugriffsmanagement bereit und unterstützt OpenID Connect, OAuth 2.0 sowie SAML [8]. Für Zero Trust ist diese Funktion grundlegend, weil Zugriffe an überprüfbare Identitäten gebunden werden und nicht an Netzwerkstandorte.



Keycloak-Logo

Im Gesamtmodell übernimmt Keycloak die Grundlage für konsistente Authentifizierung und Rollenverwaltung über alle Dienste hinweg. Detailinformationen zur konkreten Integration in den Zugriffspfad folgen in Abschnitt 2.4.

Die verfügbaren Anbindungen von Keycloak an verschiedene Anwendungen und Protokolle erleichtern die Integration in die Laborumgebung, da sowohl Webdienste als auch andere Anwendungen über Keycloak-gestützte Authentifizierung abgesichert werden können. Insbesondere auch die Möglichkeit, Keycloak an OpenZiti zu koppeln, führte zu seiner Auswahl als Identity Provider für die Zero-Trust-Architektur in dieser Arbeit. Alternativ hätten auch andere Identity Provider mit ähnlicher Funktionalität eingesetzt werden können, wie zum Beispiel Authentik oder Authelia. Keycloak wurde jedoch aufgrund bestehender Erfahrungen und Kenntnisse ausgewählt.

2.2.6. OpenZiti

OpenZiti realisiert den identitätsbasierten Netzwerkzugriff [15]. Dienste werden nicht allgemein im internen Netz erreichbar gemacht, sondern ausschliesslich über explizit definierte Identitäten und Service-Beziehungen bereitgestellt. Nähere Details zur konkreten Funktionsweise von OpenZiti und seiner Integration in die Architektur werden in Abschnitt 2.4 beschrieben.



OpenZiti-Logo

Damit unterstützt OpenZiti insbesondere Mikrosegmentierung und die Begrenzung lateraler Bewegungen. Der Fokus liegt auf berechtigten Kommunikationsbeziehungen zwischen konkreten Akteuren und Diensten.

Alternativ zu OpenZiti hätten auch andere Software-Defined-Perimeter-Lösungen oder Identity-Aware-Proxies wie Tailscale eingesetzt werden können. Diese bieten jedoch keine vergleichbar feingranulare, dienstbezogene Modellierung über Services, Konfigurationen und Dial-Policies, sodass zusätzliche Anpassungen erforderlich gewesen wären [15]. OpenZiti war deshalb die bevorzugte Wahl für diese Arbeit.

2.2.7. Envoy

Envoy wird als vorgeschalteter Proxy eingesetzt und übernimmt die technische Durchsetzung von Zugriffsentscheidungen im Kommunikationspfad [3]. Im NIST-Modell entspricht dies funktional dem Policy Enforcement Point (PEP).



Envoy-Logo

Die Komponente ermöglicht eine einheitliche Durchsetzung für mehrere Dienste, ohne Autorisierungslogik in jeder Anwendung separat zu implementieren.

Die Entscheidung für Envoy basiert auf der offiziell unterstützten Anbindung an Open Policy Agent über die `ext_authz`-Schnittstelle, mit der Richtlinienentscheidungen zentral gesteuert werden können [3, 14]. Alternativ hätten auch andere Proxy-Lösungen wie Nginx oder Traefik eingesetzt werden können; für diese wäre die Anbindung einer externen Policy-Engine jedoch mit zusätzlichem Integrationsaufwand verbunden gewesen.

2.2.8. Open Policy Agent

Open Policy Agent (OPA) übernimmt die zentrale richtlinienbasierte Entscheidung über Zugriffsanfragen [14]. Richtlinien werden deklarativ in der Sprache Rego verwaltet und sind von der technischen Durchsetzung getrennt.

Diese Trennung unterstützt konsistente Entscheidungen über mehrere Dienste hinweg und erleichtert die Wiederverwendung von Regeln im gesamten Labor. Alternativ zu Open Policy Agent hätten auch andere Policy-Engines oder selbst entwickelte Richtlinienmodule eingesetzt werden können. OPA bietet jedoch eine etablierte, gut dokumentierte Lösung, die sich über die `ext_authz`-Schnittstelle direkt in Envoy integrieren lässt [14], weshalb es die bevorzugte Wahl für diese Arbeit war.



OPA-Logo

2.2.9. Wazuh

Wazuh ergänzt die Architektur um Sicherheitsüberwachung und liefert Kontextdaten aus Endpunkten und Systemereignissen [24]. Damit stehen aktuelle Zustandsinformationen für kontinuierliche Verifikation und für die Erkennung von Abweichungen zur Verfügung.



Wazuh-Logo

Im Zero-Trust-Kontext stärkt Wazuh insbesondere den laufenden Bewertungsprozess, der über eine reine Erstauthentifizierung hinausgeht. Da Wazuh eine Log-Management- und SIEM-Funktionalität (Security Information and Event Management) bietet, ermöglicht es die zentrale Sammlung, Analyse und Korrelation von Sicherheitsereignissen in der Laborumgebung, was für die kontinuierliche Bewertung von grosser Bedeutung ist. Alternativ hätten auch separate Log-Management- oder SIEM-Lösungen eingesetzt werden können, jedoch bietet Wazuh eine integrierte Lösung, die sowohl die Sammlung von Sicherheitsereignissen als auch die Analyse und Korrelation in einem einzigen System ermöglicht.

2.2.10. Einordnung

Die Kombination der ausgewählten Technologien bildet die zentralen Rollen des NIST-Modells mit klar getrennten Verantwortlichkeiten ab [13]. Dadurch entsteht eine nachvollziehbare Grundlage für die späteren Architekturvergleiche und die Evaluation in den folgenden Kapiteln.

2.3. Gemeinsame IT/OT-Umgebung

Aus der in Abschnitt 2.1 begründeten Vergleichbarkeit der Architekturen ergibt sich die Anforderung einer gemeinsamen fachlichen Laborumgebung. Diese besteht aus klassischen IT-Diensten, OT-nahen Diensten, menschlichen Benutzern, Maschinenidentitäten und Service-Identitäten. Die konkreten Zugriffsbeziehungen sind in einer zentralen Policy-Matrix modelliert. Details zur Policy-Matrix und deren Implementation finden sich im Abschnitt 3.2. Der Abschnitt leistet einen Beitrag zu den Teilfragen 3.1 und 3.2, indem er die sicherheitsrelevanten Rollen, Dienste und Kommunikationsbeziehungen der heterogenen Umgebung beschreibt.

Die gemeinsame Umgebung ist bewusst heterogen aufgebaut. Sie enthält Webanwendungen, Monitoring, industrielle Kommunikationsprotokolle und physische beziehungsweise hardwarenahe Komponenten. Damit bildet sie nicht ein einzelnes produktives Unternehmensnetz nach, sondern stellt eine kontrollierte Laborumgebung bereit, in der typische IT/OT-Kommunikationsbeziehungen reproduzierbar untersucht werden können.

2.3.1. Benutzer, Rollen und Identitäten

Die Zugriffssteuerung der Laborumgebung basiert auf Rollen und Identitäten. Die Policy-Matrix unterscheidet zwischen menschlichen Identitäten, Maschinen-Clients und Service-Hosts. Menschliche Identitäten repräsentieren Benutzer mit fachlichen Aufgaben, Maschinen-Clients repräsentieren technische Zugriffe zwischen Komponenten, und Service-Hosts stellen private Dienste in der Architektur bereit.

Die wichtigsten menschlichen Identitäten sind Alice, Bob, ein Service-Techniker und ein separater administrativer Benutzer. Alice ist als Benutzerin für Buchhaltungs- und Intranet-Zugriffe vorgesehen. Bob ist als Maschinenbediener modelliert und besitzt Zugriff auf Intranet, Monitoring und das OT-Dashboard. Der Service-Techniker erhält keinen allgemeinen Zugriff auf alle Anwendungen, sondern spezifische Rollen für direkte OT-nahe Zugriffe, insbesondere auf den MQTT-Broker, das OT-Dashboard und den DHT11-Modbus-Dienst. Die Identität `ops-admin` ist dagegen ausschliesslich für administrative SSH-Zugriffe (z. B. für die Wartung von OpenZiti oder Keycloak) vorgesehen und besitzt dafür die Rolle `ssh-admins`. Diese Trennung zeigt, dass Berechtigungen nicht pauschal an Netzwerkzugehörigkeit, sondern an konkrete Aufgabenbereiche gebunden werden. Tabelle 2.2 fasst die zentralen Identitäten der Laborumgebung zusammen.

Identität	Typ	Rollen und Zweck
alice	Menschlich	Benutzerin mit Zugriff auf Intranet und Buchhaltung.
bob	Menschlich	Maschinenbediener mit Zugriff auf Intranet, Uptime Kuma und OT-Dashboard.
service-technician	Menschlich	Externer Service-Techniker mit direktem Zugriff auf ausgewählte OT-Dienste.
ops-admin	Menschlich	Administrativer Benutzer mit Zugriff auf die privaten SSH-Dienste von VPS1, VPS2 und On-Prem.
ot-dht11-client	Maschinen-Client	Technische Identität für den Zugriff auf den DHT11-Modbus-Dienst.
ot-fan-client	Maschinen-Client	Technische Identität für den Zugriff auf den MQTT-Broker zur Fan-Steuerung.
intranet-host	Service-Host	Bindet den Intranet-Dienst in das private OpenZiti-Netz ein.
onprem-services-host	Service-Host	Bindet die On-Prem-OT-Dienste wie OT-Dashboard, MQTT und Accounting ein.
uptime-kuma-host	Service-Host	Bindet den Uptime-Kuma-Dienst in das private OpenZiti-Netz ein.

Tabelle 2.2.: Auszug zentraler Identitäten der Laborumgebung

Die vollständige Zuordnung zwischen Identitäten, Rollen und Diensten ist in der Policy-Matrix hinterlegt, aus der auch die technischen OpenZiti- und OPA-Artefakte generiert werden (vgl. Abschnitt 3.2).

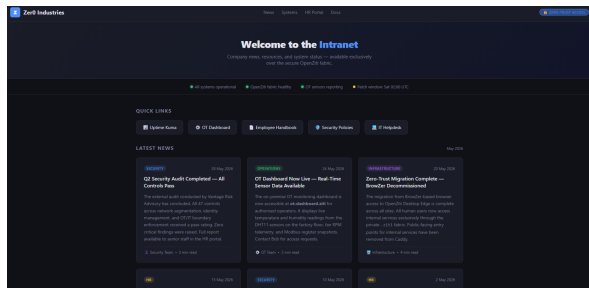
2.3.2. IT-Dienste

Die IT-Seite der Laborumgebung umfasst mehrere Web- und Monitoring-Dienste, die in Tabelle 2.3 aufgeführt und in Abbildung 2.3 visuell dargestellt sind. Das Intranet, zu sehen in Abbildung 2.3a, dient als interne Webanwendung, die einen typischen Unternehmensdienst repräsentiert. Die Buchhaltungs-Anwendung, zu sehen in Abbildung 2.3b, stellt eine Anwendung dar, welche das Bezahlen von Löhnen simuliert, die gemäss Policy-Matrix nur für Benutzer mit Accounting-Rolle vorgesehen ist. Uptime Kuma, zu sehen in Abbildung 2.3c, bildet einen Monitoring-Dienst ab, der nicht für alle Benutzer erreichbar sein soll, sondern gezielt für Benutzer mit entsprechender Monitoring-Berechtigung bereitgestellt wird.

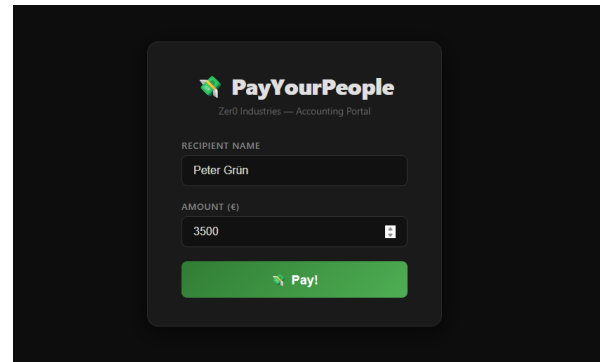
Diese Dienste werden nicht isoliert betrachtet, sondern als Ziele konkreter Zugriffsbeziehungen. Bei den HTTP-basierten Diensten wird zusätzlich zur privaten Netzwerkverbindung eine Anwendungsebene mit Authentifizierung und Autorisierung eingesetzt, die ein dienstübergreifendes Single Sign-on über Keycloak ermöglicht. Die zugehörige Zugriffskette aus OpenZiti, oauth2-proxy, Envoy und Open Policy Agent wird in Abschnitt 2.4 im Detail beschrieben.

Dienst	Kategorie	Zweck im Laborszenario
intranet	Anwendung	Interner Webdienst für reguläre Benutzerzugriffe.
accounting	Anwendung	Fachanwendung für Accounting-Benutzer.
uptime-kuma	Monitoring	Monitoring-Dienst, der über eine private OpenZiti-Adresse erreichbar ist.

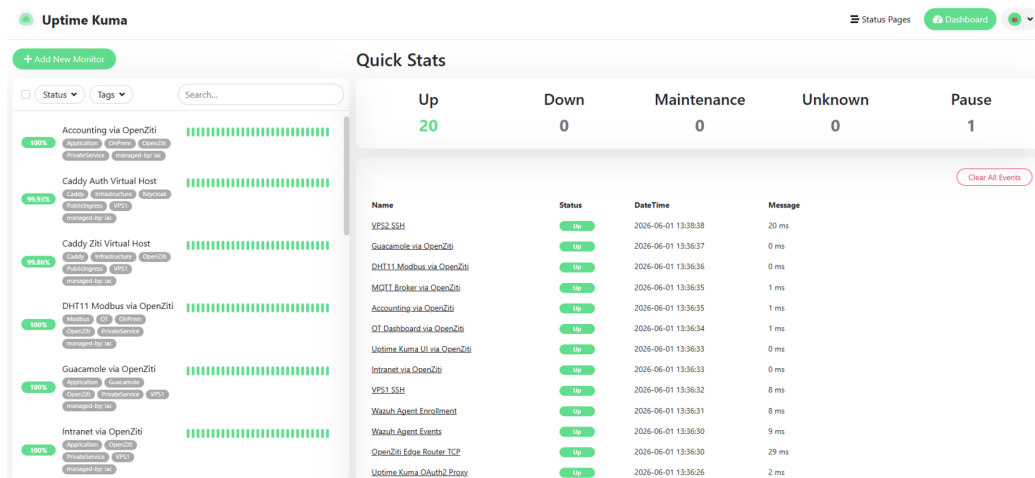
Tabelle 2.3.: IT-Dienste der gemeinsamen Laborumgebung



(a) Intranet



(b) Accounting-Anwendung



(c) Uptime Kuma Monitoring

Abbildung 2.3.: Benutzeroberflächen der IT-Dienste in der Laborumgebung

2.3.3. OT-Dienste

Die OT-Seite der Umgebung bildet typische industrielle Kommunikationsmuster in vereinfachter Form ab. Der DHT11-Dienst stellt Temperatur- und Luftfeuchtigkeitswerte über Modbus TCP bereit, der MQTT-Broker dient als Kommunikationsdrehscheibe für Zustandsmeldungen und Steuerbefehle, und das OT-Dashboard verbindet beide Seiten über eine Weboberfläche zur Anzeige und Steuerung, wie in Abbildung 2.4 zu sehen. Tabelle 2.4 listet die OT-Dienste mit den jeweils verwendeten Protokollen auf. Die Lüfter-Komponente empfängt Steuerbefehle über MQTT und veröffentlicht ihren Zustand zurück an den Broker. Damit werden sowohl lesende Sensorzugriffe als auch steuernde OT-Zugriffe abgebildet. Die technische Umsetzung dieser Komponenten, einschliesslich Hardware, Registermodell und MQTT-Topics, wird in Abschnitt 3.1 beschrieben.

Dienst	Protokoll	Funktion
ot-dht11-modbus	Modbus TCP	Bereitstellung von Temperatur- und Luftfeuchtheitswerten.
mqtt-broker	MQTT	Austausch von Sensorzuständen, Lüfter-Befehlen und Lüfter-Zuständen.
ot-dashboard	HTTP, Modbus, MQTT	Weboberfläche zur Anzeige von Sensordaten und zur Steuerung der Lüfter-Komponente.

Tabelle 2.4.: OT-Dienste der gemeinsamen Laborumgebung

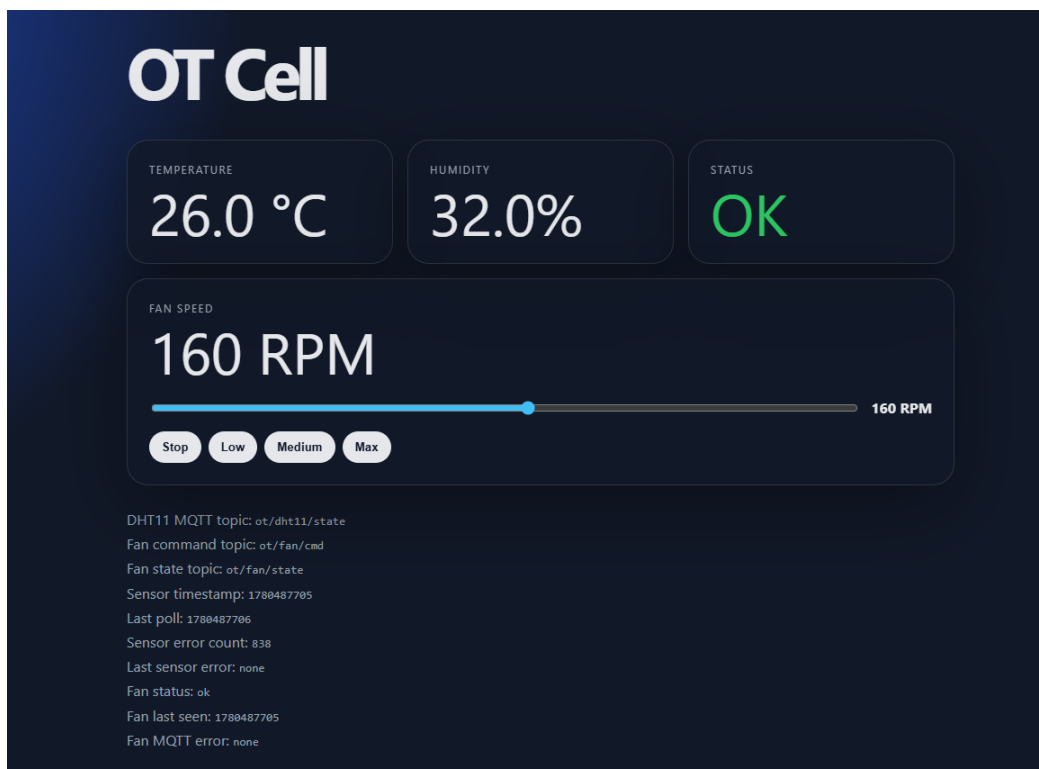


Abbildung 2.4.: OT-Dashboard zur Anzeige von Sensordaten und Lüfter-Zustand

2.3.4. Kommunikationsbeziehungen

Die regulären Kommunikationsbeziehungen sind in der Policy-Matrix dienstorientiert beschrieben. Ein Dienst besitzt jeweils eine Zieladresse, einen Intercept-Namen, bindende Service-Identitäten und erlaubte Dial-Rollen oder Dial-Identitäten. Dadurch lässt sich für jeden Dienst nachvollziehen, wer ihn anbieten darf und wer ihn konsumieren darf.

Für die Benutzerzugriffe bedeutet dies beispielsweise, dass Alice auf das Intranet und die Accounting Anwendung zugreifen kann, Bob auf Intranet, Uptime Kuma und das OT-Dashboard, und der Service-Techniker auf ausgewählte OT-Dienste. Für Maschinenkommunikation existieren eigene Identitäten, etwa für das OT-Dashboard als Modbus-Client oder für die Lüfter-Komponente als MQTT-Client. Die Kommunikation zwischen OT-Dashboard, Modbus-Dienst, MQTT-Broker und Lüfter-Komponente ist damit nicht nur technisch, sondern auch rollenbasiert modelliert.

Auch der administrative SSH-Zugriff wird als Kommunikationsbeziehung in der Policy-Matrix

modelliert: Dial-Berechtigung für die SSH-Dienste der drei Server erhält ausschliesslich die Rolle `ssh-admins`, die im Laborszenario nur der Identität `ops-admin` zugewiesen ist. Dadurch wird auch die Verwaltungsebene nicht als allgemein erreichbarer Netzwerkdienst betrachtet, sondern als explizit freigegebener privater Dienst. Die technische Umsetzung dieses Zugriffspfads über OpenZiti wird in Abschnitt 2.4 beschrieben.

In der perimeterbasierten Vergleichsarchitektur werden dieselben fachlichen Ressourcen dagegen nicht über eine dienstorientierte Policy-Matrix erschlossen, sondern über die Netzposition eines Clients, den VPN-Kontext und veröffentlichte Dienstpfade. Die daraus resultierenden Vertrauensannahmen und Zugriffswege werden in Abschnitt 2.5 beschrieben.

Die gemeinsame IT/OT-Umgebung bildet somit die fachliche Grundlage für den späteren Vergleich. Beide Netzwerkarchitekturen sollen dieselben Ressourcen und Aufgaben abbilden, unterscheiden sich jedoch darin, wie Zugriff gewährt, durchgesetzt, überwacht und eingeschränkt wird.

2.4. Zero-Trust-Architektur

Die implementierte Zero-Trust-Architektur setzt die in Abschnitt 1.6 beschriebenen Prinzipien nicht durch ein einzelnes Produkt um, sondern durch das Zusammenspiel mehrerer spezialisierter Komponenten. Der Zugriff auf Ressourcen erfolgt dienstbezogen und identitätsbasiert. Anwendungen und OT-Dienste werden nicht als allgemein erreichbare interne Systeme bereitgestellt, sondern über definierte Services, Identitäten und Policies zugänglich gemacht. Der Abschnitt adressiert die Teilfragen 3.2 und 4.1, indem er die modellierte Zero-Trust-Architektur und ihre technische Umsetzung im Labor beschreibt.

Die Architektur trennt dabei mehrere Aufgaben: Keycloak stellt menschliche Identitäten und OIDC-basierte Authentifizierung bereit, OpenZiti realisiert den privaten Netzwerkzugriff auf einzelne Dienste, oauth2-proxy vermittelt die Benutzeranmeldung für HTTP-Anwendungen, Envoy setzt Autorisierungsentscheidungen im Datenpfad durch, Open Policy Agent bewertet Zugriffsanfragen anhand der Policy-Matrix, und Wazuh ergänzt die Umgebung um sicherheitsbezogene Überwachung sowie eine Möglichkeit zur dynamischen Zugriffseinschränkung.

2.4.1. Architekturüberblick

Der zentrale Stack wird über Docker Compose bereitgestellt. Die Basisdienste umfassen PostgreSQL für den Keycloak-Zustand, Keycloak als Identity Provider, den OpenZiti Controller, einen OpenZiti Router, einen Bootstrap-Container zur Einrichtung von Identitäten und Policies sowie OPA als Policy Decision Point für anwendungsseitige Entscheidungen. Für den VPS-Betrieb ergänzt Caddy die Architektur als öffentlicher TLS-Reverse-Proxy für ausgewählte Verwaltungs- und Plattformendpunkte, insbesondere Keycloak, die OpenZiti Edge API und das Wazuh Dashboard.

Die privaten Anwendungsdienste folgen einem wiederkehrenden Muster. Ein Dienst wird intern bereitgestellt, über oauth2-proxy an Keycloak angebunden, durch Envoy vor die eigentliche Anwendung geschaltet und von OPA autorisiert. Anschliessend hostet eine OpenZiti-Service-Identität diesen Zugriffspfad in das private OpenZiti-Netz. Benutzer greifen nicht direkt auf den Container oder Host-Port zu, sondern über einen OpenZiti-Intercept wie `intranet.ziti`, `accounting.ziti`, `kuma.ziti` oder `ot-dashboard.ziti`. Nicht-HTTP-Dienste wie MQTT, Modbus, RDP und administrative SSH-Zugriffe werden ebenfalls als OpenZiti-Services modelliert, jedoch ohne oauth2-proxy- und Envoy-Schicht.

2.4.1.1. Labordomäne für öffentliche Endpunkte

Für die Laborumgebung wurde zur Vereinfachung die Domäne `prench-fress.xyz` registriert. Alle öffentlich erreichbaren Endpunkte der Zero-Trust-Umgebung verwenden Subdomains dieser Domäne, beispielsweise für Keycloak, die OpenZiti Edge API, das Wazuh Dashboard oder den Guacamole-Zugriff. Dadurch mussten TLS-Zertifikate, DNS-Einträge und externe URLs nicht über provisorische Hostnamen oder wechselnde IP-Adressen abgebildet werden. Die Domäne dient somit nicht als zusätzliches Sicherheitsmerkmal, sondern als einheitlicher Namensraum für die öffentlich exponierten Verwaltungs- und Zugangspunkte des Labors.

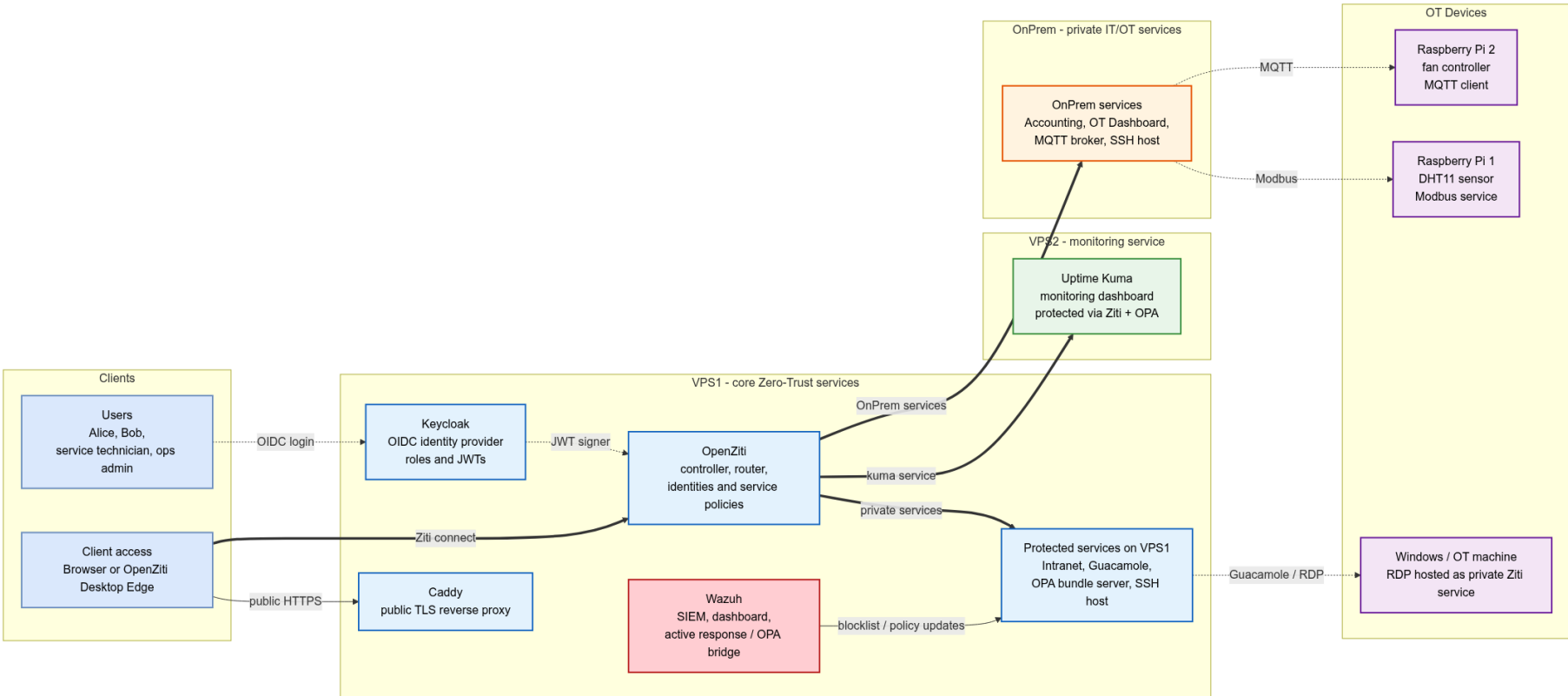


Abbildung 2.5.: Übersicht der implementierten Zero-Trust-Architektur

Abbildung 2.5 zeigt die Gesamtübersicht der implementierten Zero-Trust-Architektur. Sie dient bewusst als Landkarte der beteiligten Zonen und Hauptkomponenten. Die Detailbeziehungen werden in den folgenden Teilabbildungen aufgeteilt, damit der jeweilige Zugriffspfad ohne die übrigen Querverbindungen nachvollziehbar bleibt.

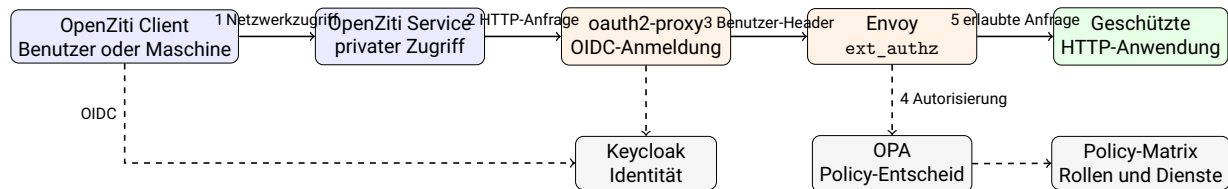


Abbildung 2.6.: Vereinfachter Zugriffspfad der implementierten Zero-Trust-Architektur (Schritt 4 bezeichnet die Autorisierungsanfrage von Envoy an OPA)

Die Abbildung 2.6 zeigt den vereinfachten Zugriffspfad für HTTP-basierte Dienste. OpenZiti entscheidet zunächst, ob eine Identität einen bestimmten Dienst auf Netzwerkebene erreichen darf. Danach erzwingt oauth2-proxy die Benutzeranmeldung über Keycloak. Envoy fragt OPA vor der Weiterleitung an die Anwendung, ob die angemeldete Identität gemäss Policy-Matrix für den Dienst berechtigt ist. Diese zweistufige Kontrolle entspricht dem Ziel, Netzwerkzugriff und fachliche Autorisierung nicht gleichzusetzen.

Der browserbasierte Wartungszugriff wird zusätzlich über Apache Guacamole abgebildet. Für die Zero-Trust-Perspektive ist Guacamole selbst als OpenZiti-Service modelliert, welcher nur durch die Rolle `service-technicians` erreicht werden darf. Für RDP-Zugriffe verwendet der Stack zusätzlich eine Maschinenidentität, die den privaten OpenZiti-Service `win-rdp` dialt; dieser bindet auf der Zielseite an `127.0.0.1:3389`.

2.4.1.2. Wartungszugriff ohne Client-Installation

Der Service-Techniker greift im Laborszenario nicht direkt mit einem lokal installierten OpenZiti Desktop Edge Client auf das Zielsystem zu. Stattdessen erfolgt der Zugriff browserbasiert über Guacamole und anschliessend per RDP auf ein internes Zielsystem. Diese Variante wurde bewusst gewählt, um zu zeigen, dass kontrollierter Wartungszugriff auch dann möglich ist, wenn auf dem Endgerät des externen Technikers keine spezifische Zero-Trust-Clientsoftware installiert werden soll. Ein Techniker kann dadurch einen eigenen Laptop verwenden und benötigt lediglich einen Browser sowie eine erfolgreiche Anmeldung mit der Rolle `service-technicians`. Der eigentliche private Netzwerkzugriff bleibt dennoch innerhalb der Laborumgebung kontrolliert, weil Guacamole beziehungsweise die zugehörige Maschinenidentität den RDP-Dienst über OpenZiti erreicht. Der resultierende Zugriffspfad ist in Abbildung 2.7 dargestellt.

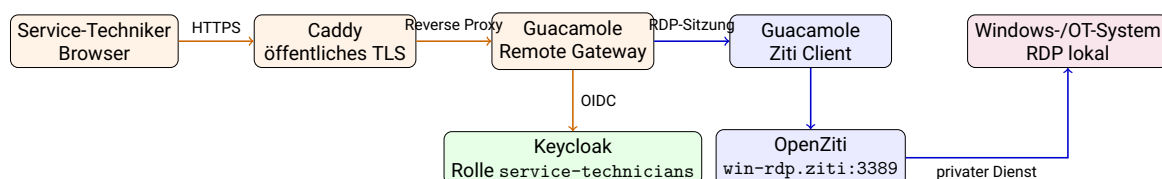


Abbildung 2.7.: Browserbasierter Wartungszugriff über Guacamole und OpenZiti

2.4.2. Identitätsbasierter Netzwerkzugriff mit OpenZiti

OpenZiti bildet die private Netzwerkebene der Zero-Trust-Architektur. Die Konfiguration wird beim Start durch den Bootstrap-Prozess eingerichtet. Das entsprechende Skript meldet sich am OpenZiti Controller an, richtet globale Router-Policies ein, konfiguriert Keycloak als externen JWT-Signer und lädt die aus der Policy-Matrix generierten OpenZiti-Ressourcen. Die Access-Matrix wird erzeugt und enthält die dienstspezifischen Identitäten, Services, Konfigurationen und Service Policies.

Jeder private Dienst besitzt in der Policy-Matrix eine Zieldefinition und einen Intercept. Die Zieldefinition beschreibt, wohin der OpenZiti-Host-Tunnel den Verkehr weiterleitet. Der Intercept beschreibt den privaten Namen und Port, den ein berechtigter Client verwendet. Für den DHT11-Modbus-Dienst lautet der Intercept beispielsweise `ot-dht11.ziti:502`, während das tatsächliche Ziel auf dem Raspberry Pi `127.0.0.1:1502` ist. Dieses Modell reduziert die Abhängigkeit von Netzwerksegmenten, weil die Erreichbarkeit nicht aus einer IP-Zugehörigkeit entsteht, sondern aus einer expliziten Service Policy.

OpenZiti unterscheidet zwischen Identitäten, die Dienste anbieten, und Identitäten, die Dienste nutzen. Host-Identitäten wie `intranet-host` oder `onprem-services-host` oder die SSH-Host-Identitäten `vps1-ssh-host`, `vps2-ssh-host` und `onprem-ssh-host` binden Dienste in das private Netz ein. Benutzer- und Maschinenidentitäten erhalten über Rollen wie `intranet-users` oder `ot-dashboard-users` Zugriff auf ausgewählte Dienste. Dadurch werden Kommunikationsbeziehungen explizit modelliert und nicht durch implizites Vertrauen in ein internes Netz ersetzt. Das Zusammenspiel aus Keycloak-Identitäten, OpenZiti-Controller und Service-Policies zeigt Abbildung 2.8.

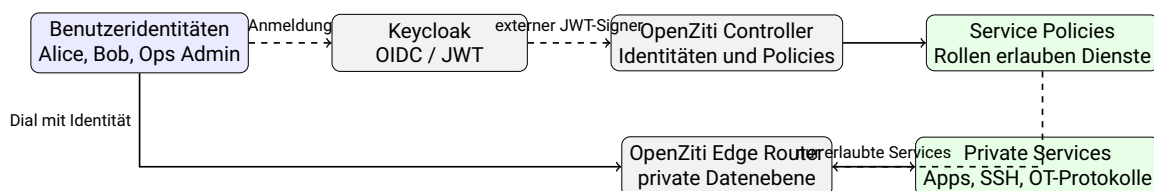


Abbildung 2.8.: Zusammenspiel von Keycloak-Identitäten und OpenZiti-Service-Policies

Der administrative Zugriff auf die Server ist ebenfalls über OpenZiti abgebildet. Die Identität `ops-admin` authentifiziert sich über Keycloak und erhält über die Rolle `ssh-admins` Zugriff auf die privaten SSH-Services `vps1-ssh.ziti:22`, `vps2-ssh.ziti:22` und `onprem-ssh.ziti:22`. Auf den jeweiligen Zielsystemen läuft eine OpenZiti-Host-Identität, welche den OpenZiti-Service an den lokalen SSH-Dienst `127.0.0.1:22` bindet. Damit wird die administrative Verwaltungsebene nicht mehr über allgemein erreichbare SSH-Endpunkte modelliert, sondern als identitätsbasierter Zero-Trust-Zugriffspfad.

2.4.3. Authentifizierung mit Keycloak

Keycloak übernimmt die Rolle des zentralen Identity Providers für menschliche Benutzer. Der Realm wird beim Start aus einem Export importiert und enthält Benutzer, OIDC-Clients und Rollen. Für die Laborumgebung sind unter anderem Alice, Bob und der Service-Techniker als Benutzer angelegt. Die Konfiguration enthält zudem OIDC-Clients für verschiedene geschützte Anwendungen, etwa Intranet, Uptime Kuma, OT-Dashboard, Accounting und Guacamole.

Für OpenZiti wird Keycloak zusätzlich als externer JWT-Signer eingebunden. Dadurch können menschliche Benutzer über OpenZiti Desktop Edge Clients authentifiziert werden, ohne dass für jeden Benutzer eine klassische Dateiidentität ausgerollt werden muss. Abbildung 2.9 zeigt den Desktop Client mit den verfügbaren privaten Diensten. Entscheidend ist dabei, dass die OpenZiti-Identität über eine externe ID mit dem Keycloak-Subjekt verknüpft wird. Diese Verbindung wird im Bootstrap-Prozess eingerichtet und erlaubt es, Keycloak-Authentifizierung und OpenZiti-Service-Policies miteinander zu verbinden.

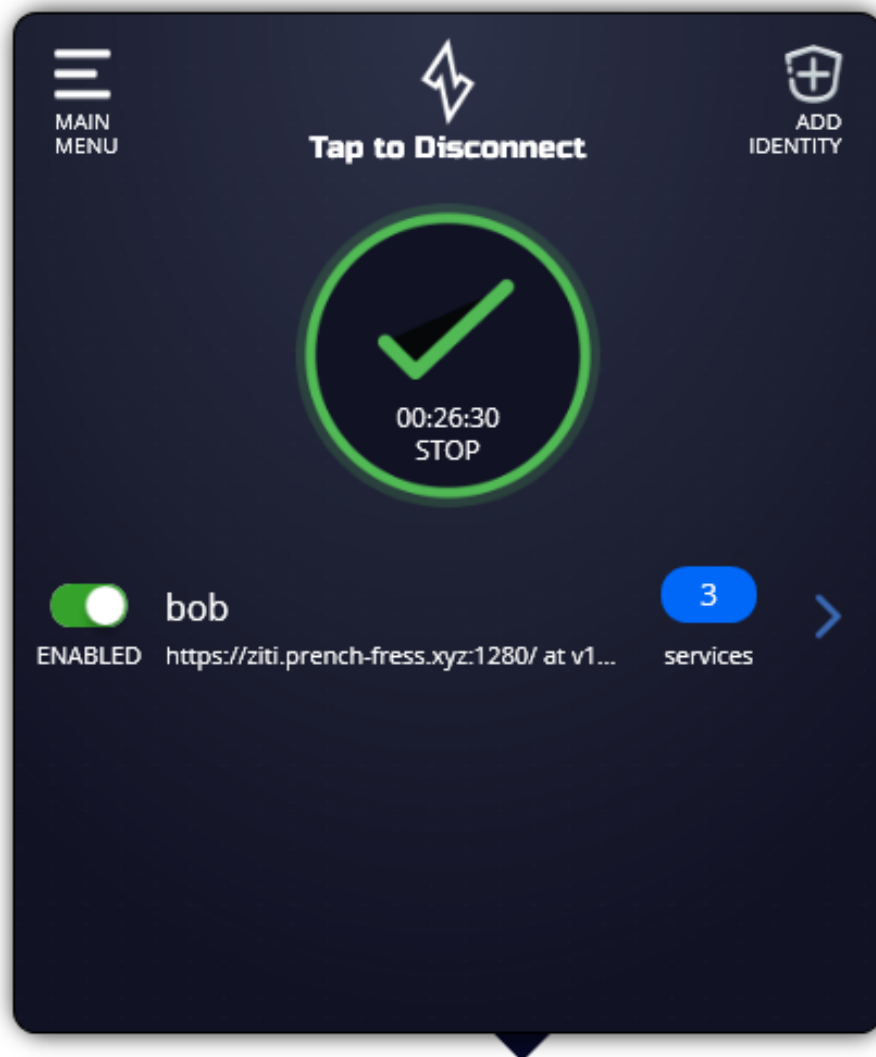


Abbildung 2.9.: OpenZiti Desktop Edge Client mit verfügbaren privaten Diensten

Alice und Bob repräsentieren in der Laborumgebung reguläre, verwaltete Benutzer des Unternehmens. Ihre Client-Systeme sind deshalb mit dem OpenZiti Desktop Edge Client ausgestattet und können nach erfolgreicher Anmeldung die jeweils freigegebenen privaten Dienste verwenden. Alice erhält darüber beispielsweise Zugriff auf Intranet und Accounting, während Bob auf Intranet, Uptime Kuma und das OT-Dashboard zugreifen kann. Der zuvor beschriebene Service-Techniker bildet bewusst ein anderes Zugriffsmuster ab: Für ihn wird kein installierter Ziti-Client auf dem eigenen Laptop vorausgesetzt, sondern der Zugriff erfolgt browserbasiert über Guacamole und den dort angebotenen RDP-Pfad.

Für HTTP-Anwendungen wird die Benutzeranmeldung über oauth2-proxy durchgeführt. oauth2-

proxy leitet Benutzer zur Keycloak-Anmeldung weiter und übergibt nach erfolgreicher Authentifizierung Identitätsinformationen als Header an den nachgelagerten Envoy-Proxy. Damit wird die Netzwerkverbindung durch OpenZiti von der anwendungsbezogenen Benutzeridentität getrennt, aber beide Ebenen werden im Zugriffspfad kombiniert.

2.4.4. Autorisierung mit Envoy und Open Policy Agent

Open Policy Agent wird als Policy Engine für HTTP-basierte Dienste eingesetzt. Envoy ruft OPA über `ext_authz` auf, bevor eine Anfrage an die eigentliche Anwendung weitergeleitet wird. Die OPA-Policies prüfen, ob eine authentifizierte Identität gemäss Policy-Matrix für den angefragten Dienst berechtigt ist. Beim Intranet wird zusätzlich berücksichtigt, ob ein Benutzer durch die Wazuh-OPA-Bridge blockiert wurde.

Die Regeln in OPA werden in Rego formuliert. Rego ist eine deklarative Policy-Sprache, mit der Entscheidungen über strukturierte Daten beschrieben werden können [14]. Im Unterschied zu imperativem Programmcode beschreibt Rego nicht primär eine Abfolge von Verarbeitungsschritten, sondern Bedingungen, unter denen eine Aussage wahr ist. Dadurch eignet sich Rego besonders für Autorisierungsentscheidungen, da Regeln zentral definiert, versioniert, getestet und unabhängig vom Anwendungscode weiterentwickelt werden können.

Listing 2.1 zeigt eine vereinfachte Rego-Regel aus dem Laborszenario. Die Regel prüft, ob der angemeldete Benutzer über mindestens eine Rolle verfügt, die für den Dienst `accounting` in der Policy-Matrix als Dial-Rolle hinterlegt ist.

```
1 authorized_for_service if {  
2   service := data.access_matrix.services["accounting"]  
3   principal := data.access_matrix.principals[username]  
4   some role in principal.roles  
5   role in service.dial_roles  
6 }
```

Listing 2.1: Vereinfachte Rego-Regel zur Autorisierung des Accounting-Dienstes

Die Regel greift zunächst auf den Dienst `accounting` innerhalb der generierten Access-Matrix zu. Anschliessend wird anhand des Benutzernamens der zugehörige Principal geladen. Mit `some role in principal.roles` wird über die Rollen des Benutzers iteriert. Die Bedingung `role in service.dial_roles` prüft, ob eine dieser Benutzerrollen auch in den erlaubten Rollen des Dienstes enthalten ist. Sobald eine Übereinstimmung besteht, ist die Regel erfüllt und der Zugriff kann erlaubt werden.

Die Autorisierung ist dadurch von den Anwendungen entkoppelt. Die Anwendungen müssen die rollenbasierte Zugriffskontrolle nicht selbst implementieren, sondern werden durch vorgeschaltete Komponenten geschützt. Die Policy-Daten liegen als generierte JSON-Dateien in den OPA-Policy-Verzeichnissen der jeweiligen Dienste. Dieses Muster wird für den zentralen Intranet-Dienst, für Uptime Kuma sowie für die On-Prem-OT-Dienste wie OT-Dashboard und Accounting verwendet.

Ein wesentlicher Vorteil dieses Ansatzes ist die Trennung von Anwendungscode, Policy-Logik und Policy-Daten. Die Anwendung stellt die fachliche Funktion bereit, Envoy setzt die Entscheidung im Datenpfad durch, OPA bewertet die Anfrage anhand der Rego-Regeln und die Access-Matrix liefert die zugrunde liegenden Rollen- und Dienstinformationen. Dadurch können Zugriffsregeln konsistent für mehrere Dienste angewendet und Änderungen an Rollen oder Dienstfreigaben zentral

nachvollzogen werden. Abbildung 2.10 fasst diese Rollenverteilung zwischen Policy Enforcement, Policy Decision und Policy-Daten zusammen.

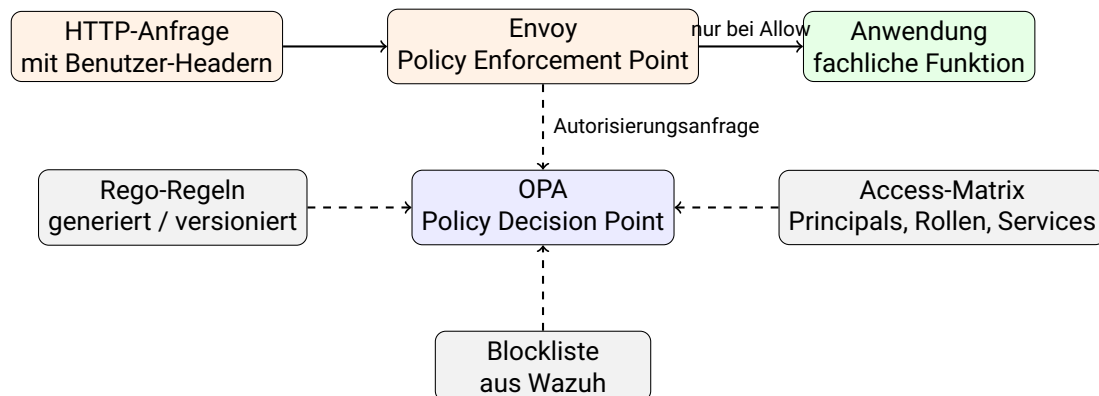


Abbildung 2.10.: Trennung von Policy Enforcement, Policy Decision und Policy-Daten

Für Health-Checks enthalten die Policies definierte Ausnahmen für lokale Prüfungen. Dadurch können Container-Orchestrierung und Bereitschaftsprüfungen funktionieren, ohne dass reale Benutzeranfragen ohne Authentifizierung erlaubt werden. Der produktive Anwendungspfad bleibt an authentifizierte Header und die Rollen der Policy-Matrix gebunden.

Die Rego-Regeln wurden im Lab bewusst einfach gehalten. Im Fokus der Arbeit steht der Vergleich zwischen einer Zero-Trust-basierten und einer perimeterbasierten Architektur. Eine sehr komplexe Policy-Logik hätte zusätzliche Einflussfaktoren in die Evaluation eingeführt und die Vergleichbarkeit erschwert. Deshalb beschränkt sich die Umsetzung im Wesentlichen auf rollenbasierte Entscheidungen, die direkt aus der Access-Matrix abgeleitet werden.

Mit Rego wären jedoch deutlich feinere Autorisierungsregeln möglich. Beispielsweise könnten zeitbasierte Einschränkungen umgesetzt werden, sodass administrative Zugriffe nur während definierter Wartungsfenster erlaubt sind. Ebenso denkbar wären Regeln, die bestimmte HTTP-Methoden einschränken, etwa lesende Zugriffe per GET erlauben, schreibende Zugriffe per POST oder DELETE aber nur für administrative Rollen zulassen. Weitere mögliche Erweiterungen wären Einschränkungen auf bestimmte Client-Identitäten, Gerätestatus, Authentifizierungsstärken oder Risikostufen aus einem Monitoring-System.

Diese erweiterten Regeln wurden nicht umgesetzt, weil sie für die Beantwortung der Forschungsfrage nicht erforderlich sind. Das Ziel des Labors ist nicht, möglichst komplexe OPA-Policies zu demonstrieren, sondern die Auswirkungen der beiden Sicherheitsarchitekturen auf Zugriffskontrolle, Angriffsfläche, Erkennung, Eindämmung und Betriebsaufwand vergleichbar zu untersuchen. Die einfache rollenbasierte Rego-Logik stellt sicher, dass die Autorisierungsentscheidungen nachvollziehbar bleiben und die Ergebnisse nicht durch zusätzliche kontextabhängige Sonderregeln verzerrt werden.

2.4.5. Monitoring und dynamische Zugriffseinschränkung mit Wazuh

Wazuh ergänzt die Architektur um SIEM- und Endpoint-Security-Funktionen. Der Wazuh-Stack besteht aus Wazuh Indexer, Manager, Dashboard und ergänzenden Hilfsdiensten. Der Manager nimmt Agentenverkehr entgegen und kann über Active Response auf sicherheitsrelevante Ereignisse reagieren.

Für die dynamische Zugriffseinschränkung wurde eine Wazuh-OPA-Bridge implementiert. Der vorgesehene Ablauf lautet: Wazuh erzeugt einen Alert, eine Active Response leitet diesen an die Bridge weiter, die Bridge ermittelt den betroffenen Benutzer, speichert ihn in einer Blockliste und synchronisiert diese Daten nach OPA. Die OPA-Policy kann anschliessend Anfragen dieses Benutzers ablehnen. Die Bridge berücksichtigt dabei unterschiedliche mögliche Benutzerfelder aus Wazuh-Alerts und verwendet eine Mapping-Datei, um lokale Benutzer- oder Agentennamen auf Keycloak-/OpenZiti-Benutzer abzubilden. Der Ablauf ist in Abbildung 2.11 zusammengefasst.

Diese Funktion ist insbesondere für HTTP-Dienste relevant, die über Envoy und OPA geschützt werden. Die Wazuh-OPA-Bridge kann zusätzlich OpenZiti-Identitäten deaktivieren und aktive Sessions entfernen. Dies wurde in den Evaluationsszenarien für Benutzer- und Host-Identitäten genutzt. Für rein fachliche Protokollentscheidungen innerhalb nicht-HTTP-Diensten wie MQTT oder Modbus wären darüber hinaus dienstspezifische Quarantäne- oder Autorisierungsmechanismen erforderlich.

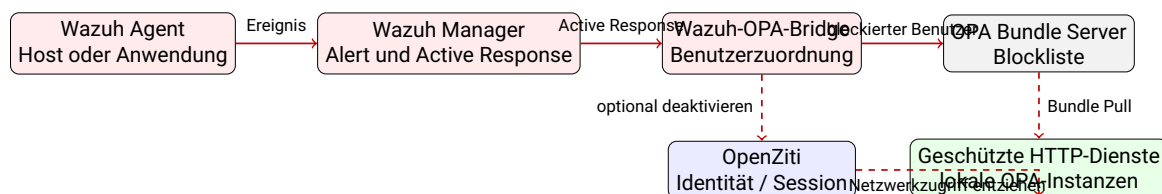


Abbildung 2.11.: Dynamische Zugriffseinschränkung über Wazuh, OPA und OpenZiti

2.4.6. Private Dienste und nicht exponierte Ressourcen

Ein zentrales Merkmal der implementierten Zero-Trust-Architektur ist, dass Dienste nicht pauschal im Netzwerk exponiert werden. Viele Komponenten veröffentlichen keine Host-Ports oder sind nur innerhalb eines Docker-Netzes beziehungsweise lokal auf einem Gerät erreichbar. Der Zugriff erfolgt über OpenZiti-Services und die zugehörigen Policies.

Das On-Prem-OT-Bundle veröffentlicht beispielsweise keine Host-Ports für Mosquitto oder das Dashboard, sondern bindet diese über die Service-Identität `onprem-services-host` in das OpenZiti-Netz ein. Auch Uptime Kuma wird über eine eigene Remote-Service-Struktur mit OPA, Envoy, oauth2-proxy und OpenZiti-Host bereitgestellt. Dasselbe Muster gilt für den bereits beschriebenen DHT11-Modbus-Dienst sowie für die privaten SSH-Zugänge der drei Server. Abbildung 2.12 zeigt diese Einbindung der On-Prem- und OT-Dienste in das private OpenZiti-Netz.

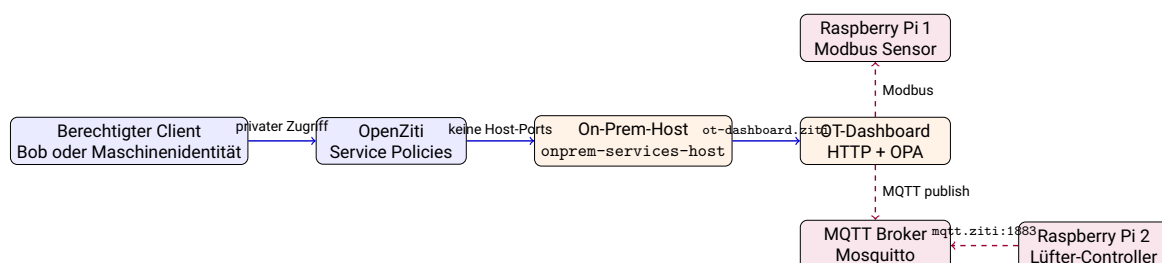


Abbildung 2.12.: Einbindung privater On-Prem- und OT-Dienste über OpenZiti

Damit verschiebt sich der Schutzzschwerpunkt von der Frage, ob sich ein Client in einem internen Netz befindet, zur Frage, ob eine konkrete Identität einen konkreten Dienst nutzen darf. Diese

Eigenschaft ist für die spätere Evaluation der Angriffsfläche, der lateralen Bewegung und der Betriebsaufwände zentral.

2.5. Perimeterbasierte Architektur

Dieser Abschnitt beschreibt die implementierte perimeterbasierte Vergleichsarchitektur und stellt dar, welche Vertrauensannahmen und Zugriffswege sie gegenüber der Zero-Trust-Architektur verwendet.

2.5.1. Architekturüberblick

Die perimeterbasierte Vergleichsarchitektur bildet ein klassisches Netzmodell mit zentralem VPN-Zugang, internen Netzbereichen und veröffentlichten Diensten ab. Im Zentrum stehen dabei nicht einzelne dienstbezogene Identitäten, sondern Netzzugehörigkeit, Routing, Firewall-Regeln und vorgeschaltete Anmeldekomponenten. Ein grober Überblick über die Architektur wird in der Abbildung 2.13 gegeben. Die Architektur dient als Gegenmodell zur Zero-Trust-Variante aus Abschnitt 2.4, in der Zugriffe explizit über OpenZiti-Services, Identitäten und Policies modelliert werden.

Der zentrale WireGuard-Endpunkt läuft auf VPS1. Die OPNsense-Firewall im Labor bildet die Unternehmensseite des Perimeters ab. Sie verbindet die Laborumgebung mit dem Schul-Lab-WAN, terminiert den WireGuard-Tunnel zur VPS1-Seite und trennt die internen Netze. Diese Konstruktion war notwendig, weil die Laborumgebung nicht direkt aus dem Internet erreichbar gemacht werden konnte und das Schul-Lab-WAN zwischen Internet und OPNsense liegt. VPS1 fungiert damit als von aussen erreichbarer VPN-Gegenpunkt, während OPNsense die lokalen LAN- und OT-Netze anbindet.

Die interne Laborumgebung ist in zwei relevante Netze gegliedert. Das LAN 192.168.1.0/24 enthält die Client-Systeme. Das OT- beziehungsweise Service-Netz 192.168.2.0/24 enthält die On-Premises-Dienste und OT-nahen Komponenten.

Die perimeterbasierten Anwendungsdienste werden als separate Dienstinstanzen beziehungsweise Overlay-Komponenten betrieben. Der Intranet-Perimeterdienst läuft auf VPS1, Uptime Kuma besitzt auf VPS2 eine eigene Perimeterinstanz, und auf dem On-Premises-Server laufen eigene Perimeterpfade für OT-Dashboard und Accounting. Vor den HTTP-Diensten steht weiterhin `oauth2-proxy`, sodass Benutzer über Keycloak authentifiziert werden. Der Unterschied zur Zero-Trust-Variante liegt daher nicht darin, dass die Dienste ohne Anmeldung erreichbar wären, sondern darin, dass Netzwerkzugang und Diensterreichbarkeit durch VPN, geroutete Netze und veröffentlichte Dienstpfade beschrieben werden und nicht durch dienstgenaue OpenZiti-Dial-Policies.

Der Wartungszugriff des Service-Technikers auf das OT-Netz erfolgt auch in der perimeterbasierten Variante browserbasiert über Guacamole. Guacamole fungiert dabei als Jump-Host: Der Techniker meldet sich am Gateway an und erreicht die OT-Systeme von dort über den gerouteten Netzpfad beziehungsweise den VPN-Kontext. Im Unterschied zur Zero-Trust-Variante wird der dahinterliegende Zugriff jedoch nicht durch dienstgenaue Identitäts-Policies begrenzt, sondern durch die Netz- und Routingkonfiguration des Jump-Host-Kontexts.

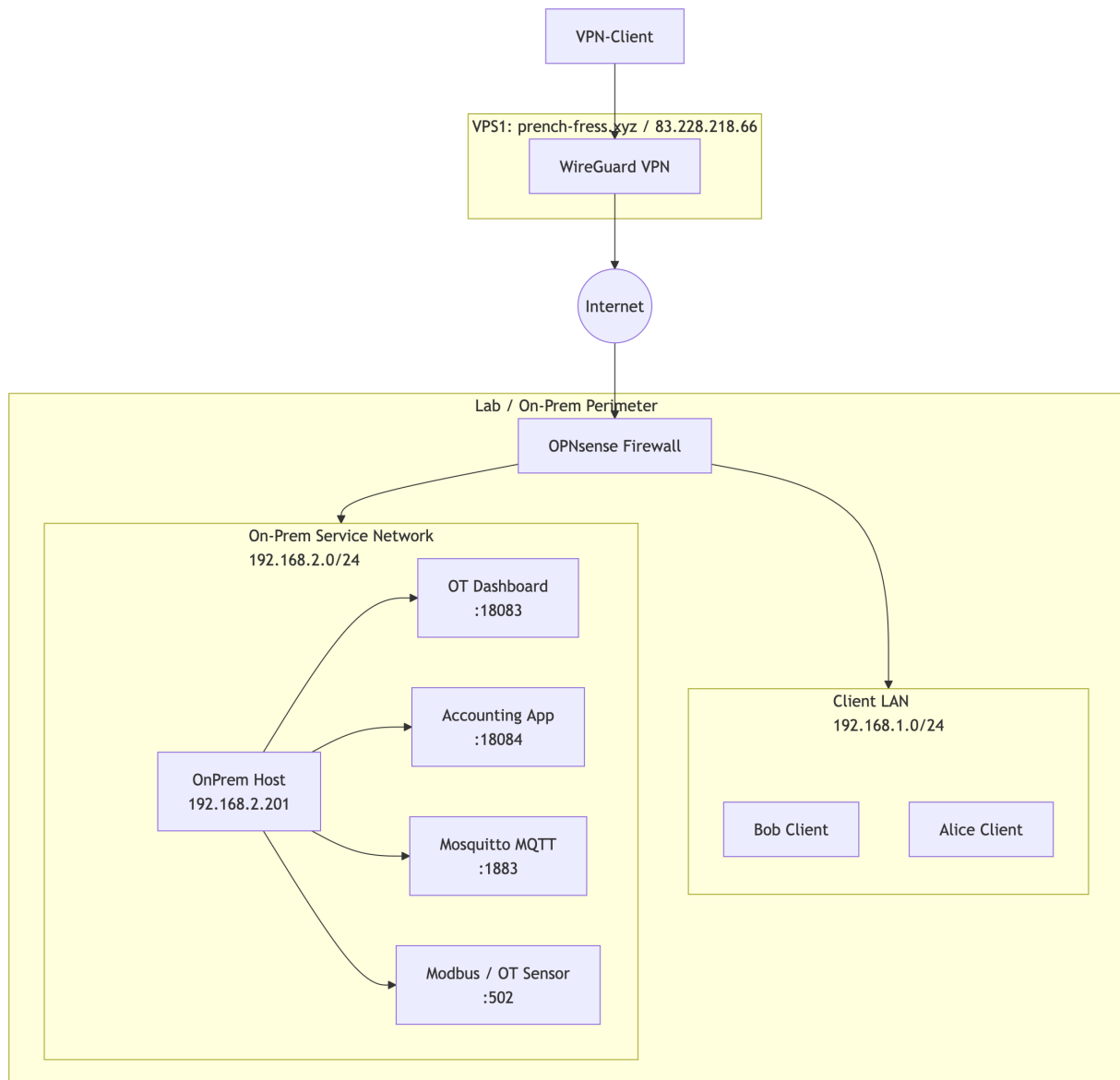


Abbildung 2.13.: Übersicht der implementierten perimeterbasierten Architektur

2.5.2. Remote Access über WireGuard

WireGuard stellt den Remote-Access-Mechanismus der perimeterbasierten Architektur bereit. Auf VPS1 läuft WireGuard containerisiert und verwendet das VPN-Netz 10.200.0.0/24. Die VPS1-Seite besitzt darin die Adresse 10.200.0.1/24. In der Laufzeitkonfiguration existieren zwei wesentliche Peer-Typen: ein OPNsense- beziehungsweise Unternehmensnetz-Peer und ein Remote-User-Peer.

Der OPNsense-Peer verbindet VPS1 mit den internen Labornetzen. Über diesen Peer werden das Client-LAN 192.168.1.0/24 und das OT-/Service-Netz 192.168.2.0/24 geroutet. Der Remote-User-Peer repräsentiert externe Benutzer oder Service-Techniker, die sich aus einem nicht lokalen Netz per VPN in die Laborumgebung einwählen. Damit entsteht im Perimetermodell ein vertrauenswürdiger Netzkontext: Wer einen gültigen VPN-Peer besitzt und dessen Verkehr geroutet wird, kann grundsätzlich die durch Routing und Firewall erlaubten Ziele erreichen.

Für die Evaluation wurden mehrere Prüfungen aus dem VPN-Kontext durchgeführt, weil dies den externen Zugriff reproduzierbar abbildet. Die daraus abgeleiteten Aussagen gelten jedoch nicht ausschliesslich für externe VPN-Clients. Ein Client im Labor-LAN erreicht dieselben perimeterbasierten Dienstpfade ebenfalls über seine Netzposition, sofern Routing, Namensauflösung und Firewall-Regeln dies zulassen. Für HTTP-Dienste bleibt zusätzlich die Anmeldung über oauth2-proxy und Keycloak erforderlich.

2.5.3. Interne Erreichbarkeit und implizites Vertrauen

Die interne Erreichbarkeit ergibt sich in der perimeterbasierten Architektur aus Netzpfaden, Firewall-Regeln und laufenden Diensten. Die OPNsense-Firewall bildet dabei den Übergang zwischen Schul-Lab-WAN, LAN, OT-/Service-Netz und VPN-Kontext. In der Laborumgebung waren die internen Regeln bewusst pragmatisch gehalten, damit die Vergleichsszenarien zuverlässig durchgeführt werden konnten. Für die architektonische Einordnung ist deshalb weniger eine fein ausgearbeitete produktive Regelbasis massgeblich, sondern das grundsätzliche Perimetermodell: Zugriff entsteht durch Netzmitgliedschaft, geroutete Zielnetze und erlaubte Dienstpfade. Diese bewusst einfach gehaltene Regelbasis begrenzt zugleich die Verallgemeinerbarkeit des Vergleichs auf stärker gehärtete Perimeterarchitekturen (vgl. Abschnitt 4.2).

In einer produktionsnäheren Variante müssten die Übergänge zwischen VPN, LAN und OT-Netz auf die jeweils notwendigen Kommunikationsbeziehungen beschränkt werden. Die untersuchte Laborarchitektur bildet diese Einschränkung jedoch nicht als identitäts- und dienstbezogene Policy ab, sondern verlässt sich auf die Kombination aus VPN-Peer, Routing, Firewall und Anwendungsanmeldung.

Damit unterscheidet sich die Angriffsfläche deutlich von der Zero-Trust-Architektur. In Zero Trust sieht eine Identität nur die Dienste, für die explizite Dial-Policies bestehen. In der perimeterbasierten Variante wird ein kompromittierter Client oder ein missbrauchter Technikerzugang dagegen über seine Netzposition bewertet: Entscheidend ist, welche Netze und Dienste aus dieser Position erreichbar sind und welche Anwendungskomponenten zusätzlich eine Anmeldung verlangen.

2.5.4. Abgrenzung zur Zero-Trust-Variante

Die perimeterbasierte Architektur verwendet weiterhin Authentifizierung für die HTTP Anwendungen. Keycloak und `oauth2-proxy` stellen sicher, dass Benutzer sich vor dem Zugriff auf die jeweiligen Webdienste anmelden müssen. Diese Anmeldung ersetzt jedoch keine dienstbezogene Netzwerkautorisierung. Ist ein Dienstpfad aus einem Netz erreichbar, entscheidet erst die Anwendungsschicht beziehungsweise der vorgeschaltete `oauth2-proxy` über die Benutzeranmeldung. Die Netzwerkebene selbst kennt keine OpenZiti-ähnliche Liste individuell sichtbarer Dienste pro Identität.

Auch die Eindämmung folgt einem gröberen Modell. Während die Zero-Trust-Architektur Benutzer- oder Host-Identitäten gezielt deaktivieren kann, liegen praktische Massnahmen im Perimeter eher bei der Sperrung eines Anwendungskontos, der Änderung eines VPN-Peers, der Anpassung von Routen oder Firewall-Regeln oder dem Abschalten eines Dienstpfads. Solche Massnahmen sind wirksam, können aber grössere Kollateraleffekte haben, weil sie häufig an Netz-, Peer- oder Dienstpfadgrenzen ansetzen statt an einer einzelnen dienstbezogenen Identität.

Eine automatische Eindämmung ist in dieser Architektur nicht als inhärente Funktion vorgesehen. Der Grund liegt nicht darin, dass einzelne Sperrmassnahmen technisch unmöglich wären, sondern darin, dass die Perimeterarchitektur keine kontinuierliche, identitäts- und dienstbezogene Verifikation als zentrales Prinzip verwendet. Zugriffsentscheidungen entstehen primär an VPN-, Routing-, Firewall- und Anwendungspunkten. Ein Alarm kann deshalb nicht ohne zusätzliche Integrationslogik direkt in eine präzise Sperrung einer einzelnen dienstbezogenen Identität übersetzt werden. Eine solche Automatisierung müsste mehrere voneinander getrennte Kontrollpunkte koordinieren, etwa WireGuard-Peers, OPNsense- beziehungsweise Firewall-Regeln, Routen, Anwendungskonten und gegebenenfalls NAC- oder Endpoint-Systeme. Dadurch wäre die Umsetzung deutlich komplexer als in einer Architektur, in der Identitäten, Services und Policies bereits an einem gemeinsamen Enforcement-Modell hängen. Zusätzlich steigt das Risiko von Kollateraleffekten, weil automatische Änderungen im Perimeter häufig ganze Netzpfade, Peers oder Dienstgruppen betreffen können und nicht nur den konkret kompromittierten Zugriff.

Die perimeterbasierte Variante ist damit nicht als unsicherer Minimalaufbau zu verstehen, sondern als anderes Sicherheitsmodell. Sie ist einfacher zu betreiben, wenn neue Netzteilnehmer oder Dienstpfade ergänzt werden sollen, erfordert aber eine sorgfältige Pflege von VPN-Peers, Netzbereichen, Firewall-Regeln und veröffentlichten Diensten. Der Vergleich mit Zero Trust untersucht daher nicht nur, ob Benutzer authentifiziert werden, sondern auf welcher Ebene Zugriff sichtbar, begrenzt, protokolliert und im Fehlerfall eingeschränkt wird.

Weitere klassische Perimeter-Elemente wie eine dedizierte DMZ oder eine produktionsnahe VLAN-Feinsegmentierung wurden in dieser Arbeit nicht als eigenständige Varianten untersucht. OPNsense ist als Labor-Firewall und VPN-Gegenstelle Bestandteil der Umsetzung, die dokumentierte Vergleichsarchitektur bleibt jedoch bewusst auf die beobachtete WireGuard-/VPN-Architektur mit LAN-, OT-/Service-Netz und authentifizierten Dienstpfaden beschränkt.

3. Umsetzung und Evaluation

Dieses Kapitel dokumentiert die praktische Umsetzung der OT-Komponenten, die reproduzierbare Bereitstellung der Laborumgebung sowie die Durchführung und Auswertung der Evaluation.

3.1. OT-Implementierung

Die OT-Implementierung bildet im Laborszenario eine kleine, aber heterogene industrielle Umgebung nach. Sie kombiniert Sensordaten, ein industrielles Kommunikationsprotokoll, MQTT-basierte Steuerung und eine Weboberfläche. Ziel ist nicht die vollständige Nachbildung einer Produktionsanlage, sondern die Bereitstellung konkreter OT-Ressourcen, die in beiden Netzwerkarchitekturen geschützt und evaluiert werden können. Der Abschnitt leistet einen Beitrag zu Teilfrage 4.1, da er die konkrete technische Umsetzung der OT-Komponenten innerhalb der Laborumgebung beschreibt.

Die Umgebung besteht aus einem Luftfeuchtigkeits- und Temperatursensor vom Typ DHT11 sowie aus einer simulierten Lüfterkomponente, die auf einer LED-Matrix dargestellt wird. Der Sensor kommuniziert über Modbus TCP, während die Lüftersteuerung über MQTT umgesetzt ist. Die Komponenten sind so realisiert, dass sowohl lesende Zugriffe auf Sensorwerte als auch steuernde Zugriffe auf eine Aktorkomponente modelliert werden können. Dadurch berücksichtigt die Arbeit nicht nur klassische Webanwendungen, sondern auch OT-nahe Kommunikationsbeziehungen.

3.1.1. DHT11-Sensor als Modbus-Service

Der DHT11-Sensor misst Temperatur und Luftfeuchtigkeit. In Abbildung 3.1 ist der eingesetzte Sensor dargestellt. Die Sensordaten werden über einen Modbus-TCP-Server bereitgestellt. Der Zugriff auf den Modbus-Dienst erfolgt über den OpenZiti-Service `ot-dht11-modbus`.

Modbus TCP ist ein offenes, Ethernet-basiertes Kommunikationsprotokoll, das von der Modbus Organization selbst als das weltweit am weitesten verbreitete industrielle Kommunikationsprotokoll bezeichnet wird [10]. Um eine OT-nahe Kommunikationsschicht abzubilden, wird Modbus TCP daher für die Kommunikation mit dem Sensor verwendet. Der DHT11 selbst dient dabei als einfache Messkomponente, während Modbus TCP die industrielle Kommunikationsbeziehung im Laborszenario modelliert.

Der Sensor wurde mit einem Raspberry Pi 3 Model B verkabelt. Der Raspberry Pi liest die Sensorwerte aus und dient als Plattform für den Modbus-TCP-Server. Der DHT11 überträgt Temperatur- und Feuchtigkeitswerte über die SIG-Leitung als digitales, zeitkodierte Signal an einen GPIO-Pin des Raspberry Pi.

Die Registerbelegung ist in Tabelle 3.1 dokumentiert. Temperatur und Luftfeuchtigkeit werden jeweils mit dem Faktor zehn skaliert, damit Dezimalwerte als Integer übertragen werden können.

Zusätzlich enthält die Registerstruktur einen Statuswert, einen Zeitstempel und einen Fehlerzähler. Damit lassen sich neben den eigentlichen Messwerten auch der Zustand des Sensors und mögliche Lesefehler erfassen.

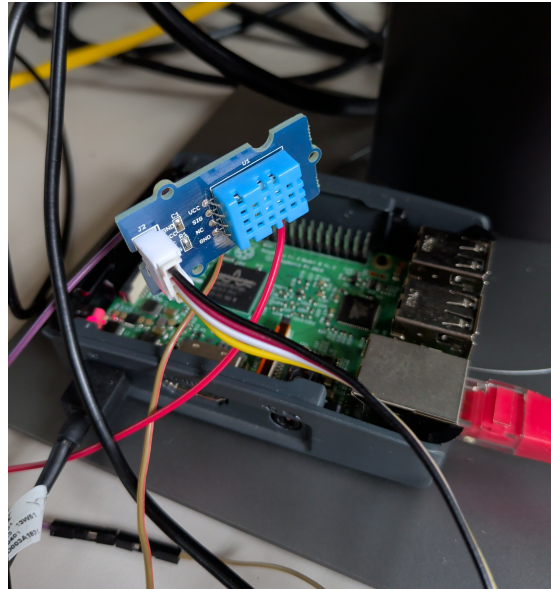


Abbildung 3.1.: Eingesetzter DHT11-Temperatur- und Luftfeuchtigkeitssensor

Register	Bedeutung
0	Temperatur in Grad Celsius, mit Faktor zehn skaliert und als vorzeichenbehafteter 16-Bit-Wert gespeichert.
1	Relative Luftfeuchtigkeit in Prozent, mit Faktor zehn skaliert.
2	Statuswert, wobei 0 einen erfolgreichen Sensorzugriff und 1 einen Lesefehler beschreibt.
3 und 4	Unix-Zeitstempel des letzten Updates, aufgeteilt in High- und Low-Word.
5	Kumulativer Fehlerzähler für Sensorlesefehler.

Tabelle 3.1.: Registermodell des DHT11-Modbus-Dienstes

Die Anbindung an OpenZiti erfolgt über eine Host-Identität. Der lokale Dienst bleibt auf dem Raspberry Pi gebunden, während berechtigte Clients ihn über den privaten Namen `ot-dht11.ziti` erreichen können. In der Policy-Matrix sind dafür die Rollen `ot-operators` und `service-technicians` als Dial-Rollen hinterlegt. Damit dürfen nur diese Rollen auf den Sensor zugreifen.

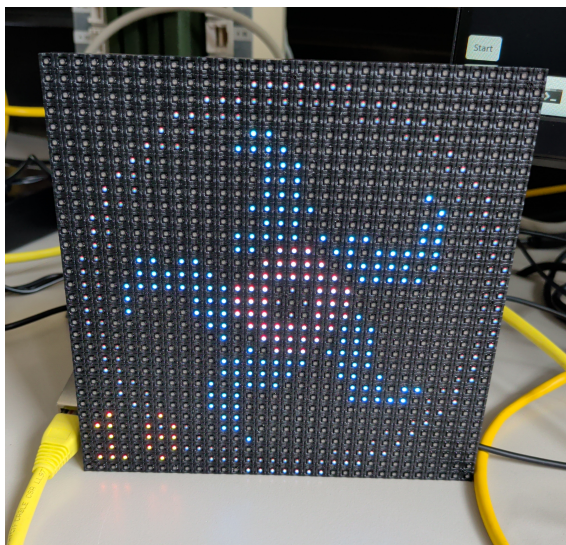
3.1.2. MQTT-Broker und Lüfter-Steuerung

MQTT ist ein standardisiertes, leichtgewichtiges Publish/Subscribe-Protokoll für IoT- und M2M-Anwendungen. Die offizielle MQTT-Webseite beschreibt MQTT als OASIS-Standard für IoT-Messaging und nennt den Einsatz in zahlreichen Branchen, darunter Automobilindustrie, Fertigung, Telekommunikation sowie Öl und Gas [11]. Die Steuerung des Lüfters erfolgt über MQTT, um zu zeigen, dass neben Modbus TCP auch weitere OT- beziehungsweise IoT-nahe Protokolle in der Architektur abgebildet werden können.

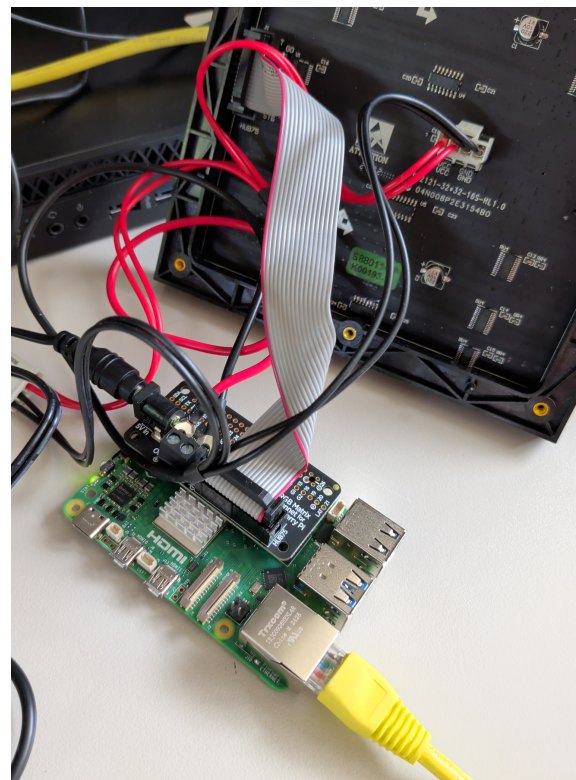
Der MQTT-Broker wird durch den Dienst Eclipse Mosquitto bereitgestellt. Er dient als Vermittlungsschicht zwischen dem Dashboard und dem simulierten Lüfter. Der MQTT-Server verwendet drei zentrale Topics: `ot/dht11/state` für veröffentlichte Sensorwerte, `ot/fan/cmd` für Steuerbefehle an den Lüfter und `ot/fan/state` für den zurückgemeldeten Zustand des Lüfters.

Der Lüfter wird auf einer LED-Matrix dargestellt, da der Aktorzustand dadurch einfach visuell nachvollziehbar ist. Da die LED-Matrix bereits aus einer früheren Arbeit verfügbar war, konnte die Lüfterkomponente ohne zusätzliche Beschaffung umgesetzt werden. Die Vorder- und Rückseite der LED-Matrix sind in Abbildung 3.2a beziehungsweise Abbildung 3.2b dargestellt. Verbunden wird die LED-Matrix über ein Bonnet der Marke Adafruit, das die Verkabelung vereinfacht.

Der Lüfter verbindet sich über MQTT mit dem Broker, abonniert das Command-Topic und setzt eine Ziel-Drehzahl. Der Zustand wird regelmässig als JSON-Nachricht veröffentlicht und enthält unter anderem aktuelle Drehzahl, Ziel-Drehzahl, Status, Zeitstempel, Zeitpunkt des letzten Befehls und Fehlerzähler.



(a) Vorderseite der LED-Matrix



(b) Rückseite der LED-Matrix

Abbildung 3.2.: LED-Matrix zur Simulation der Lüfter-Komponente

Der MQTT-Zugriff wird ebenfalls über OpenZiti modelliert. Der MQTT-Broker ist als privater Dienst `mqtt-broker` mit dem Intercept `mqtt.ziti` definiert. Maschinenidentitäten wie `ot-fan-client` erhalten Zugriff über die Rolle `mqtt-clients`. Der Service-Techniker darf gemäss Policy-Matrix ebenfalls auf den MQTT-Broker zugreifen.

3.1.3. OT-Dashboard

Das OT-Dashboard verbindet die Sensor- und Aktorseite. Die Implementierung liest zyklisch über Modbus TCP die DHT11-Register, wandelt die Registerwerte in ein JSON-Modell um und veröffent-

licht die Messwerte auf dem MQTT-Topic `ot/dht11/state`. Gleichzeitig stellt das Dashboard eine Weboberfläche sowie API-Endpunkte bereit, über die aktuelle Messwerte und der Zustand des Lüfters abgefragt werden können.

Für Steuerbefehle veröffentlicht das Dashboard JSON-Nachrichten auf `ot/fan/cmd`. Die Lüfter-Komponente konsumiert diese Nachrichten und publiziert ihren Zustand wieder auf `ot/fan/state`. Dadurch entsteht ein geschlossener Kommunikationsfluss zwischen Sensor, Dashboard, MQTT-Broker und Aktorkomponente. Das Dashboard verwendet zusätzlich eine eigene OpenZiti-Client-Identität, um den DHT11-Modbus-Dienst über den privaten OpenZiti-Service zu erreichen.

Das Dashboard selbst wird als geschützter HTTP-Dienst nach dem in Abschnitt 2.4 beschriebenen Zugriffsmuster bereitgestellt und über die Service-Identität `onprem-services-host` in OpenZiti gehostet. Berechtigte Benutzer erreichen die Oberfläche über `ot-dashboard.ziti`. Dadurch kombiniert das Dashboard OT-Kommunikation auf der Backend-Seite mit identitäts- und policybasierter Zugriffskontrolle auf der Benutzeroberfläche.

3.1.4. Einbindung in beide Netzwerkarchitekturen

Die OT-Komponenten bilden gemeinsame Ressourcen für den Vergleich der Netzwerkarchitekturen. In der Zero-Trust-Architektur erfolgt die Einbindung über OpenZiti-Services, Host-Identitäten und Dial-Rollen. Der DHT11-Modbus-Dienst, der MQTT-Broker und das OT-Dashboard (vgl. Abbildung 2.4) sind dadurch nicht als allgemein erreichbare interne Dienste modelliert, sondern als einzelne private Services mit expliziten Zugriffspolicies.

In der perimeterbasierten Vergleichsarchitektur (Abschnitt 2.5) liegen die OT-nahen Dienste dagegen im OT-/Service-Netz hinter der OPNsense-Firewall und werden aus dem Labor-LAN oder dem VPN-Kontext über die gerouteten Netze erreicht. Damit bleiben die fachlichen OT-Ressourcen vergleichbar, während sich das Zugriffsmodell von identitäts- und servicebezogener Freigabe zu netz- und dienstpfadbezogener Erreichbarkeit verschiebt.

3.2. Infrastructure as Code und Reproduzierbarkeit

Ein zentrales Ziel der Umsetzung besteht darin, eine möglichst reproduzierbare Laborumgebung bereitzustellen. Die relevanten Bestandteile werden deshalb nicht nur manuell eingerichtet, sondern weitgehend als Infrastructure as Code über versionierte Artefakte beschrieben. Dazu gehören Docker-Compose-Dateien, Shell-Skripte, Python-Skripte, Service-Konfigurationen, OpenZiti-Bootstrap-Artefakte, Keycloak-Realm-Exporte und OPA-Policy-Dateien. Dieser Ansatz erleichtert es, die Architektur nachvollziehbar aufzubauen, Änderungen zu prüfen und die technische Umsetzung mit der Dokumentation abzugleichen. Der Abschnitt leistet einen Beitrag zu Teilfrage 4.1, indem er beschreibt, wie die technische Implementierung reproduzierbar bereitgestellt wird.

Die Reproduzierbarkeit ist dabei nicht absolut: Externe Infrastruktur und physische Hardware können nicht vollständig durch das Repository erzeugt werden (vgl. Abschnitt 3.2.6). Die vorhandenen Artefakte bilden jedoch die logische Konfiguration und den grössten Teil der Dienste so ab, dass der Aufbau systematisch wiederholt werden kann.

3.2.1. Docker-Compose

Die Basis des Labors bildet der Containerisierungsdienst Docker. In den Docker-Compose-Dateien sind die eingesetzten Dienste, Netzwerke, Volumes und relevanten Startparameter festgehalten. Dadurch können zentrale Bestandteile der Laborumgebung reproduzierbar gestartet und miteinander verbunden werden. Listing 3.1 zeigt exemplarisch die Konfiguration des MQTT-Brokers Eclipse Mosquitto.

```
1  mosquito:
2    image: eclipse-mosquitto:2.0
3    restart: unless-stopped
4    volumes:
5      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
6      - mosquitto-data:/mosquitto/data
7      - mosquitto-log:/mosquitto/log
8    expose:
9      - "1883"
```

Listing 3.1: Konfiguration von Eclipse Mosquitto in der Docker-Compose-Datei

3.2.2. Bootstrap- und Deployment-Skripte

Der OpenZiti-Bootstrap ist in einem Shell-Skript umgesetzt. Das Skript wartet auf Controller und Keycloak, meldet sich am Controller an, erstellt globale Router-Policies, richtet Keycloak als externen JWT-Signer ein und lädt anschliessend die generierten OpenZiti-Ressourcen aus der Policy-Matrix. Damit wird der Zugriff auf Dienste nicht manuell über die OpenZiti-Konsole gepflegt, sondern reproduzierbar aus versionierten Daten erzeugt.

Weitere Skripte unterstützen die Inbetriebnahme und Wartung der Laborumgebung. Das Skript `access-matrix.py` validiert die Policy-Matrix und erzeugt daraus Dokumentation, OPA-Daten und OpenZiti-Konfigurationen. Skripte wie `export-ziti-identity.sh` dienen dem Export von Identitätsdateien für Remote-Hosts, darunter auch die Host-Identitäten für die privaten SSH-Dienste von VPS1, VPS2 und On-Prem. Für Wazuh existieren Skripte zur Zertifikatserzeugung, Dashboard-Konfiguration sowie Installation und Enrollment von Agenten. Die Service-Bundles enthalten eigene Deploy- und Installationsskripte, beispielsweise für Uptime Kuma, den On-Prem-OT-Stack, den DHT11-Modbus-Dienst und die Fan-Komponente.

Die Skripte reduzieren den Anteil nicht dokumentierter manueller Schritte. Gleichzeitig bleiben bestimmte Handlungen bewusst explizit, etwa das Kopieren von Identitäten auf Zielsysteme oder das Setzen von Secrets in Umgebungsdateien. Diese Schritte sind sicherheitsrelevant und können nicht sinnvoll als statische Werte im Repository abgelegt werden.

3.2.3. Policy-Matrix als zentrale Quelle

Die Datei `access-matrix.json` ist eine zentrale Quelle für Rollen, Identitäten, private Services und Plattformdienste. Aus ihr werden mehrere Artefakte abgeleitet. OPA-Daten werden in die Policy-Verzeichnisse der zentralen und entfernten Dienste geschrieben, um zu gewährleisten, dass alle Policy-Agents über die gleichen Rolleninformationen verfügen. Zusätzlich wird eine OpenZiti-Shell-Datei generiert, die Identitäten, Services, Konfigurationen und Service Policies anlegt oder aktualisiert.

Listing 3.2 zeigt einen gekürzten Ausschnitt aus der Datei `access-matrix.json`. Der Ausschnitt enthält beispielhaft eine Rolle, eine menschliche Identität sowie einen privaten Service inklusive Zieldefinition, Intercept, Bind-Identität und Dial-Berechtigungen.

```
1 {
2   "roles": {
3     "intranet-users": {
4       "description": "Human users allowed to access the intranet."
5     }
6   },
7   "identities": {
8     "alice": {
9       "type": "human",
10      "email": "alice@example.test",
11      "keycloak_id": "7863eee5-d45e-400f-94d7-6586c8e1fa93",
12      "description": "Accountant using Windows Client 1.",
13      "roles": [
14        "intranet-users",
15        "accounting-users"
16      ]
17    }
18  },
19  "services": {
20    "intranet": {
21      "category": "application",
22      "protocol": "tcp",
23      "target": {
24        "host": "intranet-oauth2-proxy",
25        "port": 4180
26      },
27      "intercept": {
28        "hostname": "intranet.ziti",
29        "port": 80
30      },
31      "bind_identities": [
32        "intranet-host"
33      ],
34      "dial_identities": [
35        "intranet-client"
36      ],
37      "dial_roles": [
38        "intranet-users",
39        "monitoring-clients"
40      ]
41    }
42  }
43 }
```

Listing 3.2: Ausschnitt aus der Datei access-matrix.json

Dieser Ansatz hat zwei Vorteile. Erstens wird die Zugriffspolitik an einer Stelle gepflegt, anstatt Rollen und Dienstfreigaben in mehreren Systemen unabhängig voneinander zu definieren. Zweitens kann die Matrix validiert werden, bevor daraus Konfigurationen entstehen. Das Skript `access-matrix.py` prüft beispielsweise, ob referenzierte Rollen und Identitäten existieren und ob Dienste gültige Ziele, Intercepts und Bind-Identitäten enthalten.

Die Policy-Matrix verbindet damit Dokumentation und technische Durchsetzung. Sie beschreibt nicht nur, wer worauf zugreifen soll, sondern liefert auch die Grundlage für OPA-Entscheidungen und OpenZiti-Service-Policies. Für eine wissenschaftliche Evaluation ist diese Nachvollziehbarkeit wesentlich, weil die bewerteten Zugriffsbeziehungen aus versionierten Projektartefakten

hervorgehen.

Die administrativen SSH-Zugriffe (vgl. Abschnitt 2.4) werden ebenfalls aus dieser Matrix erzeugt, einschliesslich der zugehörigen Host-Identitäten sowie der Bind- und Dial-Policies. Dadurch ist auch der Verwaltungszugriff Bestandteil der versionierten Zugriffspolitik und keine davon losgelöste Ausnahme.

3.2.4. Konfigurationsdateien der Sicherheitskomponenten

Neben Compose- und Skriptdateien enthält das Repository die relevanten Konfigurationsdateien der eingesetzten Sicherheitskomponenten. Keycloak wird über einen Realm-Export initialisiert, der Benutzer und OIDC-Clients enthält. OpenZiti wird über Bootstrap-Skripte und generierte Ressourcen konfiguriert. OPA verwendet Rego-Policies und die aus der Access-Matrix generierten JSON-Daten. Envoy-Konfigurationen definieren die vorgeschalteten Proxies und deren `ext_authz`-Anbindung an OPA.

Wazuh wird über eigene Manager-, Indexer- und Dashboard-Konfigurationen eingebunden. Die Active-Response-Konfiguration verweist auf ein Skript, das sicherheitsrelevante Ereignisse an die Wazuh-OPA-Bridge weiterleiten kann. Caddy wird über das `Caddyfile` als öffentlicher TLS-Endpunkt für definierte Plattformdienste konfiguriert. Mosquitto wird über eine eigene Konfigurationsdatei im On-Prem-OT-Bundle bereitgestellt. WireGuard besitzt eine eigene Konfiguration mit Interface, Peers und AllowedIPs.

Die Infrastructure-as-Code-Beschreibung beschränkt sich auf die tatsächlich eingesetzten Artefakte der OpenZiti-, Keycloak-, OPA-, Envoy-, Wazuh-, WireGuard- und Service-Komponenten. Einzelne Perimeter-Artefakte sind bewusst als Vergleichs- beziehungsweise Overlay-Dateien abgelegt und benötigen die passende Zielumgebung, etwa vorhandene Service-Dateien, Docker-Netzwerke oder eine nicht versionierte WireGuard-Laufzeitkonfiguration.

3.2.5. Herausforderungen bei der Umsetzung

Die praktische Umsetzung adressiert Teilfrage 4.2. Eine zentrale Herausforderung bestand darin, eine realitätsnahe Zero-Trust-Architektur ohne proprietäre Enterprise-Komponenten aufzubauen. Wie in Abschnitt 2.2 beschrieben, wurden deshalb Open-Source-Komponenten eingesetzt, die die logischen Rollen des NIST-Modells abbilden und zugleich im Rahmen einer wissenschaftlichen Laborumgebung betreibbar bleiben.

Eine zweite Herausforderung ergab sich aus der verteilten Laborinfrastruktur. Die lokalen Komponenten konnten nicht direkt aus dem Internet erreichbar gemacht werden, weshalb die perimeterbasierte Vergleichsarchitektur über die Kombination aus VPS1, WireGuard und OPNsense konstruiert werden musste (vgl. Abschnitt 2.5). Dadurch liess sich der Remote-Access-Kontext zwar reproduzierbar abbilden, gleichzeitig blieb die Erreichbarkeit einzelner On-Premises-Pfade von den Bedingungen des Schul-Lab-WAN abhängig.

Hinzu kamen der Modellierungsaufwand der Policy-Matrix sowie die Grenzen der Reproduzierbarkeit. Identitäten, Services, Configs sowie Bind- und Dial-Policies mussten konsistent beschrieben und aus versionierten Daten erzeugt werden. Externe Faktoren wie DNS-Einträge, Zertifikate, Secrets, OpenZiti-Identitätsdateien und physische OT-Hardware konnten dagegen nicht vollständig automatisiert werden. Diese Punkte zeigen, dass die technische Implementierung nicht nur aus der Installation einzelner Werkzeuge bestand, sondern aus der konsistenten Verbindung von Identitätsmodell, Diensten, Policies, Netzwerkpfaden und realer Hardware.

3.2.6. Grenzen der Reproduzierbarkeit

Trotz des Infrastructure-as-Code-Ansatzes bleiben Grenzen der Reproduzierbarkeit. Secrets wie Passwörter, Cookie-Secrets, private Schlüssel und API-Zugangsdaten dürfen nicht als feste Werte im Repository abgelegt werden. Sie werden über Umgebungsdateien wie `.env.example` und `.env.vps.example` vorbereitet und müssen für eine konkrete Installation ersetzt werden. Auch DNS-Einträge und öffentlich erreichbare Endpunkte, etwa für Keycloak, OpenZiti oder Wazuh, setzen externe Infrastruktur voraus.

Eine weitere Grenze betrifft physische oder hardwarenahe Komponenten. Raspberry Pis, der DHT11-Sensor und die RGB-LED-Matrix können zwar durch Skripte und Dienste vorbereitet werden, ihre physische Installation, Verkabelung und Verfügbarkeit lassen sich jedoch nicht vollständig aus dem Repository reproduzieren. Ähnliches gilt für die Verteilung von OpenZiti-Identitätsdateien auf entfernte Hosts, einschliesslich der SSH-Host-Identitäten `vps1-ssh-host`, `vps2-ssh-host` und `onprem-ssh-host`. Der Export und die Installation sind dokumentiert, die sichere Übertragung bleibt jedoch ein betrieblich kontrollierter Schritt.

Die vorhandenen Artefakte reichen damit aus, um die logische Architektur, die Dienste, die Policies und grosse Teile des Deployments nachzuvollziehen. Für eine vollständig identische Wiederholung sind jedoch zusätzlich die externe Infrastruktur, die nicht versionierten Secrets und die physische OT-Hardware erforderlich.

3.3. Durchführung der Evaluation

Die Evaluation wurde auf einer realen, verteilten Infrastruktur bestehend aus den drei Knoten (VPS1, VPS2, OnPrem) durchgeführt. Umgesetzt wurden die im Evaluationskonzept (Abschnitt 1.4) definierten Angriffs- und Wartungsszenarien. Jedes Szenario wurde sowohl in der Zero-Trust-Architektur als auch in der perimeterbasierten Architektur ausgeführt. Die vollständigen Einzelschritte, Kommandos, Zeitstempel und Rohdaten sind in den entsprechenden Anhängen dokumentiert. Der Abschnitt bildet die empirische Grundlage für Teilfrage 5.2, da hier die Vergleichsdaten für beide Architekturen erhoben werden.

3.3.1. Vorgehensweise

Alle Szenarien wurden als reproduzierbare Skripte umgesetzt. Die Skripte sind Bestandteil des Projekt-Repositorys; die daraus erzeugten Protokolle und Rohdaten sind in Anhang C dokumentiert. Dieses Vorgehen wurde gewählt, um die Abläufe zwischen den beiden Architekturen möglichst vergleichbar zu halten und manuelle Abweichungen während der Messung zu reduzieren. Die Skripte stellten sicher, dass Ausgangszustand, Angriff beziehungsweise Änderung, Verifikation, Datensammlung und Restore jeweils in derselben Reihenfolge durchgeführt und mit Zeitstempeln sowie Ergebnisartefakten dokumentiert wurden. Da die Bewertung nicht nur auf Endzuständen, sondern auch auf erzeugten Artefakten wie Logs, Routingtabellen, Docker-Zustand, WireGuard-Konfiguration, OpenZiti-Objekten, OPA-Entscheidungen und Bridge-Antworten basiert, mussten diese Daten systematisch und wiederholbar erhoben werden.

Auf eine quantitative Auswertung der Time to Detect wurde bewusst verzichtet, da sich bereits in den manuellen Testläufen zeigte, dass sie primär vom eingesetzten Endpoint-Schutz abhängt und nicht von der Netzwerkarchitektur (vgl. die Bewertung von Hypothese H2 in Abschnitt 3.4). Die vorgelagerte Erkennungszeit hängt davon ab, wie schnell Windows Defender einen Vorfall erkennt

und diesen an das SIEM meldet. Diese Leistung ist unabhängig vom Netzwerkmodell und liegt ausserhalb des Untersuchungsbereichs dieser Arbeit.

In den perimeterbasierten Angriffsläufen wurde zudem bewusst kein aktives Containment durchgeführt. In der untersuchten Perimeterimplementierung war keine vergleichbare, identitätsbezogene Containment-Mechanik vorhanden, und destruktive Firewall- oder Routingänderungen hätten die gemeinsam genutzte Laborumgebung beeinträchtigt. Die Eindämmungsmöglichkeiten der perimeterbasierten Architektur werden deshalb in Abschnitt 3.4 qualitativ eingeordnet.

Die Evaluation trennt praktische Evidenz von architektonischer Einordnung. Als praktische Evidenz gelten konkret erhobene Artefakte wie Logs, Zeitstempel, Routingtabellen, WireGuard-Zustand, Docker-Zustand, Portprüfungen, OpenZiti-Objekte, OPA-Blocklisten und Restore-Nachweise. Die architektonische Einordnung wird nur dort verwendet, wo die Artefakte alleine nicht ausreichend aussagekräftig sind oder eine zusätzliche Interpretation der Kontrollmechanismen notwendig ist.

3.3.2. Sicherheitsszenarien

Die Sicherheitsszenarien prüfen, wie beide Architekturvarianten auf typische Angriffs- und Missbrauchsfälle reagieren. Im Fokus stehen die Reichweite kompromittierter Identitäten, Endpunkte oder Dienste sowie die Möglichkeiten zur gezielten Eindämmung und Wiederherstellung.

3.3.2.1. Szenario A1: Phishing und Credential Theft

Szenario A1 setzt die in Abschnitt 1.4.2.1 definierte Ausgangslage um: Ein Angreifer verwendet per Phishing erlangte, formal gültige Zugangsdaten. Bewertet wird nicht, wie ein solcher Diebstahl verhindert werden kann, sondern wie die jeweilige Architektur auf den Einsatz dieser Zugangsdaten reagiert. Die Durchführungsschritte sind in Tabelle 3.2 zusammengefasst.

Phase	Zero Trust	Perimeter
Vorbereitung	Ausgangszustand der Ziti-Controller-Objekte und Service-Erreichbarkeit dokumentiert.	WireGuard-Schnittstellenstatus, Peers, AllowedIPs, Routen und veröffentlichte Ports dokumentiert.
Angriff	OIDC-Token-Request gegenüber Keycloak mit den kompromittierten Zugangsdaten. Anschliessend Zugriff auf Dienste über den OAuth2-Proxy versucht.	Erreichbarkeit von Diensten aus Netzposition des kompromittierten Accounts dokumentiert.
Eindämmung	Containment-Massnahme über die Wazuh-OPA-Bridge ausgelöst, um die kompromittierte Identität zu sperren.	Kein aktiver Eingriff durchgeführt.
Wiederherstellung	Sperre aufgehoben. Ausgangszustand wiederhergestellt und verifiziert.	Ursprünglicher Zustand dokumentiert.

Tabelle 3.2.: Durchführungsschritte Szenario A1

3.3.2.2. Szenario A2: Kompromittierter Endpoint

Szenario A2 operationalisiert die in Abschnitt 1.4.2.2 beschriebene vollständige Endpoint-Kompromittierung. Im Unterschied zu A1 wird der erbeutete Token nicht nur auf Anwendungsebene eingesetzt, sondern direkt zur Authentifizierung auf Netzwerkebene genutzt, um eine vollwertige Netzwerk-Session zu öffnen. Tabelle 3.3 zeigt die Schritte für beide Architekturvarianten.

Phase	Zero Trust	Perimeter
Vorbereitung	Ausgangszustand der Ziti-Objekte dokumentiert.	WireGuard-Schnittstellenstatus, AllowedIPs und erreichbare Netze dokumentiert.
Angriff	OIDC-Token verwendet, um direkt eine Ziti-API-Session über <code>ext-jwt</code> zu öffnen. Verfügbare Dienste auf Netzwerkebene enumeriert. Zugriff auf Dienste versucht.	Aus dem Netzkontext des kompromittierten Peers erreichbare Ziele und Subnetze dokumentiert.
Eindämmung	Containment über die Wazuh-OPA-Bridge ausgelöst: OPA-Blockliste aktualisiert und Ziti-Identität deaktiviert.	Kein aktiver Eingriff durchgeführt.
Wiederherstellung	Sperre aufgehoben. Ausgangszustand wiederhergestellt und verifiziert.	Ursprünglicher Zustand dokumentiert.

Tabelle 3.3.: Durchführungsschritte Szenario A2

3.3.2.3. Szenario A3: Exponierter und verwundbarer Cloud-Dienst

Szenario A3 setzt die in Abschnitt 1.4.2.3 definierte Kompromittierung eines öffentlich erreichbaren Cloud-Dienstes um. Im Zentrum steht, welche weiteren Systeme von diesem Brückenkopf aus erreichbar sind und wie gezielt die Architektur auf die Kompromittierung eines einzelnen Dienstes reagieren kann, ohne den übrigen Betrieb zu beeinträchtigen. Die zugehörigen Durchführungsschritte sind in Tabelle 3.4 dokumentiert.

Phase	Zero Trust	Perimeter
Vorbereitung	Bind- und Dial-Policies des Zielservers (intranet-host) dokumentiert. Ausgangszustand aller Host-Identitäten und Dienste erfasst.	Erreichbare Dienste und offene Ports auf VPS2 und OnPrem von VPS1 aus sondiert und dokumentiert.
Angriff	Pivot-Potenzial des kompromittierten Servers untersucht: verfügbare Dial-Policies der Host-Identität analysiert.	Lateral-Movement-Potenzial anhand erreichbarer Netzpfade und offener Ports von VPS1 aus dokumentiert.
Eindämmung	Host-Identität des kompromittierten Servers über die Wazuh-OPA-Bridge gesperrt. Erreichbarkeit aller Dienste anschliessend geprüft.	Kein aktiver Eingriff durchgeführt.
Wiederherstellung	Sperre aufgehoben. Ausgangszustand wiederhergestellt und verifiziert.	Ursprünglicher Zustand dokumentiert.

Tabelle 3.4.: Durchführungsschritte Szenario A3

3.3.2.4. Szenario A4: Insider-Threat durch externen Service-Techniker

Szenario A4 bildet den in Abschnitt 1.4.2.4 definierten Insider-Fall ab. Im Unterschied zu A1 und A2 handelt es sich nicht um gestohlene Zugangsdaten, sondern um den absichtlichen Missbrauch eines regulären Wartungszugangs. Tabelle 3.5 beschreibt die Durchführung.

Phase	Zero Trust	Perimeter
Vorbereitung	Ausgangszustand der Ziti-Objekte und Service-Policies für die Techniker-Rolle dokumentiert.	VPN-Peer und AllowedIPs des Technikers dokumentiert. Erreichbare Netze und Dienste erfasst.
Angriff	OIDC-Token für service-technician bezogen. Ziti-Session geöffnet. Sämtliche sichtbaren Dienste enumeriert. Zugriff auf Dienste ausserhalb der autorisierten Rolle versucht.	Aus dem Netzkontext des Techniker-Peers erreichbare Systeme und Dienste dokumentiert.
Eindämmung	Containment über die Wazuh-OPA-Bridge ausgelöst. Techniker-Identität gesperrt.	Kein aktiver Eingriff durchgeführt.
Wiederherstellung	Sperre aufgehoben. Ausgangszustand wiederhergestellt und verifiziert.	Ursprünglicher Zustand dokumentiert.

Tabelle 3.5.: Durchführungsschritte Szenario A4

3.3.3. Betriebsszenarien

Die Betriebsszenarien untersuchen typische Änderungen im laufenden Betrieb. Bewertet werden der notwendige Konfigurationsaufwand, die Nachvollziehbarkeit der Änderungen und die Frage, ob neue Komponenten unbeabsichtigte Auswirkungen auf bestehende Zugriffsbeziehungen haben.

3.3.3.1. Szenario B1: Hinzufügen eines neuen Clients

Szenario B1 setzt die Integration eines neuen Clients gemäss Abschnitt 1.4.3.1 um. Erfasst werden die pro Architektur notwendigen Schritte und Konfigurationsobjekte sowie mögliche Auswirkungen auf bestehende Zugriffe. Die Schritte sind in Tabelle 3.6 gegenübergestellt.

Phase	Zero Trust	Perimeter
Ausgangszustand	Anzahl vorhandener Identitäten, Services und Service-Policies dokumentiert.	Bestehende WireGuard-Peers dokumentiert.
Änderung	Neue Client-Identität erstellt und enrolled. Dial-Policy für genau einen Dienst angelegt.	Temporärer WireGuard-Peer mit zugewiesener IP hinzugefügt.
Verifikation	Sichtbare Dienste des neuen Clients sowie bestehender Identitäten geprüft.	Erreichbarkeit des Peers und bestehender Peers geprüft.
Wiederherstellung	Neue Objekte entfernt. Ausgangszustand verifiziert.	Peer entfernt. Ausgangszustand verifiziert.

Tabelle 3.6.: Durchführungsschritte Szenario B1

3.3.3.2. Szenario B2: Hinzufügen eines neuen Cloud-Dienstes

Szenario B2 setzt die Integration eines neuen Cloud-Dienstes gemäss Abschnitt 1.4.3.2 um. Dokumentiert wird, welche Konfigurationsobjekte und Schritte erforderlich sind, um den Dienst betriebsbereit und zugänglich zu machen, und ob dabei unbeabsichtigt bestehende Zugriffe beeinflusst werden. Tabelle 3.7 fasst die Vorgehensweise zusammen.

Phase	Zero Trust	Perimeter
Ausgangszustand	Anzahl vorhandener Configs, Services, Identitäten und Service-Policies dokumentiert.	Laufende Container und veröffentlichte Ports dokumentiert.
Änderung	Host-Config, Intercept-Config, Service, Client-Identität, Bind-Policy und Dial-Policy angelegt.	Temporärer Container gestartet und Port veröffentlicht.
Verifikation	Sichtbarkeit des neuen Dienstes für neue und bestehende Identitäten geprüft.	Erreichbarkeit des neuen Dienstes und Zustand bestehender Dienste geprüft.
Wiederherstellung	Alle neuen Objekte entfernt. Ausgangszustand verifiziert.	Container gestoppt, Port geschlossen. Ausgangszustand verifiziert.

Tabelle 3.7.: Durchführungsschritte Szenario B2

3.3.3.3. Szenario B3: Hinzufügen eines neuen OT-Sensors

Szenario B3 entspricht in seiner Struktur B2, betrachtet jedoch die in Abschnitt 1.4.3.3 definierte OT-nahe Änderung: die Integration eines neuen Sensors in die bestehende OT-Umgebung. Besonderes

Augenmerk liegt darauf, ob die Änderung die Sichtbarkeit des neuen Sensors auf bestehende Rollen ausweitet oder strikt begrenzt bleibt. Die Durchführung ist in Tabelle 3.8 beschrieben.

Phase	Zero Trust	Perimeter
Ausgangszustand	Anzahl vorhandener Configs, Services, Identitäten und Service-Policies dokumentiert.	Laufende Dienste auf OnPrem und veröffentlichte Ports dokumentiert.
Änderung	Host-Config, Intercept-Config, Service, Client-Identität, Bind-Policy und Dial-Policy angelegt. Service an bestehende OnPrem-Host-Identität gebunden.	Temporärer Container direkt auf OnPrem gestartet und Port lokal veröffentlicht.
Verifikation	Sichtbarkeit des neuen Sensors für neue und bestehende Identitäten (inkl. Techniker-Rolle) geprüft.	Lokale Erreichbarkeit des neuen Diensts sowie Zustand bestehender Dienste geprüft.
Wiederherstellung	Alle neuen Objekte entfernt. Ausgangszustand verifiziert.	Container gestoppt, Port geschlossen. Ausgangszustand verifiziert.

Tabelle 3.8.: Durchführungsschritte Szenario B3

3.4. Bewertung der Hypothesen

Dieser Abschnitt bewertet jede der zehn in Abschnitt 1.3 formulierten Hypothesen anhand der in Abschnitt 3.3 beschriebenen Szenariodurchführungen. Für jede Hypothese werden die Ergebnisse der Zero-Trust-Architektur und der perimeterbasierten Architektur getrennt dargestellt, anschliessend einschränkende Faktoren diskutiert und abschliessend ein begründetes Urteil gefällt. Eine Hypothese gilt als **bestätigt**, wenn die vorliegenden Messwerte die aufgestellte Erwartung in der Laborumgebung nachweislich stützen. Als **teilweise bestätigt** gilt eine Hypothese, wenn der Befund klar in die erwartete Richtung zeigt, einzelne Messdimensionen aber fehlen oder nicht quantitativ erfasst wurden. Als **nicht bestätigt** gilt eine Hypothese, wenn der erwartete Vorteil der Zero-Trust-Architektur gegenüber der perimeterbasierten Architektur in den erhobenen Daten nicht nachweisbar war. Tabelle 3.9 gibt vorab eine Übersicht über die Bewertungen.

Für alle Bewertungen gilt einschränkend, dass sich die Aussagen auf die konkret implementierten Architekturvarianten beziehen. Ergänzende Härtungsmassnahmen wie Mikrosegmentierung, Host-Firewalls, Network Access Control (NAC) oder zusätzliche Logging-Schichten waren nicht Bestandteil der untersuchten Perimeterimplementierung; auf diese Einschränkung wird bei den einzelnen Hypothesen nur dort erneut eingegangen, wo sie für die Einordnung wesentlich ist. Der Abschnitt adressiert Teilfrage 5.2, indem er die sicherheitstechnischen Vor- und Nachteile beider Architekturen im identischen Laborszenario gegenüberstellt.

H	Hypothese	Bewertung	Szenarien
H1	Verringerung der Angriffsfläche	bestätigt	A1, A2, A3, A4
H2	Schnellere Erkennung von Angriffsaktivitäten	nicht bestätigt	A1, A2, A3, A4
H3	Schnellere und gezieltere Eindämmung	bestätigt	A1, A2, A3, A4
H4	Höhere Nachvollziehbarkeit des Angriffspfads	bestätigt	A1, A2, A3, A4
H5	Keine unverhältnismässige Einschränkung des legitimen Betriebs	nicht bestätigt	A1, A2, A3, A4, B1, B2, B3
H6	Begrenzter Angriffserfolg nach Erstkompromittierung	bestätigt	A1, A2, A3, A4
H7	Schutz kritischer OT-Ressourcen	nicht bestätigt	A2, A4
H8	Höherer Einrichtungsaufwand	bestätigt	B1, B2, B3
H9	Höherer Betriebs- und Pflegeaufwand	teilweise bestätigt	B1, B2, B3
H10	Höhere Anpassungsfähigkeit trotz Komplexität	bestätigt	B1, B2, B3

Tabelle 3.9.: Übersicht der Hypothesenbewertungen

3.4.1. Sicherheitsbezogene Hypothesen

H1: Verringerung der Angriffsfläche

Zero Trust Die Szenarien A1 bis A4 zeigen aus unterschiedlichen Angreiferperspektiven, dass die Angriffsfläche in der untersuchten Zero-Trust-Architektur deutlich begrenzt wird. In den Szenarien A1 und A2 konnte die kompromittierte Identität `bob` ausschliesslich jene drei Dienste enumerieren und erreichen, für die explizite Dial-Policies konfiguriert waren: `intranet`, `ot-dashboard` und `uptime-kuma-direct`. Die übrigen acht Dienste, darunter `accounting`, `mqtt-broker`, `ot-dht11-modbus`, `guacamole-svc`, `win-rdp` sowie sämtliche SSH-Services, waren für diese Identität weder sichtbar noch erreichbar.

In Szenario A3 wurden die Bind- und Dial-Policies der kompromittierten Host-Identität `intranet-host` untersucht. Diese Identität verfügte lediglich über eine Bind-Policy für den Intranet-Dienst und über keine Dial-Policies. Dadurch war ausgehend von dieser Identität kein Zugriff auf andere Dienste innerhalb des Ziti-Fabrics möglich. Laterale Bewegungen wurden in der getesteten Konfiguration somit nicht durch Netzwerktopologie, sondern durch die fehlende Dienstberechtigung unterbunden.

In Szenario A4 konnte die legitim authentifizierte Identität `service-technician` exakt vier Dienste enumerieren: `guacamole-svc`, `mqtt-broker`, `ot-dashboard` und `ot-dht11-modbus`. IT-Dienste und

administrative Services waren für diese Identität nicht sichtbar. Akteure ohne Zugriff auf OpenZiti konnten die Dienste innerhalb des Ziti-Fabrics nicht enumerieren, sondern lediglich öffentlich erreichbare Endpunkte wie den OpenZiti-Controller, Wazuh oder Keycloak erkennen. Insgesamt ergibt sich die Begrenzung der Angriffsfläche in der Zero-Trust-Implementierung daher nicht aus einer Netzwerktopologie, sondern aus explizit konfigurierten, identitäts- und dienstbezogenen Ziti-Dial-Policies.

Perimeter Im perimeterbasierten Szenario A2 war das WireGuard-Interface wg0 mit dem Adressbereich 10.200.0.1/24 konfiguriert. Der Peer, der den kompromittierten Endpoint repräsentierte, verfügte über AllowedIPs für die Netze 192.168.0.0/24, 192.168.1.0/24 und 192.168.2.0/24. Damit erhielt der Peer Zugriff auf drei vollständige private Subnetze. In Szenario A4 galt derselbe subnetzbasierte Zugriffskontext für den Techniker-Peer.

Die potenziell erreichbare Angriffsfläche umfasst daher grundsätzlich alle Systeme und Dienste, die sich innerhalb der freigegebenen Subnetze befinden und deren Ports erreichbar sind. Die Zugriffskontrolle erfolgt in dieser Implementierung primär auf Netzebene und nicht auf Ebene einzelner Anwendungen oder Dienste.

Einschränkungen Der Vergleich ist auf die konkret untersuchten Implementierungen zu beziehen. In der perimeterbasierten Architektur hatte der gewählte Client Zugriff auf mehrere vollständige Subnetze. Wäre stattdessen ein stärker eingeschränkter Benutzer-Client betrachtet worden, hätte die erreichbare Angriffsfläche geringer ausfallen können, beispielsweise durch Beschränkung auf ein einzelnes Subnetz.

Zudem kann auch eine klassische Perimeterarchitektur durch ergänzende Massnahmen wie Host-Firewalls, NAC, dienstspezifische VPN-Profile, VLAN-Segmentierung oder Mikrosegmentierung granularer abgesichert werden. Diese Massnahmen waren jedoch nicht Bestandteil der untersuchten Perimeterimplementierung. Die Ergebnisse erlauben daher keine pauschale Aussage über alle möglichen Perimeterarchitekturen, sondern bewerten die getestete Konfiguration im Vergleich zur implementierten Zero-Trust-Architektur.

Fazit: bestätigt Die Hypothese H1 wird für die untersuchten Implementierungen bestätigt. In allen vier Angriffsszenarien war die erreichbare Angriffsfläche in der Zero-Trust-Architektur auf explizit freigegebene Dienste beschränkt. Nicht autorisierte Dienste waren für die jeweilige Identität weder sichtbar noch erreichbar.

Diese Begrenzung entspricht einer zentralen architektonischen Eigenschaft von Zero Trust. Durch Prinzipien wie *Least Privilege*, *deny by default*, explizite Verifikation sowie identitäts- und dienstbezogene Policy-Durchsetzung wird Zugriff nicht aufgrund der Position im Netzwerk gewährt, sondern muss für jede Identität und jeden Dienst explizit autorisiert werden. Dadurch wird die Angriffsfläche bereits durch die Grundlogik der Architektur reduziert.

Die untersuchte Perimeterarchitektur bot diese Eigenschaft in der getesteten Form nicht. Der Zugriff erfolgte primär subnetzbasiert, wodurch ein kompromittierter Peer potenziell alle erreichbaren Systeme und offenen Dienste innerhalb der freigegebenen Netze adressieren konnte. Eine vergleichbare Granularität müsste im Perimetermodell zusätzlich durch Massnahmen wie Host-Firewalls, NAC, dienstspezifische VPN-Profile oder Mikrosegmentierung konfiguriert werden. Im direkten Vergleich zeigt die Zero-Trust-Implementierung daher eine systematischere Begrenzung der Angriffsfläche. H1 ist somit als bestätigt zu bewerten.

H2: Schnellere Erkennung von Angriffsaktivitäten

Zero Trust In den manuellen Testläufen zeigte sich eine stark variierende *Time to Detect*. Diese Schwankungen liessen sich nicht auf Unterschiede zwischen der Zero-Trust-Architektur und der perimeterbasierten Architektur zurückführen. Stattdessen hing die Erkennungszeit im Wesentlichen davon ab, zu welchem Zeitpunkt Windows Defender den jeweiligen Vorfall erkannte und an Wazuh weiterleitete. Da bereits in der manuellen Prüfung eine hohe Streuung der Messwerte festgestellt wurde, hätte eine weiterführende quantitative Auswertung keine belastbare Aussage über den Einfluss der Netzwerkarchitektur ermöglicht. Auf eine weiterführende quantitative Auswertung der *Time to Detect* wurde deshalb verzichtet; die Bewertung der Hypothese stützt sich auf die Beobachtungen der manuellen Testläufe.

Der einzige relevante Unterschied zwischen den beiden untersuchten Umgebungen bestand darin, dass in der Zero-Trust-Umgebung ein SIEM aktiv war. Dieses konnte jedoch nur dann einen Alert verarbeiten, wenn zuvor eine Meldung durch Windows Defender erzeugt wurde. Die eigentliche initiale Erkennung des Angriffseignisses lag somit nicht beim Netzwerkmodell oder beim SIEM, sondern beim vorgeschalteten Virenschutz. Die Erkennungszeit wurde folglich primär durch die Detektionslogik von Windows Defender bestimmt.

Perimeter In der perimeterbasierten Umgebung war kein SIEM aktiv. Daraus lässt sich jedoch nicht unmittelbar ableiten, dass die Erkennungszeit in dieser Architektur schlechter war. Windows Defender übernahm in beiden Umgebungen dieselbe Funktion als initiale Erkennungskomponente. Da dessen Detektionsverhalten unabhängig vom zugrunde liegenden Netzwerkmodell ist, konnte in den Testläufen kein belastbarer architekturbedingter Unterschied in der tatsächlichen Erkennungszeit festgestellt werden.

Einschränkungen Windows Defender ist kein Bestandteil der untersuchten Netzwerkarchitekturen und lag ausserhalb des direkten Einflussbereichs dieser Arbeit. Seine Erkennungsleistung kann unter anderem durch Signaturstand, Scanzeitpunkt, Systemzustand oder Dateiposition beeinflusst werden. Eine Auswertung der *Time to Detect* hätte daher primär die Leistungsfähigkeit des eingesetzten Virenschutzes bewertet und nicht die Zero-Trust- beziehungsweise Perimeterarchitektur selbst. Die Aussagekraft der Messung im Hinblick auf H2 ist daher eingeschränkt.

Fazit: nicht bestätigt Die Hypothese H2 konnte für die untersuchten Implementierungen nicht bestätigt werden. Die Testläufe zeigten keinen belastbaren Hinweis darauf, dass die Zero-Trust-Architektur Angriffsaktivitäten schneller erkannte als die perimeterbasierte Architektur. Die gemessene *Time to Detect* wurde hauptsächlich durch Windows Defender bestimmt, der in beiden Umgebungen identisch eingesetzt wurde und nicht Bestandteil der untersuchten Netzwerkarchitektur war.

Zwar verfügte die Zero-Trust-Umgebung zusätzlich über ein SIEM, dieses beeinflusste jedoch nicht die initiale Erkennung des Vorfalls, sondern verarbeitete lediglich bereits erzeugte Defender-Meldungen weiter. Damit konnte kein architekturbedingter Vorteil der Zero-Trust-Implementierung hinsichtlich der Erkennungsgeschwindigkeit nachgewiesen werden. H2 ist daher als nicht bestätigt zu bewerten.

H3: Schnellere und gezieltere Eindämmung

Zero Trust In allen vier Angriffsszenarien wurde die Eindämmung automatisiert über die Wazuh-OPA-Bridge ausgelöst. Die gemessene *Time to Contain* (TTC) betrug in jedem Lauf rund eine Sekunde (vgl. Anhang C). Gemessen wurde jeweils die Zeit zwischen dem Eingang der Meldung bei Wazuh und der wirksamen Sperrung der betroffenen Identität. Die Messung beschreibt damit nicht die gesamte Zeit vom Angriffsbeginn bis zur Eindämmung, sondern den Containment-Prozess nach erfolgter Alarmierung.

Die Eindämmung erfolgte in allen Fällen identitätsgenau. In den Szenarien A1, A2 und A4 wurde ausschliesslich die jeweils betroffene Benutzeridentität gesperrt; Server-Blocklisten blieben unverändert. In Szenario A3 wurde ausschliesslich die kompromittierte Host-Identität `intranet-host` deaktiviert. Die übrigen sieben Host-Identitäten blieben aktiv. Andere Identitäten, Dienste oder Kommunikationsbeziehungen wurden zu keinem Zeitpunkt beeinträchtigt. Damit zeigte die Zero-Trust-Implementierung eine schnelle und zugleich gezielte Eindämmung mit minimalen Kollateraleffekten.

Perimeter In den perimeterbasierten Läufen wurde kein aktives Containment umgesetzt. Grundsätzlich wären auch in einer Perimeterarchitektur verschiedene Eindämmungsmassnahmen denkbar, beispielsweise das Entfernen eines WireGuard-Peers, das Anpassen von `AllowedIPs`, Änderungen an Firewall-Regeln oder die Sperrung eines Anwendungskontos im Identity Provider (IdP). Diese Massnahmen wirken jedoch typischerweise auf Peer-, Netz-, Port- oder Anwendungsebene und nicht unmittelbar auf eine einheitliche, architekturweit durchgesetzte Zugriffsidentität.

Eine Sperre, die exakt eine kompromittierte Identität betrifft und gleichzeitig alle anderen Identitäten, Dienste und Kommunikationsbeziehungen unverändert lässt, war in der untersuchten Perimeterkonfiguration nicht implementiert. Eine vergleichbare Präzision hätte zusätzliche Integrationen und Automatisierungen erfordert, etwa durch die Kopplung von SIEM, IdP, NAC, Firewall-Regeln oder Endpoint-Security-Systemen.

Einschränkungen Da in der perimeterbasierten Umgebung kein aktives Containment durchgeführt wurde, liegen für diese Architektur weder gemessene TTC-Werte noch empirisch erhobene Kollateraleffekte vor. Die Bewertung der Perimeter-Eindämmung ist daher qualitativ und architekturbasiert. Es ist möglich, dass eine stärker integrierte Perimeterumgebung mit automatisierter IdP-Kontosperre, NAC-Quarantäne oder Firewall-Orchestrierung eine ähnliche Geschwindigkeit und Präzision erreichen könnte. Solche Mechanismen waren jedoch nicht Bestandteil der untersuchten Implementierung und wurden daher nicht getestet.

Fazit: bestätigt Die Hypothese H3 wird für die untersuchten Implementierungen bestätigt. Die Zero-Trust-Architektur ermöglichte in allen vier Szenarien eine konsistent schnelle Eindämmung von rund einer Sekunde ab Eingang der Meldung bei Wazuh. Zugleich erfolgte die Sperrung gezielt auf Ebene der betroffenen Benutzer- oder Host-Identität, ohne dass andere Identitäten, Dienste oder Kommunikationsbeziehungen beeinträchtigt wurden.

Diese Fähigkeit entspricht einer zentralen Stärke identitätsbasierter Zero-Trust-Architekturen. Da Zugriffe nicht primär über Netzpositionen oder Subnetze, sondern über explizite Identitäten und Policies gesteuert werden, kann auch die Eindämmung unmittelbar an dieser Identität ansetzen. Prinzipien wie *Least Privilege*, zentrale Policy-Durchsetzung, identitätsbezogene Autorisierung

und *deny by default* unterstützen damit nicht nur die Zugriffskontrolle, sondern auch eine präzise Reaktion auf kompromittierte Identitäten.

Die untersuchte Perimeterarchitektur verfügte über keine äquivalente, beobachtete Containment-Mechanik. Eine Eindämmung wäre zwar auch dort möglich, müsste jedoch über zusätzliche Konfigurationen und Integrationen auf Peer-, Netz-, Firewall-, NAC- oder Anwendungsebene umgesetzt werden. Im direkten Vergleich zeigte die Zero-Trust-Implementierung daher eine schnellere und gezieltere Eindämmung mit geringeren Kollateraleffekten. H3 ist somit als bestätigt zu bewerten.

H4: Höhere Nachvollziehbarkeit des Angriffspfads

Zero Trust Die Zero-Trust-Artefakte dokumentieren für die untersuchten Angriffsszenarien eine durchgängige Ereigniskette innerhalb eines gemeinsamen Identitätskontexts. Dazu gehören der Authentifizierungszeitpunkt und der HTTP-Status im Keycloak-Log, die sichtbaren Dienste aus der Ziti-Session-Enumeration, die wirksamen Zugriffsberechtigungen anhand der Dial-Policy-Grants, der Eindämmungszeitpunkt und die durchgeführte Massnahme aus der OPA-Bridge-Antwort, der Identitätsstatus nach dem Containment sowie die anschliessende Wiederherstellung mit leerem Blocklisten-Status. Dadurch lassen sich die zentralen Schritte von der Authentifizierung über die Dienstsichtbarkeit bis zur Eindämmung und Wiederherstellung nachvollziehen.

In Szenario A3 ist zusätzlich dokumentiert, dass die Host-Identität `intranet-host` über keine Dial-Policies verfügt. Damit ist aus den Artefakten ersichtlich, dass diese Identität in der untersuchten Konfiguration keine Berechtigung für ausgehende Zugriffe auf andere Dienste hatte. Ein lateraler Zugriffspfad über diese Identität war somit anhand der Policy-Daten nicht ableitbar.

Perimeter Die perimeterbasierten Artefakte bestehen aus WireGuard-Interface-Snapshots, Routing-Tabellen, Docker-Zuständen, Portproben und lokalen Hostzuständen. Diese Artefakte beschreiben primär Netzwerk- und Hostzustände zu bestimmten Zeitpunkten. Sie enthalten jedoch keine direkte Verknüpfung zwischen einer Benutzeridentität, einem aufgerufenen Dienst, einer zugrundeliegenden Policy-Entscheidung und einer daraus resultierenden Eindämmungsmassnahme.

Eine zusammenhängende Rekonstruktion des Angriffspfads von der Authentifizierung über die Dienstnutzung bis zur Wiederherstellung ist aus diesen Artefakten allein nicht möglich. Insbesondere fehlt ein einheitlicher Identitätsbezug, der einzelne Netzwerk- und Hostereignisse konsistent derselben handelnden Identität zuordnet.

Einschränkungen Auch perimeterbasierte Umgebungen können eine hohe Nachvollziehbarkeit erreichen, wenn zusätzliche Logging- und Analysekomponenten eingesetzt werden. Dazu zählen beispielsweise ein zentrales SIEM, VPN-Logs, Proxy-Logs, DNS-Logs, Firewall-Logs, Daten aus Endpoint-Detection-and-Response-Systemen (EDR) oder zentralisierte Authentifizierungsprotokolle. Solche zusätzlichen Logging-Schichten waren jedoch nicht Bestandteil der untersuchten Perimeterimplementierung. Die Bewertung bezieht sich daher auf die nativ erzeugten Artefakte der beiden getesteten Architekturen.

Zudem ist zu berücksichtigen, dass die Nachvollziehbarkeit in beiden Architekturmodellen von der Qualität, Vollständigkeit und Korrelation der erhobenen Logs abhängt. Der identitätszentrierte Ansatz kann jedoch die Abhängigkeit von leicht veränderbaren Netzwerkmerkmalen wie IP- oder

MAC-Adressen reduzieren. Dies kann die Zuordnung von Ereignissen robuster machen als rein netzwerkbasierte Artefakte. Dieser Aspekt wurde in der vorliegenden Evaluation jedoch nicht gesondert getestet. Die Zero-Trust-Implementierung bietet in der getesteten Form dennoch den Vorteil, dass wesentliche Ereignisse bereits über Identitäten und Policies miteinander verknüpft sind.

Fazit: bestätigt Die Hypothese H4 wird für die untersuchten Implementierungen bestätigt. Die Zero-Trust-Architektur erzeugte in den getesteten Szenarien einen durchgängigen Audit-Trail, der Authentifizierung, Dienstsichtbarkeit, Policy-Wirkung, Eindämmung und Wiederherstellung in einem gemeinsamen Identitätskontext abbildet. Dadurch konnte der Angriffspfad deutlich konsistenter nachvollzogen werden als in der perimeterbasierten Umgebung.

Die Perimeterimplementierung erzeugte dagegen vor allem fragmentierte Netzwerk- und Hostartefakte ohne direkten Bezug zu einer konkreten Benutzer- oder Dienstidentität. Eine vergleichbare Rekonstruierbarkeit müsste dort durch zusätzliche Logging-, Korrelations- und Analysemechanismen hergestellt werden. Der identitätszentrierte Ansatz der Zero-Trust-Architektur bietet hierbei einen strukturellen Vorteil, da Ereignisse nicht primär über IP-Adressen, Subnetze oder Ports, sondern über explizite Identitäten und Policy-Entscheidungen zugeordnet werden können. H4 ist daher als bestätigt zu bewerten.

H5: Keine unverhältnismässige Einschränkung des legitimen Betriebs

Zero Trust In der Zero-Trust-Architektur erfolgte die Eindämmung gezielt durch Isolation betroffener Identitäten oder Dienste. In Szenario A3 war nach der Deaktivierung der Host-Identität *intranet-host* von elf geprüften Diensten nur ein Dienst nicht erreichbar; die übrigen zehn Dienste blieben verfügbar. In A1, A2 und A4 wurden ausschliesslich Benutzeridentitäten gesperrt, sodass die Dienste selbst für andere berechnigte Identitäten weiterhin erreichbar blieben.

Perimeter Auch in einer perimeterbasierten Architektur kann eine vergleichbare Isolation erreicht werden, indem ein betroffener Dienst heruntergefahren, ein Port geschlossen oder der Zugriff über Firewall-Regeln blockiert wird. Die technische Umsetzung unterscheidet sich zwar von der Policy-basierten Isolation in Zero Trust, die Wirkung aus Nutzersicht kann jedoch gleich sein: Der betroffene Dienst ist nicht mehr erreichbar, während andere Dienste weiterhin verfügbar bleiben können.

Einschränkungen Die Evaluation zeigt eine präzise technische Eindämmung in der Zero-Trust-Architektur. Sie belegt jedoch nicht, dass die daraus resultierende Betriebsunterbrechung aus Nutzersicht geringer ist als bei einer einfachen Dienstabstaltung oder Blockierung in einer perimeterbasierten Umgebung.

Fazit: nicht bestätigt Die Hypothese H5 wird nicht bestätigt. Zero Trust ermöglicht eine gezielte Eindämmung durch Policy-basierte Isolation. Eine vergleichbare Isolation kann jedoch auch in einer perimeterbasierten Architektur durch das Herunterfahren oder Blockieren des betroffenen Dienstes erreicht werden. Aus Nutzersicht ist die Einschränkung in beiden Fällen ähnlich, da der betroffene Dienst nicht verfügbar ist. Ein eindeutiger Vorteil von Zero Trust hinsichtlich der

Vermeidung unverhältnismässiger Betriebsunterbrechungen konnte daher nicht nachgewiesen werden.

H6: Begrenzter Angriffserfolg nach Erstkompromittierung

Zero Trust In allen vier Angriffsszenarien war der maximale Angriffserfolg nach der Erstkompromittierung durch die konfigurierten Dial-Policies begrenzt. In den Szenarien A1 und A2 konnte die kompromittierte Identität `bob` nach erfolgreicher Authentifizierung ausschliesslich drei Dienste erreichen. Kritische Dienste wie `accounting`, `mqtt-broker` sowie sämtliche Administrationszugänge lagen ausserhalb der zugewiesenen Dial-Policy-Grants und waren damit für diese Identität nicht erreichbar.

In Szenario A3 verfügte die kompromittierte Host-Identität `intranet-host` über keine Dial-Policies. Ein Angreifer mit Kontrolle über diesen Server konnte über das Ziti-Fabric daher keine weiteren Dienste ansteuern. Laterale Bewegungen über die untersuchte Zero-Trust-Kommunikationsschicht waren dadurch in dieser Konfiguration nicht möglich. In Szenario A4 blieb die Identität `service-technician` auf vier autorisierte Wartungsdienste beschränkt. IT- und Admin-Dienste waren in seiner Session nicht sichtbar und nicht erreichbar.

Perimeter Im perimeterbasierten Szenario A2 verfügte der kompromittierte Peer über `AllowedIPs` für drei vollständige /24-Subnetze. Alle in diesen Subnetzen betriebenen Dienste mit offenen Ports wären für einen Angreifer in diesem Netzkontext potenziell erreichbar gewesen. Eine Kompromittierung eines erreichbaren Dienstes hätte den Angriffserfolg zusätzlich erhöhen und weitere laterale Bewegungen innerhalb der jeweiligen Subnetze ermöglichen können.

In Szenario A4 galt ein vergleichbarer subnetzbasierter Zugriffskontext für den Techniker-Peer. Eine direkte, architekturweite Begrenzung auf einzelne autorisierte Anwendungsdienste war in der untersuchten Perimeterkonfiguration nicht umgesetzt. Der mögliche Angriffserfolg nach einer Erstkompromittierung hing somit stärker von der Netzsegmentierung, den offenen Ports und den Schutzmechanismen der einzelnen Zielsysteme ab.

Einschränkungen Auch perimeterbasierte Architekturen können laterale Bewegungen und den Angriffserfolg nach einer Erstkompromittierung einschränken, beispielsweise durch Netzsegmentierung, Mikrosegmentierung, Host-Firewalls, Jump-Host-Konzepte oder zusätzliche Zugriffskontrollen auf Anwendungsebene. Diese Massnahmen waren jedoch nicht Bestandteil der untersuchten Perimeterimplementierung. Die freigegebenen Subnetze definieren daher lediglich eine potenzielle Obergrenze der Erreichbarkeit, nicht zwingend den tatsächlich realisierten Angriffserfolg. Ebenso ist nicht davon auszugehen, dass jeder erreichbare Dienst automatisch kompromittiert werden kann. Entscheidend ist jedoch, dass die untersuchte Perimeterkonfiguration eine solche Weiterbewegung nicht bereits auf Ebene einzelner autorisierter Dienste strukturell begrenzte. Einschränkend ist zudem zu berücksichtigen, dass TCP-Prüfungen von VPS1 beziehungsweise VPS2 in Richtung On-Premises-Netz in der Laborumgebung in Timeouts liefen; die Aussagen zur lateralen Erreichbarkeit beschreiben daher den konkret beobachteten Netzpfad (vgl. Abschnitt 4.2).

Fazit: bestätigt Die Hypothese H6 wird für die untersuchten Implementierungen bestätigt. In der Zero-Trust-Architektur blieb der Angriffserfolg nach einer Erstkompromittierung auf jene Dienste beschränkt, die der jeweiligen Identität explizit durch Policies zugewiesen waren. Dadurch konnten

kompromittierte Benutzer- oder Host-Identitäten nicht automatisch auf weitere Netzbereiche oder nicht autorisierte Dienste zugreifen.

Diese Begrenzung entspricht einer zentralen architektonischen Eigenschaft von Zero Trust. Durch Prinzipien wie *Least Privilege*, *deny by default*, identitätsbasierte Autorisierung und dienstbezogene Policy-Durchsetzung wird laterale Bewegung nicht primär durch Netzgrenzen, sondern durch explizite Zugriffsentscheidungen kontrolliert. Zero Trust unterstützt damit bereits durch seine Grundlogik eine Begrenzung des möglichen Schadens nach einer Erstkompromittierung.

Die untersuchte Perimeterarchitektur bot diese Eigenschaft in der getesteten Form nicht. Ein kompromittierter Peer erhielt Zugriff auf vollständige Subnetze. Dadurch waren alle dort vorhandenen Systeme und offenen Dienste potenzielle Angriffsziele. Wäre ein solcher erreichbarer Dienst zusätzlich kompromittiert worden, hätte dies den Angriffserfolg erhöhen und weitere laterale Bewegungen innerhalb der freigegebenen Netzbereiche ermöglichen können. Eine vergleichbare Begrenzung lateraler Bewegungen wäre auch in perimeterbasierten Netzen möglich, müsste jedoch durch zusätzliche Segmentierungs-, Firewall-, NAC- oder Anwendungskontrollen umgesetzt werden.

Im direkten Vergleich zeigte die Zero-Trust-Implementierung daher eine stärkere Begrenzung des Angriffserfolgs nach Erstkompromittierung. Während der Angriffserfolg im Zero-Trust-Modell durch explizite Identitäts- und Dienst-Policies begrenzt wurde, war er im untersuchten Perimetermodell stärker von der Erreichbarkeit ganzer Subnetze und der Absicherung einzelner Zielsysteme abhängig. H6 ist somit als bestätigt zu bewerten.

H7: Schutz kritischer OT-Ressourcen

Zero Trust Szenario A2 zeigt, dass die kompromittierte Identität `bob` trotz geöffneter Ziti-Netzwerksession keine OT-Dienste enumerieren konnte. Weder `mqtt-broker` noch `ot-dht11-modbus` erschienen in der Session-Enumeration. Der Zugriff auf OT-Ressourcen war somit für diese Identität nicht möglich, da keine entsprechenden Dial-Policies vorhanden waren.

Szenario A4 zeigt dagegen den Zugriff einer autorisierten Wartungsidentität. Die Identität `service-technician` konnte exakt jene vier OT- und Wartungsdienste enumerieren, die für diese Rolle vorgesehen waren: `guacamole-svc`, `mqtt-broker`, `ot-dashboard` und `ot-dht11-modbus`. IT- und Administrationsdienste waren für diese Identität nicht sichtbar. Die Zero-Trust-Implementierung setzte die Rollengrenze damit auch für eine legitim authentifizierte Identität dienstgenau durch.

Ein Restrisiko bleibt jedoch innerhalb der freigegebenen OT-Dienste bestehen. Zero Trust begrenzt den Netzwerkzugang zu diesen Diensten, ersetzt aber nicht die fachliche Zugriffskontrolle, Härtung und Überwachung innerhalb der jeweiligen Anwendung oder OT-Komponente.

Perimeter In der perimeterbasierten Variante wurde der Zugriff auf OT-Ressourcen durch eine klassische Netzsegmentierung umgesetzt. Die OT-Geräte befanden sich in einem eigenen Subnetz und waren nicht allgemein aus allen Netzbereichen erreichbar. Der Zugriff erfolgte über einen dafür vorgesehenen Kontext, insbesondere über den Jump-Host beziehungsweise über Peers mit entsprechendem Netz- oder Routingzugriff. Dadurch wurde bereits mit einer vergleichsweise einfachen Perimeterisierung eine Schutzwirkung für die OT-Geräte erzielt.

Gleichzeitig unterscheidet sich die Schutzlogik vom Zero-Trust-Modell. In der untersuchten Perimeterkonfiguration erfolgte die Begrenzung primär über Netzbereiche, Routing und erreichbare Subnetze, nicht über eine identitäts- und dienstgenaue Policy-Matrix. Ein Peer mit Zugriff auf das OT-Subnetz kann potenziell alle dort erreichbaren Systeme und offenen Dienste adressieren, sofern keine weiteren Firewalls, Host-Regeln oder Anwendungskontrollen greifen. Der Schutz ist damit stärker von der Qualität der Netzsegmentierung und der Absicherung der Übergänge abhängig.

Einschränkungen Der Schutz kritischer OT-Ressourcen lässt sich nicht allein anhand des zugrunde liegenden Architekturmodells bewerten. Sowohl Zero Trust als auch perimeterbasierte Architekturen können OT-Komponenten wirksam schützen, sofern sie konsequent umgesetzt werden. Zero Trust erreicht dies vor allem durch identitäts- und dienstbezogene Policies, *Least Privilege* und *deny by default*. Perimeterbasierte OT-Architekturen können dagegen durch dedizierte OT-Zonen, Firewalls, Jump-Hosts, Allowlisting, Monitoring und streng kontrollierte Übergänge zwischen IT- und OT-Netzen ein hohes Schutzniveau erreichen.

Die untersuchte Implementierung zeigt daher zwar unterschiedliche Schutzmechanismen, erlaubt aber keine allgemeine Aussage, dass eines der beiden Architekturmodelle kritische OT-Ressourcen grundsätzlich besser schützt. Entscheidend ist vielmehr, wie konsequent die jeweilige Architektur umgesetzt, segmentiert, überwacht und betrieben wird.

Fazit: nicht bestätigt Die Hypothese H7 wird für die untersuchten Implementierungen nicht bestätigt. Die Zero-Trust-Architektur begrenzte den Zugriff auf OT-Ressourcen zwar sehr präzise über identitäts- und dienstbezogene Policies. Nicht autorisierte Identitäten konnten OT-Dienste nicht enumerieren oder erreichen, während die autorisierte Wartungsidentität ausschliesslich die vorgesehenen OT- und Wartungsdienste nutzen konnte.

Gleichzeitig zeigte auch die perimeterbasierte Architektur eine Schutzwirkung, indem OT-Geräte in einem eigenen Netzbereich betrieben und nur über definierte Zugriffswege, insbesondere den Jump-Host beziehungsweise freigegebene Netzkontexte, erreichbar waren. Damit kann nicht grundsätzlich behauptet werden, dass Zero Trust im Schutz kritischer OT-Ressourcen per se überlegen ist. Vielmehr bieten beide Ansätze Schutzmechanismen mit unterschiedlicher Granularität: Zero Trust schützt primär identitäts- und dienstbezogen, während das Perimetermodell über Netzsegmentierung, kontrollierte Übergänge und abgeschottete OT-Zonen schützt. H7 ist daher als nicht bestätigt zu bewerten.

3.4.2. Betriebsbezogene Hypothesen

H8: Höherer Einrichtungsaufwand bei Zero Trust

Zero Trust Die drei Betriebsszenarien zeigen, dass Änderungen in der Zero-Trust-Architektur jeweils mehrere explizite Konfigurationsobjekte erfordern. In B1, dem Hinzufügen eines neuen Clients, wurden eine neue Identität (Gesamtanzahl: 23→24) sowie eine Dial-Policy (24→25) angelegt. Die Laufzeit betrug rund 2 Sekunden (vgl. Anhang C).

In B2, dem Hinzufügen eines neuen Cloud-Dienstes, und B3, dem Hinzufügen eines neuen OT-Sensors, wurden jeweils zwei Configs (24→26), ein Service (11→12), eine Identität (23→24), eine Bind-Policy sowie eine Dial-Policy (24→26) erstellt. Die Laufzeit betrug in beiden Fällen rund 3 Sekunden. Damit ist jede Änderung explizit über Identitäten, Services, Konfigurationen

und Policies modelliert. Die erzeugten Objekte sind über die Controller-API einsehbar und durch Baseline-, Verify- und Restore-Snapshots nachvollziehbar.

Perimeter In der perimeterbasierten Architektur war der unmittelbare Konfigurationsumfang in den untersuchten Betriebsszenarien geringer. In B1 wurde ein einzelner WireGuard-Peer mit einer IP-Adresse hinzugefügt. Die Laufzeit betrug rund 1 Sekunde. In B2 wurde ein Container gestartet und ein Port (18085:80) veröffentlicht; die Laufzeit betrug rund 3 Sekunden. In B3 wurde ein Container direkt On-Premises gestartet und ein Port (18086:80) lokal veröffentlicht; die Laufzeit betrug rund 5 Sekunden.

In keinem der drei perimeterbasierten Läufe war ein zusätzliches, explizites Service-Policy-Objekt erforderlich. Der Zugriff wurde primär durch Peer-Konfiguration, Netzerreichbarkeit und Portfreigaben bestimmt. Dadurch fiel die Anzahl der neu zu modellierenden Sicherheitsobjekte geringer aus als in der Zero-Trust-Implementierung.

Einschränkungen Die reine Laufzeit der Automatisierungsskripte ist nur eingeschränkt aussagekräftig, da beide Architekturen automatisiert konfiguriert wurden und manuelle Planungs- oder Konfigurationszeiten nicht gemessen wurden. Für die Bewertung des Einrichtungsaufwands ist daher weniger die technische Ausführungsdauer entscheidend, sondern vor allem die Anzahl und Abhängigkeit der zu modellierenden Sicherheitsobjekte.

Zero Trust erfordert insbesondere bei neuen Diensten ein explizites Zugriffsmodell aus Identität, Service, Host-Config, Intercept-Config, Bind-Policy und Dial-Policy. Dieser Modellierungsaufwand bleibt auch bei Automatisierung bestehen, da die erforderlichen Beziehungen fachlich definiert und korrekt umgesetzt werden müssen. Gleichzeitig ist dieser Mehraufwand nicht nur als Nachteil zu bewerten, sondern bildet die Grundlage für die zuvor beobachtete dienst- und identitätsgenaue Zugriffskontrolle.

Fazit: bestätigt Die Hypothese H8 wird für die untersuchten Implementierungen bestätigt. In allen drei Betriebsszenarien erforderte die Zero-Trust-Architektur mehr explizite Konfigurationsobjekte als die perimeterbasierte Architektur. Besonders bei der Bereitstellung neuer Dienste mussten neben der eigentlichen Identität und dem Service auch zugehörige Configs sowie Bind- und Dial-Policies erstellt werden.

Der höhere Einrichtungsaufwand ergibt sich damit aus der Grundlogik der Zero-Trust-Architektur: Zugriffe werden nicht implizit über Netzzugehörigkeit oder Portfreigaben ermöglicht, sondern müssen explizit über Identitäten, Dienste und Policies modelliert werden. Dieser zusätzliche Modellierungsaufwand ist der strukturelle Preis für die granularere Zugriffskontrolle, bessere Nachvollziehbarkeit und gezieltere Eindämmung. H8 ist daher als bestätigt zu bewerten.

H9: Höherer Betriebs- und Pflegeaufwand

Zero Trust Die Baseline-/Verify-/Restore-Snapshots aller drei Betriebsszenarien zeigen, dass temporäre Objekte explizit gezählt, zugeordnet und entfernt werden können. Nach dem Restore stimmten die Objektzahlen in allen Läufen exakt mit der Baseline überein: 23 Identitäten, 24 Service-Policies, 11 Services und 24 Configs. Dies zeigt, dass der Objektbestand im Zero-Trust-Controller konsistent nachvollzogen und wiederhergestellt werden konnte.

Gleichzeitig bedeutet der höhere Objektbestand, dass diese Objekte kontinuierlich gepflegt werden müssen. Jede Identität, jede Policy, jeder Service und jede Config stellt ein eigenes Verwaltungsobjekt dar, das aktuell gehalten, überwacht, dokumentiert und bei Bedarf entfernt werden muss. Der Pflegeaufwand ergibt sich damit insbesondere aus der kontinuierlichen Verwaltung der Beziehungen zwischen Identitäten, Diensten und Policies.

Perimeter Die perimeterbasierten Läufe erforderten pro Änderung weniger explizite Konfigurationsobjekte. Der Pflegeaufwand wird dadurch jedoch nicht vollständig eliminiert, sondern teilweise in andere Bereiche verlagert. Dazu zählen Peer-Zuordnungen, AllowedIPs-Bereiche, Portfreigaben, Routing-Einträge und Firewall-Regeln, die ebenfalls konsistent dokumentiert, überprüft und gepflegt werden müssen.

Im Unterschied zu Zero Trust sind diese Elemente in der untersuchten Perimeterumgebung weniger direkt einer konkreten Identität oder einem einzelnen Dienst zugeordnet. Dadurch kann der Pflegeaufwand weniger sichtbar sein und sich stärker über verschiedene technische Ebenen verteilen. Die geringere Anzahl expliziter Objekte bedeutet daher nicht zwangsläufig einen geringeren Gesamtbetriebsaufwand.

Einschränkungen Der Betriebs- und Pflegeaufwand lässt sich nur eingeschränkt objektiv und quantitativ messen, da er nicht allein von der Anzahl technischer Objekte abhängt. Relevant sind zusätzlich Faktoren wie Dokumentationsqualität, Automatisierungsgrad, organisatorische Prozesse, Änderungsfrequenz sowie die Erfahrung der administrierenden Personen. Die vorliegende Evaluation versteht sich daher als Annäherung an den Pflegeaufwand anhand beobachtbarer Konfigurationsobjekte und ihrer Beziehungen.

Der Vergleich bewertet insbesondere, wie sichtbar, explizit und eindeutig die jeweiligen Konfigurationselemente in den untersuchten Architekturen modelliert sind. Während Zero Trust viele Verwaltungsobjekte zentral und identitätsbezogen abbildet, verteilt sich der Pflegeaufwand in der Perimeterarchitektur stärker auf Netzwerk-, Routing-, Peer- und Firewall-Konfigurationen. Dadurch bleibt der Gesamtaufwand beider Ansätze nur eingeschränkt vergleichbar.

Fazit: teilweise bestätigt Die Hypothese H9 wird für die untersuchten Implementierungen teilweise bestätigt. Zero Trust erhöht den sichtbaren und explizit verwalteten Objektbestand gegenüber dem Perimetermodell. Dieser Mehraufwand ist anhand der Identitäten, Services, Configs sowie Bind- und Dial-Policies nachvollziehbar und in den Snapshot-Artefakten messbar.

Gleichzeitig ist der Perimeteraufwand nicht zwingend geringer, sondern weniger sichtbar. Er liegt unter anderem in Peer-Verwaltung, AllowedIPs, Firewall-Regeln, Routing-Einträgen, Portfreigaben und Dokumentation. H9 ist daher teilweise bestätigt: Der explizite Pflegeaufwand steigt in der Zero-Trust-Architektur, während der Gesamtaufwand beider Architekturen auf Basis der vorliegenden Daten nur eingeschränkt vergleichbar ist.

H10: Höhere Anpassungsfähigkeit trotz Komplexität

Zero Trust In allen drei Betriebsszenarien wurden Änderungen exakt auf die adressierte Identität und den adressierten Dienst begrenzt. In B1 sah der neue Client `b1-maint-client` ausschliesslich `intranet`, nicht mehr und nicht weniger. In B2 sah `b2-cloud-client` ausschliesslich

b2-cloud-service. In B3 sah b3-ot-sensor-client ausschliesslich b3-ot-sensor. Die bestehenden Identitäten bob (3 Dienste) und service-technician (4 Dienste) blieben in keinem der drei Läufe verändert. Insbesondere wurde in B3 die bestehende OT-Wartungsrolle des Technikers nicht automatisch auf den neuen Sensor ausgeweitet.

Perimeter In der perimeterbasierten Variante von B1 wurde ein neuer Peer mit IP-Kontext hinzugefügt, ohne automatische Einschränkung auf einen einzelnen Dienst. In B2 war der neue Dienst nach der Portfreigabe hostseitig erreichbar. Eine Einschränkung auf eine bestimmte Client-Identität wurde durch den Perimetermechanismus nicht erzwungen. In B3 war der neue OT-Dienst lokal auf OnPrem erreichbar. Die Steuerung, wer diesen Dienst nutzen darf, obliegt ausschliesslich dem Netzkontext und etwaigen Firewall-Regeln.

Einschränkungen Auch perimeterbasierte Architekturen können präzise Zugriffskontrolle implementieren, etwa durch eng definierte Firewall-Regeln, dedizierte VPN-Tunnel pro Dienst oder NAC-Mechanismen. Diese Kontrollen müssen jedoch zusätzlich und manuell aufgebaut werden. Der Perimetermechanismus selbst erzwingt sie nicht strukturell. Zero Trust macht die Einschränkung zum Standard: Ohne explizite Dial-Policy ist kein Zugriff möglich.

Fazit: bestätigt Zero Trust ermöglicht präzise, identitäts- und dienstgenaue Anpassungen, ohne bestehende Zugriffsbeziehungen unbeabsichtigt zu erweitern. Dieses Verhalten folgt strukturell aus dem deny-by-default-Modell, solange das Policy-Modell korrekt gepflegt wird, und ist nicht das Ergebnis sorgfältiger manueller Zusatzkonfiguration. Die Perimeterumgebung bietet diese strukturelle Eigenschaft nicht. H10 ist somit als bestätigt zu bewerten.

3.4.3. Gesamtbetrachtung

Über alle zehn Hypothesen hinweg ergibt sich ein konsistentes Gesamtbild: Die Vorteile der untersuchten Zero-Trust-Architektur konzentrieren sich auf jene Bewertungsdimensionen, in denen die identitäts- und dienstbezogene Policy-Durchsetzung unmittelbar wirksam wird. Die Begrenzung der Angriffsfläche (H1), die gezielte Eindämmung (H3), die Nachvollziehbarkeit des Angriffspfads (H4), die Begrenzung des Angriffserfolgs nach Erstkompromittierung (H6) und die präzise Anpassbarkeit (H10) wurden in allen zugehörigen Szenarien bestätigt. In sämtlichen Fällen liegt derselbe Mechanismus zugrunde: Zugriffe, Sichtbarkeit, Sperrungen und Änderungen setzen an expliziten Identitäten, Diensten und Policies an und nicht an Netzpositionen, Subnetzen oder Ports. Dieser Befund war über alle vier Angreiferperspektiven der Szenarien A1 bis A4 hinweg stabil, unabhängig davon, ob gestohlene Zugangsdaten, ein kompromittierter Endpoint, ein kompromittierter Diensthost oder der Missbrauch eines legitimen Wartungszugangs simuliert wurde.

Ein zweites wiederkehrendes Muster betrifft die nicht bestätigten Hypothesen. H2, H5 und H7 scheiterten nicht daran, dass die Zero-Trust-Implementierung die erwartete Eigenschaft nicht aufwies, sondern daran, dass der erwartete *Vorsprung* gegenüber dem Perimetermodell nicht nachweisbar war. Bei H2 wurde die Erkennungszeit massgeblich durch Windows Defender bestimmt, eine Komponente, die in beiden Umgebungen identisch eingesetzt wurde und nicht Bestandteil der Netzwerkarchitektur ist. Bei H5 und H7 konnte die perimeterbasierte Architektur mit einfacheren Mitteln, etwa Dienstabschaltung, Portblockierung oder Netzsegmentierung mit dediziertem OT-Subnetz und Jump-Host, eine aus Nutzersicht beziehungsweise aus Schutzsicht vergleichbare Wirkung erzielen. Die Ergebnisse zeigen damit, dass die Stärken von Zero Trust dort liegen, wo

Granularität, Identitätsbezug und strukturelle Garantien entscheidend sind, und nicht zwingend dort, wo der gewünschte Effekt auch mit größeren Mechanismen erreichbar ist.

Die betriebsbezogenen Hypothesen bestätigen den erwarteten Trade-off. Der höhere Einrichtungsaufwand (H8) ist anhand der zusätzlich zu modellierenden Identitäten, Services, Configs sowie Bind- und Dial-Policies klar messbar. Der laufende Pflegeaufwand (H9) steigt in der Zero-Trust-Architektur sichtbar an, wobei der Vergleich dadurch erschwert wird, dass der Perimeteraufwand weniger explizit ist und sich über Peer-, Routing-, Port- und Firewall-Konfigurationen verteilt. Gleichzeitig zeigen die Szenarien B1 bis B3, dass derselbe Objektbestand, der den Mehraufwand verursacht, die in den Angriffsszenarien beobachteten Sicherheitseigenschaften erst ermöglicht: Jede Änderung blieb exakt auf die adressierte Identität und den adressierten Dienst begrenzt, bestehende Zugriffsbeziehungen wurden in keinem Lauf unbeabsichtigt erweitert, und der Restore stellte den Ausgangszustand in allen Läufen exakt wieder her. Der Mehraufwand und die Granularität sind somit zwei Seiten derselben architektonischen Eigenschaft.

Auffällig ist zudem, dass die bestätigten Sicherheitsvorteile in der Zero-Trust-Implementierung strukturell verankert sind, während sie im Perimetermodell grundsätzlich nachrüstbar wären. In den Einschränkungen zu H1, H3, H4, H6 und H10 wurde jeweils festgehalten, dass eine vergleichbare Granularität auch perimeterbasiert erreichbar ist, etwa durch Mikrosegmentierung, Host-Firewalls, NAC, dienstspezifische VPN-Profilen oder zusätzliche Logging- und Orchestrierungsschichten. Diese Massnahmen müssen jedoch zusätzlich konfiguriert und konsistent gepflegt werden, während Zero Trust die restriktive Grundhaltung über *deny by default* und *Least Privilege* zum Standardverhalten macht. Der Unterschied liegt damit weniger im theoretisch Erreichbaren als in der Frage, ob die Begrenzung das Ergebnis expliziter Zusatzmassnahmen oder eine Grundeigenschaft der Architektur ist.

Die Aussagekraft dieser Gesamtbetrachtung unterliegt den bereits in den Einzelbewertungen genannten Einschränkungen. Die Ergebnisse beziehen sich auf die konkret implementierten Architekturvarianten in der Laborumgebung. Insbesondere wurde die perimeterbasierte Architektur ohne ergänzende Härtungsmassnahmen betrieben, und für sie liegen keine gemessenen Containment-Werte vor. Die *Time to Contain* beschreibt zudem nur den Prozess nach erfolgter Alarmierung, nicht die Gesamtzeit ab Angriffsbeginn. Eine pauschale Überlegenheit von Zero Trust gegenüber allen denkbaren Perimeterarchitekturen lässt sich aus den Daten daher nicht ableiten. Innerhalb dieses Rahmens zeigen die Ergebnisse jedoch ein klares Bild: Die untersuchte Zero-Trust-Implementierung bietet systematische, reproduzierbar nachgewiesene Vorteile bei Angriffsflächenbegrenzung, Eindämmung, Nachvollziehbarkeit und kontrollierter Anpassbarkeit, erkaufte diese aber mit einem höheren expliziten Modellierungs- und Pflegeaufwand und ohne nachweisbaren Vorteil bei der initialen Angriffserkennung.

4. Fazit

Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen, beantwortet die Forschungsfrage und ordnet die Aussagekraft der Evaluation kritisch ein.

4.1. Beantwortung der Forschungsfrage

Die zentrale Forschungsfrage dieser Arbeit lautet (vgl. Abschnitt 1.2):

Welche Vorteile bietet eine Zero-Trust-Architektur in einer heterogenen Netzwerkumgebung mit IT- und OT-Geräten im Vergleich zu einer perimeterbasierten Sicherheitsarchitektur?

Auf Grundlage der in Abschnitt 3.3 durchgeführten Szenarien und der Hypothesenbewertung in Abschnitt 3.4 lässt sich diese Frage für die untersuchten Implementierungen wie folgt beantworten: Die Zero-Trust-Architektur bietet gegenüber der perimeterbasierten Architektur vier empirisch nachgewiesene sicherheitsbezogene Vorteile, die alle auf demselben architektonischen Prinzip beruhen, nämlich der identitäts- und dienstbezogenen Zugriffskontrolle anstelle eines impliziten Netzwerkvertrauens. Die Zusammenführung der Ergebnisse greift die zuvor beantworteten Teilfragen 1.1 bis 5.2 auf und verdichtet sie zur Beantwortung der zentralen Forschungsfrage.

Erstens begrenzt Zero Trust die Angriffsfläche systematisch (H1). In allen vier Angriffsszenarien waren für kompromittierte oder missbräuchlich verwendete Identitäten ausschliesslich die explizit per Dial-Policy freigegebenen Dienste sichtbar und erreichbar, während der kompromittierte WireGuard-Peer der Perimeterarchitektur Zugriff auf drei vollständige Subnetze erhielt. Dieser Vorteil beruht unmittelbar auf den Prinzipien *deny by default* und *Least Privilege*: Da kein Zugriff implizit gewährt wird, sondern jede Kommunikationsbeziehung explizit autorisiert werden muss, reduziert sich die erreichbare Angriffsfläche bereits durch die Grundhaltung der Architektur.

Zweitens bleibt der Angriffserfolg nach einer Erstkompromittierung begrenzt (H6): Laterale Bewegungen wurden nicht durch Netzgrenzen, sondern durch fehlende Dienstberechtigungen unterbunden, im Extremfall des Szenarios A3 vollständig, da die kompromittierte Host-Identität über keine Dial-Policies verfügte. Hier wirken *Least Privilege* und die dienstbezogene Policy-Durchsetzung im Sinne einer Mikrosegmentierung auf Dienstebene: Selbst eine erfolgreich kompromittierte Identität kann sich nur innerhalb der ihr minimal zugewiesenen Berechtigungen bewegen.

Drittens ermöglicht die Architektur eine schnelle und gezielte Eindämmung (H3): Die automatisierte Sperrung der betroffenen Benutzer- oder Host-Identität erfolgte in allen Läufen innerhalb von rund einer Sekunde nach Eingang der Meldung und ohne Beeinträchtigung anderer Identitäten, Dienste oder Kommunikationsbeziehungen. Grundlage dafür sind die identitätsbasierte Zugriffskontrolle und die zentrale Policy-Durchsetzung in Verbindung mit der kontinuierlichen Verifikation: Weil jeder Zugriff fortlaufend gegen die zentrale Policy-Entscheidung geprüft wird, entzieht die Deaktivierung einer einzelnen Identität dieser unmittelbar und architekturweit den Zugriff, ohne dass Netze, Ports oder Dienste pauschal gesperrt werden müssen.

Viertens erzeugt die Architektur einen durchgängigen, identitätsbezogenen Audit-Trail von der Authentifizierung über die Dienstsichtbarkeit bis zur Eindämmung und Wiederherstellung (H4), während die Perimeterumgebung nur fragmentierte Netzwerk- und Hostartefakte ohne einheitlichen Identitätsbezug lieferte. Dieser Vorteil folgt aus der expliziten Verifikation jedes Zugriffs (*Never Trust, Always Verify*): Da jede Zugriffsentscheidung an eine verifizierte Identität und eine konkrete Policy gebunden ist, entstehen die für die Rekonstruktion notwendigen Artefakte bereits durch den regulären Betrieb der Architektur.

Für die heterogene IT-/OT-Umgebung ist darüber hinaus als betrieblicher Vorteil die in den Betriebsszenarien bestätigte Anpassungsfähigkeit (H10) relevant: Neue Clients, Cloud-Dienste und OT-Sensoren konnten integriert werden, ohne dass bestehende Rollen, etwa die OT-Wartungsrolle des Service-Technikers, unbeabsichtigt erweitert wurden. Auch dieses Verhalten ist eine direkte Folge von *deny by default*: Ohne explizite Dial-Policy ist kein Zugriff möglich, weshalb neue Komponenten bestehende Zugriffsbeziehungen strukturell nicht verändern können. Gerade in Umgebungen, in denen IT-Dienste, OT-Komponenten und externe Wartungszugänge mit unterschiedlichen Schutzbedarfen koexistieren, erlaubt dies eine präzisere Abbildung von Rollen- und Dienstgrenzen, als sie über reine Netzzugehörigkeit möglich ist.

Diese Vorteile gelten jedoch nicht uneingeschränkt. Kein nachweisbarer Vorteil ergab sich bei der Erkennungsgeschwindigkeit (H2), da die initiale Erkennung in beiden Umgebungen durch den architekturunabhängigen Endpoint-Schutz bestimmt wurde, sowie bei der Vermeidung unverhältnismässiger Einschränkungen des legitimen Betriebs durch Eindämmung (H5) und beim Schutz kritischer OT-Ressourcen (H7), wo die perimeterbasierte Architektur mit Netzsegmentierung und Jump-Host eine vergleichbare, wenn auch gröbere Schutzwirkung erzielte. Den nachgewiesenen Vorteilen steht zudem ein bestätigter Mehraufwand gegenüber: Die Einrichtung erfordert mehr explizite Konfigurationsobjekte (H8), und der sichtbare Pflegeaufwand im Betrieb steigt (H9), auch wenn der Perimeteraufwand teilweise lediglich weniger sichtbar in Peer-, Routing- und Firewall-Konfigurationen verlagert ist. Wie in Abschnitt 3.4.3 dargelegt, ist dieser Mehraufwand der strukturelle Preis für die beobachtete Granularität.

Die Antwort ist an die Bedingungen der Evaluation gebunden. Sie beruht auf einer kontrollierten Laborumgebung und auf zwei konkreten Implementierungen: einer Zero-Trust-Architektur auf Basis von Keycloak, OpenZiti, OPA und Wazuh sowie einer perimeterbasierten Architektur auf Basis von WireGuard ohne ergänzende Härtungsmassnahmen wie Mikrosegmentierung, NAC oder zusätzliche Logging-Schichten. Eine stärker ausgebaute Perimeterarchitektur könnte einzelne der beobachteten Unterschiede verringern, müsste die entsprechenden Kontrollen jedoch durch zusätzliche Massnahmen herstellen, die Zero Trust strukturell über *deny by default* und *Least Privilege* erzwingt. Die weiteren Einschränkungen werden in Abschnitt 4.2 diskutiert.

Zusammenfassend lässt sich die Forschungsfrage dahingehend beantworten, dass die Vorteile einer Zero-Trust-Architektur in einer heterogenen IT-/OT-Umgebung primär in der systematischen Begrenzung der Angriffsfläche, der Eindämmung kompromittierter Identitäten mit minimalen Kollateraleffekten, der durchgängigen Nachvollziehbarkeit von Angriffspfaden sowie der kontrollierten, rollengenauen Anpassbarkeit liegen. Diese Vorteile sind nicht das Ergebnis einzelner Zusatzwerkzeuge, sondern folgen aus der Grundlogik der Architektur und wurden in der Evaluation reproduzierbar nachgewiesen. Sie sind gegen einen höheren expliziten Modellierungs- und Pflegeaufwand abzuwägen und ersetzen weder Endpoint-Schutz noch sichere Anwendungen. Damit ist das Ziel der Arbeit erreicht, die Vorteile einer Zero-Trust-Architektur gegenüber einer perimeterbasierten Sicherheitsarchitektur in einem identischen Laborszenario systematisch zu untersuchen.

4.2. Limitationen

Dieser Abschnitt beschreibt die Einschränkungen der Laborumgebung, der Messmethoden und der Übertragbarkeit der Ergebnisse auf produktive IT/OT-Umgebungen.

Die Ergebnisse beruhen auf einer kontrollierten Laborumgebung und sind deshalb nicht direkt mit einer produktiven Unternehmens- oder Produktionsumgebung gleichzusetzen. Die Umgebung enthält ausgewählte IT- und OT-nahe Dienste, Benutzerrollen, Raspberry-Pi-Komponenten und verteilte Serverstandorte, bildet aber keine vollständige industrielle Anlage ab. Aspekte wie Hochverfügbarkeit, grössere Benutzerzahlen, komplexe Produktionsprozesse, Schichtbetrieb, Legacy-Steuerungen oder umfangreiche Netzwerksegmentierung wurden nicht vollständig nachgebildet.

Die Angriffsszenarien wurden kontrolliert und nicht destruktiv durchgeführt. A1 simulierte Credential Theft mit bekannten Labordaten, A2 verwendete ein kontrolliert erzeugtes OIDC-Token, A3 simulierte die Kompromittierung eines Diensthhosts über Policy-Inspektion und Blockierung, und A4 prüfte den Missbrauch legitimer Zugangsdaten. Es wurden keine realen Phishing-Kampagnen, Malware-Infektionen, Exploits oder destruktiven Manipulationen gegen OT-Komponenten durchgeführt. Dadurch sind die Ergebnisse reproduzierbarer und ethisch vertretbar, decken aber nicht das vollständige Verhalten realer Angriffe ab.

Eine methodische Annahme betrifft die eingesetzten Komponenten selbst: Es wird davon ausgegangen, dass die verwendeten Open-Source-Komponenten keine Sicherheitslücken aufweisen (vgl. Abschnitt 2.2). Bewertet wurden die Auswirkungen ihrer Integration in die Gesamtarchitektur, nicht die Sicherheit der einzelnen Produkte.

Eine weitere Einschränkung betrifft die Erkennung. Die beiden Umgebungen waren in diesem Punkt nicht vollständig symmetrisch, da nur die Zero-Trust-Umgebung über ein SIEM verfügte. Die Zero-Trust-Läufe dokumentieren Keycloak-, OpenZiti-, OPA-, Wazuh-OPA-Bridge- und Log-Artefakte, aber keine produktive automatische Time to Detect über ein real ausgelöstes Alarmereignis. Die Time-to-Contain-Werte sind daher belastbarer als die Aussagen zur automatischen Erkennungszeit. Die Hypothese H2 wurde entsprechend als nicht bestätigt bewertet (vgl. Abschnitt 3.4).

Für die perimeterbasierte Architektur wurden keine destruktiven Firewall- oder Routingänderungen vorgenommen. Die Eindämmung wurde deshalb vor allem anhand vorhandener Architektur- und Laufzeitartefakte eingeordnet. Praktische Massnahmen wie das Entfernen eines WireGuard-Peers, das Schliessen eines Ports oder das Stoppen eines Containers wurden nur dort durchgeführt, wo sie als temporäre Wartungsszenarien vorgesehen waren. Kollateraleffekte produktiver Sperrmassnahmen in einer realen Perimeterumgebung wurden nicht vollständig getestet.

Die On-Premises-Erreichbarkeit war in den perimeterbasierten Läufen nur eingeschränkt messbar. Lokale On-Premises-Dienste konnten auf dem On-Premises-Server bestätigt werden, TCP-Prüfungen von VPS1 beziehungsweise VPS2 auf 192.168.2.201 liefen jedoch in Timeouts. Die Ergebnisse zur lateralen Bewegung in Richtung On-Premises-Netz beschreiben daher den konkret beobachteten Netzpfad der Laborumgebung und nicht zwingend alle möglichen Perimeterkonfigurationen.

Schliesslich ist die Reproduzierbarkeit durch externe und physische Faktoren begrenzt. DNS-Einträge, VPS-Systeme, Secrets, Zertifikate, OpenZiti-Identitätsdateien, Raspberry Pis, Sensorik, LED-Matrix und Verkabelung lassen sich nicht vollständig aus dem Repository erzeugen. Die Infrastructure-as-Code-Artefakte beschreiben die logische Architektur und grosse Teile des Deployments, ersetzen aber nicht die reale Bereitstellung der Infrastruktur und Hardware.

4.3. Ausblick

Erstens ist eine naheliegende Erweiterung die Vergrößerung der OT-Umgebung. Die Arbeit betrachtet mit DHT11/Modbus, MQTT, OT-Dashboard und Lüfter-Simulation ausgewählte OT-nahe Kommunikationsbeziehungen. Weiterführende Untersuchungen könnten zusätzliche Protokolle wie OPC UA, Profinet, EtherNet/IP oder herstellerspezifische Schnittstellen einbeziehen. Dadurch liesse sich prüfen, ob die beobachteten Vorteile der dienstbezogenen Zugriffskontrolle auch bei komplexeren industriellen Kommunikationsmustern bestehen bleiben.

Zweitens sollte die automatische Erkennung produktionsnäher untersucht werden. Die aktuelle Evaluation dokumentiert die Eindämmung und die relevanten Artefakte, misst aber keine vollständige automatische Time to Detect. Eine Folgestudie könnte reale oder realitätsnähere Wazuh-Regeln, Endpoint-Agenten, Netzwerkereignisse und Active-Response-Ketten einsetzen, um Erkennungszeit, Fehlalarme und operative Belastung belastbarer zu bewerten.

Drittens wäre eine weitergehende dynamische Policy-Anpassung sinnvoll. In der Laborumgebung kann die Wazuh-OPA-Bridge Benutzer oder Server blockieren und damit OPA- beziehungsweise OpenZiti-Zustände beeinflussen. Für nicht-HTTP-Dienste wie MQTT oder Modbus wären zusätzliche Mechanismen interessant, beispielsweise das automatisierte Entfernen von Dial-Attributen, temporäre Quarantäne-Rollen oder kontextabhängige OpenZiti-Service-Policies.

Viertens sollte die perimeterbasierte Vergleichsarchitektur in zukünftigen Arbeiten breiter variiert werden. Der vorliegende Vergleich nutzt eine WireGuard-/VPN-Architektur mit veröffentlichten Diensten. Ergänzende Varianten mit dedizierter Firewall-Appliance, VLANs, DMZ-Segmenten oder industrieller Zonierung nach Purdue-Modell könnten zeigen, wie stark die Ergebnisse von der konkreten Ausprägung des Perimeters abhängen.

Schliesslich wäre eine stärkere Automatisierung der Evaluation hilfreich. Die bestehenden Skripte erzeugen bereits strukturierte Artefakte pro Szenario. Darauf aufbauend könnten Auswertungen, Tabellen, Diagramme und Regressionsprüfungen automatisch generiert werden. Dies würde die Vergleichbarkeit mehrerer Läufe verbessern und die Arbeit von einer einmaligen Laborevaluation in Richtung kontinuierlich überprüfbarer Sicherheitsmessungen weiterentwickeln.

Literatur

- [1] Md. Faizul Bari u. a. «On Orchestrating Virtual Network Functions». In: *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, S. 50–56. doi: [10.1109/CNSM.2015.7367338](https://doi.org/10.1109/CNSM.2015.7367338).
- [2] Cisco. *Security Outcomes for Zero Trust*. 2024. URL: <https://www.cisco.com/c/en/us/products/security/zero-trust-outcomes-report.html> (besucht am 09. 03. 2026).
- [3] Envoy Project Authors. *What is Envoy*. 2026. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy (besucht am 20. 04. 2026).
- [4] ETSI. *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action*. Introductory White Paper Issue 1. European Telecommunications Standards Institute, Okt. 2012. URL: https://portal.etsi.org/nfv/nfv_white_paper.pdf (besucht am 03. 06. 2026).
- [5] European Union Agency for Cybersecurity (ENISA). *ENISA Threat Landscape 2023*. 2023. URL: <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023> (besucht am 20. 03. 2026).
- [6] Federal Statistical Office (FSO). *Working from home – 2017–2023*. Data based on the Swiss Labour Force Survey. Swiss Federal Statistical Office. 2024. URL: <https://www.bfs.admin.ch/asset/en/33748242> (besucht am 09. 03. 2026).
- [7] Fabio Federici, Davide Martintoni und Valerio Senni. «A Zero-Trust Architecture for Remote Access in Industrial IoT Infrastructures». In: *Electronics* 12.3 (2023), S. 566. doi: [10.3390/electronics12030566](https://doi.org/10.3390/electronics12030566).
- [8] Keycloak Authors. *Keycloak Documentation*. 2026. URL: <https://www.keycloak.org/documentation> (besucht am 20. 04. 2026).
- [9] Mandiant. *M-Trends 2023*. 2023. URL: <https://www.mandiant.com/resources/m-trends-2023> (besucht am 20. 03. 2026).
- [10] Modbus Organization. *About Us*. Die Modbus Organization bezeichnet sich als „steward of the world’s most widely adopted industrial communication protocol“. URL: <https://www.modbus.org/about-us> (besucht am 10. 06. 2026).
- [11] MQTT.org. *MQTT: The Standard for IoT Messaging*. Beschreibt MQTT als OASIS-Standard für IoT-Messaging und nennt den Einsatz in verschiedenen Industrien. URL: <https://mqtt.org/> (besucht am 10. 06. 2026).
- [12] National Cyber Security Centre. *Network Architectures*. 2021. URL: <https://www.ncsc.gov.uk/collection/device-security-guidance/infrastructure/network-architectures> (besucht am 20. 03. 2026).
- [13] National Institute of Standards and Technology. *Zero Trust Architecture (NIST Special Publication 800-207)*. Techn. Ber. NIST, 2020. URL: <https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf> (besucht am 09. 03. 2026).
- [14] Open Policy Agent Contributors. *Open Policy Agent Documentation*. 2026. URL: <https://www.openpolicyagent.org/docs/latest/> (besucht am 20. 04. 2026).
- [15] OpenZiti. *OpenZiti Documentation*. 2026. URL: <https://openziti.io/docs/> (besucht am 20. 04. 2026).
- [16] Palo Alto Networks. *What Is Zero Trust Architecture?* 2026. URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture> (besucht am 09. 03. 2026).

- [17] PwC. *Global Digital Trust Insights 2025*. PricewaterhouseCoopers (PwC). 2025. URL: <https://www.pwc.ch/en/insights/cybersecurity/global-digital-trust-2025.html> (besucht am 09. 03. 2026).
- [18] Rajiv Raich, Anagha Raich und Nandkishor Kinhekar. «Securing the Convergence of IT and OT Networks in Cyber Physical System: Policy, Architecture and Implementation Challenges». In: *ITM Web of Conferences* 79 (2025), S. 01005. doi: [10.1051/itmconf/20257901005](https://doi.org/10.1051/itmconf/20257901005).
- [19] Karen Scarfone und Paul Hoffman. *Guidelines on Firewalls and Firewall Policy*. NIST Special Publication 800-41 Revision 1. National Institute of Standards und Technology, 2009. doi: [10.6028/NIST.SP.800-41r1](https://doi.org/10.6028/NIST.SP.800-41r1).
- [20] Murugiah Souppaya und Karen Scarfone. *Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security*. NIST Special Publication 800-46 Revision 2. National Institute of Standards und Technology, 2016. doi: [10.6028/NIST.SP.800-46r2](https://doi.org/10.6028/NIST.SP.800-46r2).
- [21] Keith Stouffer u. a. *Guide to Industrial Control Systems (ICS) Security*. NIST Special Publication 800-82 Revision 2. National Institute of Standards und Technology, 2015. doi: [10.6028/NIST.SP.800-82r2](https://doi.org/10.6028/NIST.SP.800-82r2).
- [22] Swisscom AG. *Internet of Things: Wie Schweizer Unternehmen IoT einsetzen*. Swisscom. 1. Okt. 2025. URL: <https://www.swisscom.ch/de/business/enterprise/themen/iot/internet-of-things-in-schweizer-unternehmen.html> (besucht am 09. 03. 2026).
- [23] Verizon. *2023 Data Breach Investigations Report*. 2023. URL: <https://www.verizon.com/business/resources/reports/dbir/> (besucht am 20. 03. 2026).
- [24] Wazuh Inc. *Wazuh Documentation*. 2026. URL: <https://documentation.wazuh.com/current/index.html> (besucht am 20. 04. 2026).
- [25] Claudio Zanasi u. a. «A Zero Trust approach for the cybersecurity of Industrial Control Systems». In: *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*. IEEE, 2022. doi: [10.1109/NCA57778.2022.10013559](https://doi.org/10.1109/NCA57778.2022.10013559).
- [26] Tianzhu Zhang u. a. «NFV Platforms: Taxonomy, Design Choices and Future Challenges». In: *IEEE Transactions on Network and Service Management* 18.1 (2021), S. 30–48. doi: [10.1109/TNSM.2020.3045381](https://doi.org/10.1109/TNSM.2020.3045381).

Glossar

Credential Theft Diebstahl von Zugangsdaten, beispielsweise durch Phishing oder Auslesen lokaler Anmeldedaten. 8

DMZ Demilitarisierte Zone; separates Netzwerksegment, in dem öffentlich erreichbare Dienste von internen Kernsystemen getrennt betrieben werden. 14

Endpoint Endgerät oder Client-System eines Benutzers, beispielsweise ein Arbeitsplatzrechner, Laptop oder technisches Bediengerät. 8

Firewall Sicherheitskomponente zur Kontrolle von Netzwerkverkehr zwischen Netzen oder Hosts anhand definierter Regeln. 13

Infrastructure as Code Ansatz, bei dem Infrastruktur, Dienste und Konfigurationen über versionierte Dateien und Skripte beschrieben werden, um Aufbau und Änderungen reproduzierbar zu machen. 42

Insider Threat Bedrohung durch eine Person mit legitimen Zugriffsrechten, die diese Rechte missbräuchlich oder ausserhalb des vorgesehenen Aufgabenbereichs nutzt. 9

IoT Internet of Things; vernetzte Geräte, Sensoren oder Systeme, die Daten erfassen, austauschen oder Aktionen auslösen können. 40

IT Information Technology; Systeme und Dienste zur Verarbeitung, Speicherung und Bereitstellung digitaler Informationen, beispielsweise Webanwendungen, Benutzerverwaltung oder Monitoring. 22

Laterale Bewegung Ausbreitung eines Angreifers von einem bereits kompromittierten System oder Zugang zu weiteren Systemen innerhalb einer Umgebung. 12

Least Privilege Prinzip, nach dem Benutzer, Systeme und Dienste nur die minimal notwendigen Berechtigungen für ihre jeweilige Aufgabe erhalten. 12

Mikrosegmentierung Feingranulare Unterteilung von Kommunikationsbeziehungen, sodass Zugriffe nicht pauschal für ganze Netze, sondern gezielt für einzelne Ressourcen oder Dienste erlaubt werden. 12

Modbus TCP Industrielles Kommunikationsprotokoll zur Übertragung von Registerdaten über TCP/IP, in der Laborumgebung zur Bereitstellung von Sensordaten verwendet. 39

MQTT Leichtgewichtiges Publish-/Subscribe-Protokoll, das in der Laborumgebung zur Übertragung von Sensorzuständen und Steuerbefehlen verwendet wird. 40

Netzwerkperimeter Logische oder technische Grenze zwischen einem kontrollierten internen Netzwerk und externen Netzen wie dem Internet. 13

- OAuth 2.0** Autorisierungsframework, mit dem Anwendungen kontrollierten Zugriff auf Ressourcen erhalten können, ohne Passwörter direkt weiterzugeben. 20
- On-Premises** Betrieb von Systemen auf lokaler, selbst kontrollierter Infrastruktur statt ausschliesslich in externen Cloud- oder VPS-Umgebungen. 35
- OpenID Connect** Identitätsprotokoll auf Basis von OAuth 2.0, das Anwendungen die Authentifizierung von Benutzern über einen Identity Provider ermöglicht. 20
- OT** Operational Technology; Systeme zur Überwachung oder Steuerung physischer Prozesse, Maschinen, Sensoren oder Aktoren. 22
- Perimeterbasierte Sicherheitsarchitektur** Sicherheitsarchitektur, die den Schutz primär an der Grenze zwischen internem und externem Netzwerk umsetzt, beispielsweise mit Firewalls, VPN-Gateways und veröffentlichten Diensten. 13
- Policy** Formalisierte Regel, die beschreibt, unter welchen Bedingungen ein Zugriff erlaubt oder verweigert wird. 26
- Policy Administrator** Teil des Policy Decision Point, der Entscheidungen der Policy Engine in konkrete Anweisungen für die durchsetzende Komponente überführt. 12
- Policy Decision Point** Logische Komponente im Zero-Trust-Modell, die entscheidet, ob ein angefragter Zugriff gemäss Richtlinien und Kontextinformationen erlaubt wird. 12
- Policy Enforcement Point** Komponente, welche die Entscheidung über einen Zugriff technisch durchsetzt und Verbindungen erlaubt oder blockiert. 12
- Policy Engine** Teil des Policy Decision Point, der eine Zugriffsanfrage anhand von Richtlinien und verfügbaren Kontextdaten bewertet. 12
- Policy-Matrix** Zentrale, tabellarisch beziehungsweise strukturiert gepflegte Beschreibung der Rollen, Identitäten, Dienste und erlaubten Kommunikationsbeziehungen der Laborumgebung. 22
- Remote Access** Fernzugriff auf interne Systeme oder Dienste von ausserhalb des lokalen Netzwerks, beispielsweise über VPN oder identitätsbasierte Zugriffsdienste. 14
- SIEM** Security Information and Event Management; Systemklasse zur Sammlung, Korrelation und Auswertung sicherheitsrelevanter Ereignisse. 21
- Time to Contain** Zeitspanne zwischen der Einleitung einer Eindämmungsmassnahme und der bestätigten Begrenzung eines Angriffs oder Missbrauchs. 58
- Time to Detect** Zeitspanne zwischen dem Beginn oder Auftreten eines sicherheitsrelevanten Ereignisses und dessen Erkennung beziehungsweise Alarmierung. 47
- VLAN** Virtual Local Area Network; logische Segmentierung eines physischen Netzwerks in getrennte Broadcast-Domänen. 38
- VPN** Virtual Private Network; Technik, mit der entfernte Benutzer oder Standorte über einen geschützten Tunnel auf Netzwerkressourcen zugreifen können. 14
- VPS** Virtual Private Server; virtuell bereitgestellter Server, der in der Laborumgebung als Cloud-beziehungsweise externer Serverstandort verwendet wird. 35

Zero Trust Sicherheitsmodell, bei dem Zugriffe nicht aufgrund einer internen Netzwerkposition vertraut werden. Jede Kommunikation muss explizit authentifiziert, autorisiert und anhand definierter Richtlinien geprüft werden. 11

Hilfsmittelverzeichnis

Aufgabenbereich	Tools
Literatur-Recherche und Verwaltung	Google, Google Scholar, ChatGPT Deep Research, Google NotebookLM (Verwaltung der Dokumente)
Datenanalyse und Visualisierung	Python, pandas, matplotlib
Ideengenerierung	ChatGPT, Anthropic Claude
Text-Bearbeitung (Python, Bash, Docker etc.)	JetBrains IntelliJ, ChatGPT, Anthropic Claude
Texterstellung, Textoptimierung und Rechtschreibprüfung	DeepL, ChatGPT, Anthropic Claude, Language-Tool
Dokumenterstellung	LaTeX
Zusammenarbeit und Projektmanagement	Microsoft Teams, Threema, GitLab
DevOps	GitLab, GitLab CI

A. Projektplan

A.1. Projektmanagement

Für die Durchführung des Projekts wurde der Rational Unified Process (RUP) als Projektmanagementmethode gewählt. Dieses Vorgehen gilt als geeignet, da es eine iterative und inkrementelle Struktur bietet, ohne dabei viel Overhead zu verursachen. Der Prozess bietet klare Phasen mit überprüfbaren Meilensteinen, die eine systematische Vorgehensweise ermöglichen. Die Meilensteine werden in zweiwöchigen Abständen definiert. Diese gelten, in Anlehnung an Scrum, als Sprints. Auf diese Weise lässt sich der Forschungsprozess in überschaubare Abschnitte unterteilen, der Fortschritt regelmässig überprüfen und Erkenntnisse in den Projektverlauf einfließen lassen.

Die Projektkommunikation erfolgte über verschiedene Kanäle, darunter wöchentliche Meetings über Microsoft Teams, Kurztextnachrichten über Threema für schnelle Abstimmungen sowie Planung der Meetings, sowie Vor-Ort-Meetings im CN-Lab für die praktische Zusammenarbeit an der Laborinfrastruktur. Der Ablauf der Meetings wurde flexibel gestaltet, wobei die neu abgeschlossenen Arbeitspakete, die nächsten Schritte und die Planung der kommenden zwei Wochen im Fokus standen. Durch die Kombination verschiedener Kommunikationsmittel konnte eine effiziente Zusammenarbeit gewährleistet werden, die sowohl die Koordination als auch die praktische Umsetzung der Projektaufgaben unterstützte.

A.1.1. Zeitplan

Tabelle A.1 zeigt den Projektzeitplan anhand der Meilensteine. Die Zeiträume entsprechen den Start- und Fälligkeitsdaten der Meilensteine. Die Zeitführung erfolgte in GitLab, wo die Arbeitspakete als Issues mit entsprechenden Due Dates angelegt, sowie die entsprechenden Zeitaufwände dokumentiert wurden. Die Meilensteine wurden als Milestones in GitLab definiert und fungierten als Sprint-Ziele, die jeweils mehrere Issues bündelten. Durch die konsequente Nutzung von GitLab für die Planung, Dokumentation und Nachverfolgung der Arbeitspakete konnte der Projektfortschritt transparent und nachvollziehbar gestaltet werden.

Tabelle A.1.: Zeitplan des Projekts anhand der Meilensteine

Meilenstein	Start	Ende	Feb 2	Mär 1	Mär 2	Apr 1	Apr 2	Mai 1	Mai 2	Jun 1
01 - Grundlagenforschung	16.02.2026	27.02.2026								
02 - Modell- und Hypothesenentwicklung	02.03.2026	16.03.2026								
03 - Grundlagen und Evaluationskonzept	17.03.2026	30.03.2026								
04 - Architekturkonzept	31.03.2026	13.04.2026								
05 - Technologiestack und Laboraufbau	14.04.2026	27.04.2026								
06 - Zero-Trust-Implementierung	28.04.2026	11.05.2026								
07 - Monitoring und OT-Integration	12.05.2026	25.05.2026								
08 - Evaluation und Abschluss	26.05.2026	08.06.2026								

A.2. Risikoanalyse

Die folgende Risikomatrix A.1 zeigt die zentralen Risiken, die im Rahmen dieser Bachelorarbeit identifiziert wurden, sowie die jeweils vorgesehenen Massnahmen zur Risikominderung.

A.2.1. Risiken

R1: Begrenzte Projektzeit durch Teilzeitstudium. Da beide Teammitglieder das Studium berufsbegleitend absolvieren, steht für die Bachelorarbeit nur ein begrenztes Zeitfenster pro Woche zur Verfügung. Die parallele Belastung durch Beruf, Studium und private Verpflichtungen kann dazu führen, dass Arbeitspakete nicht wie geplant abgeschlossen werden oder sich einzelne Projektphasen verdichten. Dadurch besteht das Risiko, dass insbesondere in der Implementierungs- und Evaluationsphase zusätzlicher Zeitdruck entsteht.

R2: Verzerrung der Evaluation durch Szenarioauswahl. Da die Angriffs- und Wartungsszenarien im Rahmen der Arbeit selbst definiert wurden, besteht das Risiko, dass sie unbeabsichtigt jene Eigenschaften besonders stark gewichten, in denen eine Zero-Trust-Architektur erwartungsgemäss Vorteile bietet. Dies betrifft insbesondere identitätsbasierte Zugriffskontrolle, laterale Bewegung, Service-Sichtbarkeit und gezielte Eindämmung. Dadurch könnte der Vergleich mit der perimeterbasierten Architektur weniger ausgewogen wirken und die Aussagekraft der Ergebnisse einschränken.

R3: Zeitliche Ausfälle und eingeschränkte Verfügbarkeit. Geplante und ungeplante Abwesenheiten einzelner Teammitglieder können die kontinuierliche Bearbeitung der Arbeit beeinträchtigen. Dazu zählen beispielsweise Krankheit, berufliche Verpflichtungen, politische Termine oder andere fixe Abwesenheiten. Solche Ausfälle können zu Verzögerungen führen, insbesondere wenn Aufgaben stark von einer einzelnen Person abhängig sind oder gemeinsame Abstimmungen kurzfristig nicht stattfinden können.

R4: Abhängigkeit von externer Laborinfrastruktur. Die Evaluation basiert auf einer verteilten Laborumgebung mit mehreren Systemen, darunter VPS-Instanzen, ein On-Premises-Host, Docker-Services sowie OT-nahe Komponenten. Fällt ein Teil dieser Infrastruktur aus oder ist sie während einer geplanten Evaluationsphase nicht erreichbar, können Testszenarien nicht wie vorgesehen durchgeführt oder reproduziert werden. Dies kann zu Verzögerungen im Projektablauf führen und die Vergleichbarkeit einzelner Messergebnisse beeinträchtigen.

A.2.2. Geplante Risikominderung

Zur Minimierung der Risiken wurden für jedes Risiko geeignete Massnahmen definiert:

Für **R1 (Begrenzte Projektzeit durch Teilzeitstudium)** wird eine realistische Sprint- und Wochenplanung mit klar priorisierten Arbeitspaketen eingesetzt. Projektzeiten werden frühzeitig reserviert und regelmässig überprüft. Durch Timeboxing, klare Zuständigkeiten und eine bewusste Trennung zwischen Pflicht- und Zusatzaufgaben soll verhindert werden, dass sich einzelne Phasen unkontrolliert ausweiten.

Für **R2 (Verzerrung der Evaluation durch Szenarioauswahl)** wird die Auswahl der Szenarien transparent begründet und an der Forschungsfrage ausgerichtet. Zusätzlich werden die Annahmen und Grenzen der Laborumgebung dokumentiert. Die Ergebnisse werden nicht als allgemeingültige

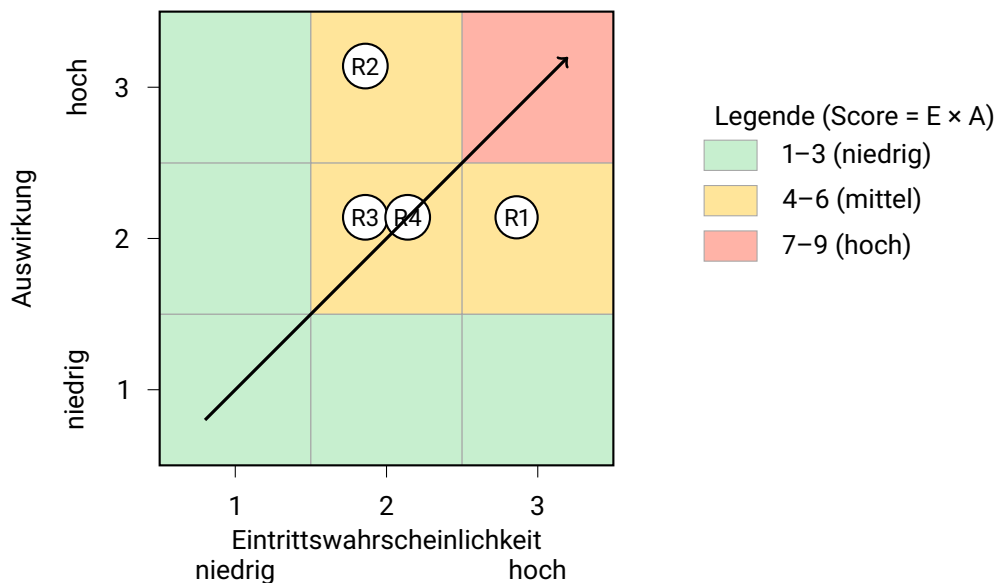


Abbildung A.1.: 3x3-Risikomatrix mit vier identifizierten Risiken

Aussage für alle IT-/OT-Umgebungen dargestellt, sondern auf die untersuchte Architektur und die definierten Szenarien bezogen.

Um **R3 (Zeitliche Ausfälle und eingeschränkte Verfügbarkeit)** zu begegnen, werden geplante Abwesenheiten frühzeitig im Team abgestimmt und bei der Sprintplanung berücksichtigt. Aufgaben werden so verteilt, dass keine kritischen Einzelabhängigkeiten entstehen. Durch regelmässige Abstimmungen, gemeinsame Dokumentation und kurze Übergaben soll sichergestellt werden, dass Arbeitspakete bei Bedarf vom jeweils anderen Teammitglied weitergeführt werden können.

Für **R4 (Abhängigkeit von externer Laborinfrastruktur)** werden zentrale Komponenten frühzeitig aufgebaut und regelmässig auf ihre Verfügbarkeit geprüft. Die Evaluationsszenarien werden mehrfach ausgeführt und die Ergebnisse mit Logs, Zeitstempeln und Zustandsdaten dokumentiert. Zusätzlich werden Wiederherstellungsschritte und alternative Durchführungszeitpunkte eingeplant, falls einzelne Systeme temporär nicht verfügbar sind.

A.2.3. Rückblickende Risikoanalyse

Die folgende Risikomatrix A.2 zeigt die Bewertung der Risiken zum Projektende. Durch die konsequente Umsetzung der geplanten Massnahmen konnten alle Risiken im niedrigen Bereich gehalten werden.

A.2.3.1. Bewertung der eingetretenen Risiken

R1: Begrenzte Projektzeit durch Teilzeitstudium.

Die eingeschränkte verfügbare Projektzeit war während des gesamten Semesters spürbar, insbesondere in Phasen mit hoher beruflicher oder studienbezogener Belastung. Durch die strukturierte Planung, klare Priorisierung und zusätzliche Zeitressourcen in der Schlussphase konnte das Risiko jedoch ausreichend kontrolliert werden. Kritische Arbeitspakete konnten dadurch innerhalb des verfügbaren Zeitrahmens abgeschlossen werden.

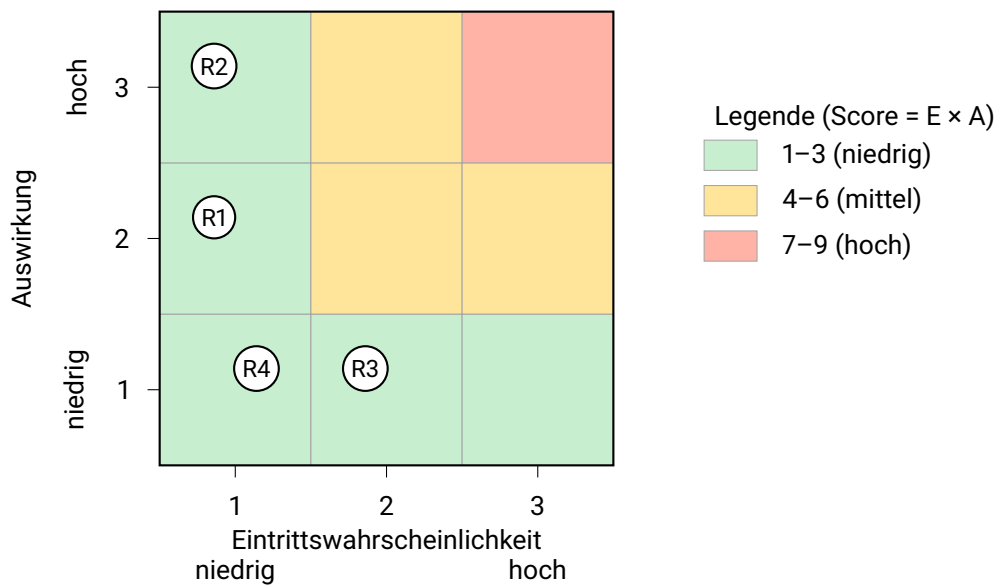


Abbildung A.2.: Aktualisierte Risikomatrix zum Projektabschluss

R2: Verzerrung der Evaluation durch Szenarioauswahl.

Die Szenarien wurden während der Evaluation kritisch geprüft und jeweils mit klaren Annahmen, erwarteten Zugriffen und dokumentierten Einschränkungen versehen. Dadurch konnte das Risiko reduziert werden, dass die Ergebnisse als zu allgemein oder einseitig interpretiert werden. Die Aussagekraft bleibt bewusst auf die untersuchte Laborumgebung und die definierten Vergleichsszenarien beschränkt.

R3: Zeitliche Ausfälle und eingeschränkte Verfügbarkeit.

Während des Projekts kam es zu einzelnen Phasen eingeschränkter Verfügbarkeit, unter anderem durch krankheitsbedingte Ausfälle und fixe externe Termine. Da relevante Informationen dokumentiert und Aufgaben nicht ausschliesslich von einer Person abhängig waren, konnten diese Ausfälle weitgehend kompensiert werden. Der Projektfortschritt wurde dadurch punktuell verlangsamt, jedoch nicht wesentlich gefährdet.

R4: Abhängigkeit von externer Laborinfrastruktur.

Die Abhängigkeit von der verteilten Laborinfrastruktur blieb während des Projekts ein relevantes Risiko, da mehrere Systeme koordiniert betrieben werden mussten. Durch regelmässige Kontrollen, dokumentierte Wiederherstellungsschritte und wiederholte Evaluationsläufe konnten Ausfälle oder kurzfristige Einschränkungen jedoch abgefedert werden. Die Durchführung der zentralen Testszenarien war dadurch nicht wesentlich gefährdet.

B. Selbstreflexion

B.1. Selbstreflexion Ivo Cavelti

Die Bachelorarbeit war für mich eine intensive, aber auch sehr lehrreiche Erfahrung. Rückblickend hat mir die vorgängige Studienarbeit stark dabei geholfen, mich auf die Bachelorarbeit und die damit verbundenen Herausforderungen vorzubereiten. Viele Abläufe, Erwartungen und Schwierigkeiten waren mir dadurch bereits vertrauter, was mir half, die Arbeit bewusster anzugehen.

Ein wichtiger Unterschied zur Studienarbeit war die Verteilung des Aufwands. Bei der Studienarbeit konzentrierte sich ein grosser Teil der Arbeit auf die letzten Tage, was zu einer sehr hohen Belastung gegen Ende führte. Für die Bachelorarbeit gelang es uns besser, den Aufwand über den gesamten Zeitraum zu verteilen. Das hat nicht nur den Druck reduziert, sondern auch dazu beigetragen, dass wir strukturierter und kontrollierter arbeiten konnten.

Trotzdem habe ich während der Arbeit erneut gemerkt, dass es mir bei Projekten über einen längeren Zeitraum manchmal Mühe bereitet, konstant dranzubleiben und mich immer wieder neu zu motivieren. Gerade bei umfangreichen Arbeiten ist es nicht immer einfach, kontinuierlich Fortschritte zu machen, auch wenn das Ziel klar ist. In dieser Hinsicht war die Bachelorarbeit für mich auch eine persönliche Herausforderung, da sie viel Selbstdisziplin und Durchhaltevermögen erforderte.

Sehr hilfreich war für mich die Möglichkeit, einen eigenen Raum im CM-Labor nutzen zu können. Dadurch konnte ich mir an der OST einen festen Arbeitsplatz einrichten, was mir half, mich besser auf die Arbeit zu fokussieren und eine gewisse Routine aufzubauen. Ebenfalls rückblickend sehr sinnvoll war die Entscheidung, zwei Wochen freizunehmen. Dadurch konnte ich mich in dieser wichtigen Phase vollständig auf die Bachelorarbeit konzentrieren, ohne parallel durch andere Verpflichtungen abgelenkt zu sein.

Die Zusammenarbeit mit Baruyr empfand ich als sehr positiv. Da wir seit rund vier Jahren fast alle Projekte gemeinsam durchführen, kennen wir unsere Arbeitsweise gut. Das hat die Zusammenarbeit deutlich erleichtert. Wir hatten eine ähnliche Vorstellung davon, wie die Arbeit aufgebaut und umgesetzt werden sollte, und konnten uns dadurch effizient abstimmen. Diese gemeinsame Erfahrung war besonders wertvoll, da sie viele Diskussionen vereinfachte und uns half, als Team fokussiert zu bleiben.

Mit dem Resultat der Bachelorarbeit bin ich grundsätzlich zufrieden. Gleichzeitig wurde mir erneut bewusst, dass mir diese Art von Arbeit vermutlich nie ganz einfach fallen wird und mich auch in Zukunft stark fordern dürfte. Dennoch gehört es für mich dazu, sich solchen Herausforderungen zu stellen, sich manchmal zu Dingen zu zwingen und in schwierigen Phasen durchzubeissen. Gerade deshalb war die Bachelorarbeit für mich nicht nur fachlich, sondern auch persönlich eine wertvolle Erfahrung.

B.2. Selbstreflexion Baruyr Bousnaian

Der Einstieg in die Bachelorarbeit fiel mir deutlich leichter als noch bei der Studienarbeit. Ich wusste bereits, wie eine grössere wissenschaftliche Arbeit abläuft, und musste mich entsprechend weniger lange orientieren. Das half mir vor allem dabei, die Arbeit zu strukturieren und den Aufwand realistisch einzuschätzen.

Auch die Motivation war von Anfang an höher, weil mich das Thema persönlich stärker interessierte als frühere Projekte. Zero Trust traf genau mein Interessengebiet, und entsprechend engagiert setzte ich mich mit den Inhalten auseinander. Besonders motivierend fand ich, dass ich neue Technologien kennenlernen und gleich praktisch ausprobieren konnte.

Das Projekt brachte aber auch Herausforderungen mit sich. In der Konzeptionsphase war ich mir lange nicht sicher, ob unsere Architekturüberlegungen in der Praxis wirklich funktionieren würden. Um diese Unsicherheit zu verringern, haben wir viel experimentiert und verschiedene Technologien getestet. Ganz loswerden liess sie sich aber nicht. Deshalb achteten wir bewusst auf eine möglichst modulare Architektur, damit sich einzelne Komponenten bei Bedarf austauschen oder anpassen lassen.

Rückblickend floss ein grosser Teil des Aufwands in die Einarbeitung ins Thema Zero Trust. Die Vergleichsarchitektur fiel deshalb kleiner aus als ursprünglich geplant. Für den gegebenen Rahmen war dieser Fokus aus meiner Sicht trotzdem richtig, weil zuerst ein solides Verständnis von Zero Trust aufgebaut werden musste. Bei einer zukünftigen Arbeit würde ich den Scope aber bewusster abwägen, gerade weil die thematische Grundlage jetzt vorhanden ist.

Persönlich habe ich einmal mehr gemerkt, dass ich unter moderatem Zeitdruck produktiver arbeite. In der Mitte des Projekts liess meine Arbeitsmoral teilweise nach, gegen Ende stieg sie dann wieder deutlich an. Am Schluss hat das zwar zu einem erfolgreichen Abschluss geführt, mir aber auch gezeigt, dass eine konstantere Arbeitsweise in zukünftigen Projekten langfristig sinnvoller wäre.

Positiv war ausserdem, dass die Belastung neben Beruf und Studium diesmal insgesamt geringer ausfiel als während der Studienarbeit. Der zusätzliche Aufwand der Bachelorarbeit liess sich dadurch trotz paralleler Verpflichtungen gut bewältigen, und es blieb genug Raum, um mich vertieft mit dem Thema zu beschäftigen.

C. Evaluationsprotokolle

Dieser Anhang sammelt die detaillierten Protokolle der einzelnen Evaluationsläufe. Im Hauptteil werden nur die vergleichbaren Resultate dargestellt; die vollständigen Einzelschritte, Zeitpunkte, Kommandos, Screenshots, Logauszüge und Rohdaten werden hier abgelegt. Dadurch bleiben die Ergebnisse im Haupttext übersichtlich, während die Durchführung weiterhin nachvollziehbar bleibt. Die Protokolltabellen umfassen die Zero-Trust-Läufe in Tabelle C.1, Tabelle C.9, Tabelle C.10, Tabelle C.14, Tabelle C.11, Tabelle C.12 und Tabelle C.13 sowie die perimeterbasierten Läufe in Tabelle C.2, Tabelle C.3, Tabelle C.4, Tabelle C.5, Tabelle C.6, Tabelle C.7 und Tabelle C.8.

C.1. Protokollvorlage

Für jedes Szenario wird dieselbe Protokollstruktur verwendet:

- Szenario-ID und Run-ID
- Datum, Uhrzeit und beteiligte Personen
- Ausgangszustand der Umgebung
- verwendete Architekturvariante
- durchgeführte Schritte und verwendete Kommandos
- erhobene Messwerte und Beobachtungen
- relevante Logs, Screenshots oder Konfigurationsauszüge
- Abweichungen, Fehler oder Wiederholungen des Laufs

C.2. Protokoll A1 – Phishing und Credential Theft

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario A1. Der Test simuliert keinen realen Phishing-Versand, sondern verwendet kontrolliert die kompromittierten Zugangsdaten des Benutzers `bob`. Dadurch kann untersucht werden, welche Dienste nach einer erfolgreichen Authentifizierung tatsächlich erreichbar sind und wie gezielt die kompromittierte Identität eingedämmt werden kann.

C.2.1. Zero-Trust-Architektur

1. Die Zero-Trust-Dienste werden gestartet und der Ausgangszustand wird geprüft.
2. Es wird sichergestellt, dass `bob` nicht bereits blockiert ist.

3. Der Ausgangszustand der Identität bob und die erwarteten Rollen werden als Baseline dokumentiert.
4. Die kompromittierten Zugangsdaten bob / bob-password werden kontrolliert vom Testclient verwendet.
5. Der Angreifer verwendet die kompromittierten Zugangsdaten für einen kontrollierten Keycloak-Token-Request.
6. Die Beobachtungen aus Keycloak, OpenZiti, OPA und der Wazuh-OPA-Bridge werden gesammelt.
7. Die Identität bob@example.test wird über die Wazuh-OPA-Bridge blockiert.
8. Die Blockliste der Wazuh-OPA-Bridge wird geprüft, um die Eindämmung zu bestätigen.
9. Zum Abschluss wird die Blockierung aufgehoben und der legitime Ausgangszustand wiederhergestellt.

Feld	Wert
Run-ID	A1-ZT-001
Datum und Uhrzeit	04.06.2026, 11:30:04–11:30:46 UTC
Architekturvariante	Zero Trust
Kompromittierte Identität	bob / bob@example.test
Simulierter Angriffsvektor	Kontrollierte Verwendung kompromittierter Zugangsdaten statt realem Phishing-Versand
Erwartet erlaubte Dienste	intranet.ziti:80, kuma.ziti:80, ot-dashboard.ziti:80
Erwartet blockierte Dienste	accounting.ziti:80, mqtt.ziti:1883, ot-dht11.ziti:502, guacamole.ziti:80, win-rdp.ziti:3389, vps1-ssh.ziti:22, vps2-ssh.ziti:22, onprem-ssh.ziti:22
Erfolgreiche Authentifizierung	Ja. Der kontrollierte Token-Request an Keycloak war erfolgreich und lieferte einen HTTP-Status 200. Tokens wurden in den Rohdaten redigiert.
Tatsächlich erreichte Dienste vor Eindämmung	intranet.ziti:80, kuma.ziti:80, ot-dashboard.ziti:80 wurden in der manuellen Prüfung als erlaubt bestätigt.
Tatsächlich blockierte Dienste vor Eindämmung	accounting.ziti:80, mqtt.ziti:1883, ot-dht11.ziti:502, guacamole.ziti:80, win-rdp.ziti:3389, vps1-ssh.ziti:22, vps2-ssh.ziti:22, onprem-ssh.ziti:22 wurden in der manuellen Prüfung als verweigert bestätigt.
Erkennungsnachweise	Keycloak-Tokenantwort, Zeitstempel des Angriffsschritts, manuelle Service-Prüfung und Logauszüge der beteiligten Container im Verzeichnis evaluation/A1/A1-ZT-001/.
Time to Detect (TTD)	Nicht als automatische Alarmzeit gemessen; der Testlauf dokumentiert die manuelle Auswertung der erzeugten Artefakte.
Eindämmungsmassnahme	Blockierung von bob@example.test über die Wazuh-OPA-Bridge
Time to Contain (TTC)	Ca. 1 Sekunde für den automatisierten Blockierungsschritt: 11:30:22Z bis 11:30:23Z.
Zugriffsergebnis nach Eindämmung	Die Bridge führte bob@example.test in der Blockliste. Nach dem Restore war die Blockliste wieder leer.
Auswirkungen auf legitime Benutzer	Die Massnahme betraf gezielt die Identität bob@example.test; Server-Blocklisten blieben leer.
Relevante Logs oder Screenshots	evaluation/A1/A1-ZT-001/keycloak-token-result.json, service-results.csv, contain-block-response.json, blocked-users-after.json, restore-unblock-all.json, timestamps.txt, keycloak.log, wazuh-opa-bridge.log.
Abweichungen oder Einschränkungen	Der Test simuliert Credential Theft kontrolliert über bekannte Labordaten und führt keinen realen Phishing-Versand durch.

Tabelle C.1.: Protokollfelder für A1 in der Zero-Trust-Architektur

C.3. Protokoll A1 – Perimeterbasierte Architektur

Der perimeterbasierte Lauf A1-PERIMETER-REAL-001 dokumentiert Credential-Theft aus Sicht veröffentlichter Dienste und VPN-/Netzvoraussetzungen. Der Lauf prüft nicht eine OpenZiti-Service-Liste pro Identität, sondern die aus Netzposition, veröffentlichten Ports, WireGuard-Zustand, Routen und Firewallzustand resultierende Erreichbarkeit.

Feld	Wert
Run-ID	A1-PERIMETER-REAL-001
Datum und Uhrzeit	05.06.2026, 21:06:05Z–21:08:07Z
Architekturvariante	Perimeterbasierte WireGuard-/VPN-Architektur mit veröffentlichten Perimeterdiensten.
Szenarioziel	Bewertung, welche Netzwerk- und Dienstpfade bei missbräuchlich verwendeten Applikationszugangsdaten beziehungsweise vorhandenen Perimeterpfaden sichtbar sind.
Ablauf	precheck, baseline, attack, contain, collect, restore; die Schritte sind in timestamps.txt dokumentiert.
Erhobene Artefakte	wireguard-state.log, wireguard-config-redacted.log, vps1-routes.log, vps2-routes.log, onprem-routes.log, vps1-docker-ps.log, vps2-docker-ps.log, onprem-docker-ps-sudo.log, reachability-from-vps1.log, reachability-from-vps2.log, reachability-from-onprem.log, credential-use-finding.md, containment-finding.md.
Erreichbarkeit	reachability-from-vps1.log bestätigt 127.0.0.1:18080 auf VPS1 als offen sowie 80 und 443 auf VPS2 als offen. Die geprüften On-Premises-Ziele 192.168.2.201:22, :18083, :18084, :1883 und :502 liefen aus dieser Perspektive in Timeouts.
Credential-Finding	credential-use-finding.md hält fest, dass gestohlene Applikationszugangsdaten bei veröffentlichten oauth2-proxy-Diensten verwendet werden können, für private Dienste aber weiterhin VPN-Mitgliedschaft oder ein erreichbarer veröffentlichter Port als Netzwerkvoraussetzung besteht.
Eindämmung	containment-finding.md dokumentiert keine Wazuh/OpenZiti-äquivalente Benutzerblockierung. Praktische Massnahmen liegen bei IdP-Sperre, WireGuard-Peer-Entfernung oder Port-/Firewall-/Routenänderungen.
Restore	Der Restore-Schritt veränderte keine Zero-Trust-Artefakte; temporäre Perimeterobjekte waren in A1 nicht Gegenstand des Szenarios.
Einschränkungen	Es wurden keine destruktiven Firewall- oder Routingänderungen durchgeführt. On-Premises-Nachweise hängen von erreichbarem SSH-/Sudo-Zugriff und den tatsächlich offenen Netzpfeilen ab.

Tabelle C.2.: Protokollfelder für A1 in der perimeterbasierten Architektur

C.4. Protokoll A2 – Perimeterbasierte Architektur

Der Lauf A2-PERIMETER-REAL-001 dokumentiert den kompromittierten VPN-/Endpoint-Kontext. Im Unterschied zur Zero-Trust-Variante wird keine Ziti-Client-API-Session geöffnet und keine dienst-

bezogene Dial-Policy enumeriert. Bewertet werden stattdessen WireGuard-Peers, AllowedIPs, Routen, veröffentlichte Ports und TCP-Erreichbarkeit.

Feld	Wert
Run-ID	A2-PERIMETER-REAL-001
Architekturvariante	Perimeterbasierte WireGuard-/VPN-Architektur.
Szenarioziel	Bewertung, welche Erreichbarkeit ein kompromittierter Endpoint im VPN-/Netzkontext besitzt und wie dessen Eindämmung im Perimetermodell erfolgt.
Vorgehen	Erfassung von WireGuard-Zustand, AllowedIPs, Routingtabellen, Docker-/Hostzustand, Firewallartefakten und TCP-Erreichbarkeit aus mehreren Perspektiven.
WireGuard-Zustand	Das Interface wg0 lief mit 10.200.0.1/24. Ein Peer hatte AllowedIPs für 10.200.0.2/32, 192.168.0.0/24, 192.168.1.0/24 und 192.168.2.0/24; ein zweiter Peer war auf 10.200.0.10/32 beschränkt.
Erhobene Artefakte	wireguard-state.log, wireguard-config-redacted.log, vps1-routes.log, vps2-routes.log, onprem-routes.log, reachability-from-vps1.log, reachability-from-vps2.log, reachability-from-onprem.log, Docker- und Firewall-Logs sowie containment-finding.md.
Eindämmungsbefund	containment-finding.md hält fest, dass ein kompromittierter VPN-Endpoint auf WireGuard-Peer-, Routen-, Firewall- oder Service-Port-Ebene eingedämmt wird. Eine beobachtete per Applikationsidentität wirkende Sperre analog zur Deaktivierung einer einzelnen Ziti-Identität lag nicht vor.
Einordnung	Die Perimeterarchitektur begrenzt den Endpoint nicht durch eine pro Dienst ausgewertete Client-Policy, sondern durch Peer-Konfiguration, AllowedIPs, Routen, Firewallzustand und laufende Dienste.
Einschränkungen	Es wurden keine destruktiven Firewall- oder Routingänderungen durchgeführt; der Lauf dokumentiert vorhandene Laufzeit- und Architektureigenschaften.

Tabelle C.3.: Protokollfelder für A2 in der perimeterbasierten Architektur

C.5. Protokoll A3 – Perimeterbasierte Architektur

Der Lauf A3-PERIMETER-REAL-001 prüfte laterale Erreichbarkeit aus Sicht eines kompromittierten Cloud- beziehungsweise VPS-Systems. Im Perimetermodell ergibt sich das Pivot-Potenzial aus Netzpfeilen, offenen Hostports, Firewallzustand und laufenden Diensten.

Feld		Wert
Run-ID		A3-PERIMETER-REAL-001
Architekturvariante		Perimeterbasierte WireGuard-/VPN-Architektur mit veröffentlichten Diensten.
Szenarioziel		Bewertung der lateralen Erreichbarkeit von einem kompromittierten VPS-/Cloud-Kontext aus.
Zentrales Artefakt		lateral-from-vps1.log dokumentiert die geprüften Zielsysteme und Ports aus Sicht von VPS1.
Erreichbare Ziele		VPS2 war auf 22, 80 und 443 erreichbar.
Nicht erreichte Premises-Ziele	On-	192.168.2.201:22, :18083, :18084, :1883 und :502 liefen in Timeouts.
Erhobene Artefakte		lateral-from-vps1.log, wireguard-state.log, wireguard-config-redacted.log, Routen-, Docker-, Firewall- und Hostzustandsdateien sowie containment-finding.md.
Eindämmungsbefund		containment-finding.md beschreibt Eindämmung über Container-/Dienststopp, Host-Firewall, Portschiessung oder Routenänderung. Dies ist Host- beziehungsweise Dienstpfad-Containment, nicht die Deaktivierung einer einzelnen Ziti-Host-Identität.
Einordnung		Der Lauf belegt praktisch erreichbare Pivot-Ziele zu VPS2 und gleichzeitig eine Einschränkung des beobachteten Netzpfads nach On-Premises. Die Aussage gilt für den gemessenen Laborpfad, nicht für alle möglichen produktionsnahen Perimetersegmentierungen.
Einschränkungen		Es wurde kein destruktiver Exploit und keine produktive Sperrmassnahme durchgeführt; die Perimeter-Containment-Aussage beruht auf vorhandenen Laufzeit- und Architekturartefakten.

Tabelle C.4.: Protokollfelder für A3 in der perimeterbasierten Architektur

C.6. Protokoll A4 – Perimeterbasierte Architektur

Der Lauf A4-PERIMETER-REAL-001 bewertet den externen Service-Techniker als VPN-/Peer-Kontext. Der zentrale Unterschied zur Zero-Trust-Variante besteht darin, dass kein dienstbezogenes Wartungsservice Set für die Identität enumeriert wird, sondern die erreichbaren Netze und Dienstpfade des VPN-Kontexts betrachtet werden.

Feld	Wert
Run-ID	A4-PERIMETER-REAL-001
Datum und Uhrzeit	05.06.2026, 21:08:23Z-21:10:21Z
Architekturvariante	Perimeterbasierte WireGuard-/VPN-Architektur.
Szenarioziel	Bewertung, ob ein legitimer, aber missbräuchlich handelnder Service-Techniker auf einzelne OT-Wartungsdienste begrenzt wird oder breitere Netzpfade erhält.
Vorgehen	Erfassung von WireGuard-Peers, AllowedIPs, Routen, Hostzuständen, Firewallzustand und Erreichbarkeit.
Technician-Scope-Finding	technician-scope-finding.md hält fest, dass der Techniker im Perimetermodell als VPN-Peer-/Netzkontext repräsentiert wird. Der beobachtete Peer mit On-Premises-Bezug hatte AllowedIPs für 192.168.0.0/24, 192.168.1.0/24 und 192.168.2.0/24, nicht eine applikationsspezifische Liste einzelner OT-Wartungsdienste.
Eindämmungsbefund	containment-finding.md beschreibt Sperrmöglichkeiten über Credential-Entzug, WireGuard-Peer-Entfernung, Routenentfernung oder Firewalländerung. Eine beobachtete per Benutzer und Dienst granulare Sperre, die andere erlaubte Dienste derselben Identität unverändert lässt, lag nicht vor.
Erhobene Artefakte	wireguard-state.log, wireguard-config-redacted.log, technician-scope-finding.md, containment-finding.md, Routen-, Docker-, Firewall- und Reachability-Logs.
Einschränkungen	Es wurden keine destruktiven Firewall- oder Routingänderungen durchgeführt; bewertet wird die beobachtete Laborarchitektur.

Tabelle C.5.: Protokollfelder für A4 in der perimeterbasierten Architektur

C.7. Protokoll B1 – Perimeterbasierte Architektur

Im Lauf B1-PERIMETER-REAL-001 wurde das Hinzufügen eines neuen Clients als temporärer WireGuard-Peer umgesetzt. Der Lauf prüft damit den Betriebsaufwand eines peerbasierten Remote-Access-Onboardings.

Feld	Wert
Run-ID	B1-PERIMETER-REAL-001
Datum und Uhrzeit	05.06.2026, 21:10:35Z–21:12:41Z
Szenarioziel	Hinzufügen und Entfernen eines temporären Remote-Access-Clients im Perimetermodell.
Änderung	b1-change.log dokumentiert den temporären Peer b1-temp-peer mit 10.200.0.99/32.
Verifikation	b1-verify.log zeigt den zusätzlichen Peer in wg0. Die vorhandenen Routen zu 192.168.0.0/24, 192.168.1.0/24 und 192.168.2.0/24 bleiben über wg0 dokumentiert.
Zeitwert	Die Änderung dauerte gemäss timestamps.txt rund eine Sekunde von 21:11:33Z bis 21:11:34Z.
Restore	Nach Restore zeigte post-restore-wireguard-state.log wieder den bereinigten WireGuard-Zustand ohne temporären b1-temp-peer.
Erhobene Artefakte	b1-change.log, b1-verify.log, b1-cleanup.log, post-restore-wireguard-state.log, wireguard-state.log, timestamps.txt sowie Host-, Docker-, Routen-, Firewall- und Reachability-Artefakte.
Einordnung	Das Onboarding ist technisch kurz und objektarm, aber peer- und IP-basiert. Eine Freigabe nur eines einzelnen Applikationsdienstes wird durch diesen Perimetermechanismus nicht als eigene dienstbezogene Identität modelliert.

Tabelle C.6.: Protokollfelder für B1 in der perimeterbasierten Architektur

C.8. Protokoll B2 – Perimeterbasierte Architektur

Im Lauf B2-PERIMETER-REAL-001 wurde ein zusätzlicher Cloud- beziehungsweise VPS-Dienst als temporärer veröffentlichter Container umgesetzt. Der Lauf prüft damit den Aufwand und die Granularität einer portbasierten Dienstveröffentlichung im Perimetermodell.

Feld	Wert
Run-ID	B2-PERIMETER-REAL-001
Datum und Uhrzeit	05.06.2026, 21:12:41Z–21:15:05Z
Szenarioziel	Hinzufügen und Entfernen eines temporären Cloud-Dienstes über eine veröffentlichte Portfreigabe.
Änderung	b2-change.log dokumentiert den Container b2-cloud-service mit Portabbildung 0.0.0.0:18085->80/tcp.
Verifikation	b2-verify.log bestätigt 127.0.0.1:18085 als offen; der bestehende Intranet-Perimeterdienst auf 127.0.0.1:18080 blieb ebenfalls offen.
Zeitwert	Die Änderung lief gemäss timestamps.txt im Fenster 21:13:47Z bis 21:13:55Z. Innerhalb dieses Fensters wurde vor dem Start ein Restore-Schritt ausgeführt und der Container anschliessend gestartet.
Restore	b2-restore.log dokumentiert die Entfernung von b2-cloud-service.
Erhobene Artefakte	b2-change.log, b2-verify.log, b2-restore.log, timestamps.txt sowie Host-, Docker-, Routen-, Firewall- und Reachability-Artefakte.
Einordnung	Der Dienst wurde über einen Host-Port veröffentlicht. Im Gegensatz zur Zero-Trust-Variante entsteht keine separate Dial-Policy für genau eine neue Identität; die Begrenzung hängt von Hostport, Firewall, Routen und vorgeschalteten Anwendungskontrollen ab.

Tabelle C.7.: Protokollfelder für B2 in der perimeterbasierten Architektur

C.9. Protokoll B3 – Perimeterbasierte Architektur

Im Lauf B3-PERIMETER-REAL-001 wurde ein zusätzlicher OT-naher Dienst direkt auf dem On-Premises-Server als temporärer Container umgesetzt. Der Lauf prüft damit eine lokale portbasierte Erweiterung im OT-/Service-Netz.

Feld	Wert
Run-ID	B3-PERIMETER-REAL-001
Datum und Uhrzeit	05.06.2026, 21:15:05Z–21:17:21Z
Szenarioziel	Hinzufügen und Entfernen eines temporären OT-Sensor-Dienstes auf On-Premises.
Änderung	b3-change-onprem.log dokumentiert den Container b3-ot-sensor mit Portabbildung 0.0.0.0:18086->80/tcp.
Lokale Verifikation	b3-verify-onprem-local.log bestätigt 127.0.0.1:18086 als offen. Gleichzeitig blieben die bestehenden lokalen Perimeterdienste 127.0.0.1:18083 und 127.0.0.1:18084 offen.
Verifikation aus VPS1	b3-verify-vps1-path.log zeigt Timeouts für 192.168.2.201:18086 und 192.168.2.201:18083. Damit war der temporäre Dienst lokal auf On-Premises offen, aus Sicht des gemessenen VPS1-Pfads aber nicht erreichbar.
Zeitwert	Die Änderung lief gemäss timestamps.txt im Fenster 21:16:00Z bis 21:16:05Z. Innerhalb dieses Fensters wurde vor dem Start ein Restore-Schritt ausgeführt und der Container anschliessend gestartet.
Restore	b3-restore-onprem.log entfernt den temporären Container; die anschliessende Abfrage liefert keinen verbliebenen b3-ot-sensor-Container.
Erhobene Artefakte	b3-change-onprem.log, b3-verify-onprem-local.log, b3-verify-vps1-path.log, b3-restore-onprem.log, timestamps.txt sowie Host-, Docker-, Routen-, Firewall- und Reachability-Artefakte.
Einordnung	Die Änderung ist host- und portbezogen. Eine automatische Erweiterung einer einzelnen Rollen- oder Identitätsmatrix wie in Zero Trust wird im Perimeterlauf nicht modelliert; die tatsächliche Erreichbarkeit hängt von lokalem Port, Firewall, Routen und Netzpfad ab.

Tabelle C.8.: Protokollfelder für B3 in der perimeterbasierten Architektur

C.10. Protokoll A2 – Kompromittierter Endpoint

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario A2. Im Unterschied zu A1, wo gestohlene Zugangsdaten ausschliesslich auf HTTP-Dienstebene eingesetzt wurden, simuliert A2 eine vollständige Endpoint-Kompromittierung. Der Angreifer verwendet das gestohlene OIDC-Token, um sich direkt beim OpenZiti-Controller über die Client-API anzumelden – genau wie der OpenZiti Desktop Edge Client auf dem kompromittierten Gerät. Dadurch wird der Angriff auf die Netzwerkebene ausgedehnt.

Warum Option A (Netzwerkebene)? bob ist eine menschliche SSO-Identität in OpenZiti; es existiert

keine enrollte Datei-Identität. Die lokale Zugangsinformation auf dem kompromittierten Endpoint ist das vom OpenZiti Desktop Edge Client zwischengespeicherte Keycloak-OIDC-Token. Option A verwendet dieses Token mit dem Endpoint `/edge/client/v1/authenticate?method=ext-jwt` (Authorization: Bearer), identisch zum tatsächlichen Client-Authentifizierungsfluss. Dies ermöglicht die Enumeration aller Dienste, die die kompromittierte Identität auf Ziti-Ebene erreichen kann.

C.10.1. Zero-Trust-Architektur

1. Die Zero-Trust-Dienste werden gestartet und der Ausgangszustand geprüft.
2. Es wird sichergestellt, dass bob nicht bereits blockiert ist.
3. Der Ausgangszustand der Identität bob und die leere Blockliste werden als Baseline dokumentiert.
4. Das Keycloak-OIDC-Token wird per kontrolliertem Password-Grant simuliert (Credential-Diebstahl vom Endpoint).
5. Das Token wird beim Ziti-Controller über den Client-API-Endpoint `/edge/client/v1/authenticate?method=ext-jwt` eingelöst.
6. Die Ziti-Session enumeriert alle Dienste, die die kompromittierte Identität dialen kann.
7. Die Beobachtungen aus Keycloak, OpenZiti und der Wazuh-OPA-Bridge werden gesammelt.
8. Die Identität bob@example.test wird über die Wazuh-OPA-Bridge blockiert.
9. Der Ziti-Controller bestätigt, dass die Identität deaktiviert ist und keine aktiven API-Sessions mehr bestehen.
10. Zum Abschluss wird die Blockierung aufgehoben und der Ausgangszustand wiederhergestellt.

Feld	Wert
Run-ID	A2-ZT-001
Datum und Uhrzeit	04.06.2026, 12:00:58–12:06:52 UTC
Architekturvariante	Zero Trust
Kompromittierte Identität	bob / bob@example.test
Simulierter Angriffsvektor	Option A: Keycloak-Token vom Endpoint gestohlen, über Ziti-Client-API auf Netzwerkebene eingesetzt
Ziti-Session erfolgreich	Ja. /edge/client/v1/authenticate?method=ext-jwt lieferte HTTP 200.
Dienste gemäss Ziti-Enumeration	intranet, ot-dashboard, uptime-kuma-direct (3 Dienste)
MQTT / Modbus erreichbar	Nein. mqtt-broker und ot-dht11-modbus erscheinen nicht in der Session – bob hat keine Dial-Policy für diese Dienste.
Tatsächlich erreichte Dienste (manuell)	intranet.ziti:80, kuma.ziti:80, ot-dashboard.ziti:80 wurden als erlaubt bestätigt.
Tatsächlich blockierte Dienste (manuell)	accounting.ziti:80, mqtt.ziti:1883, ot-dht11.ziti:502, guacamole.ziti:80, win-rdp.ziti:3389, vps1-ssh.ziti:22, vps2-ssh.ziti:22, onprem-ssh.ziti:22 wurden als verweigert bestätigt.
Unterschied zu A1	In A1 wurde das Token nur auf HTTP-Dienstebene eingesetzt. In A2 wird die Ziti-Netzwerkebene direkt kompromittiert. Das Ergebnis ist identisch: nur die drei Dienste mit bob-Dial-Policy sind erreichbar.
Eindämmungsmassnahme	Blockierung von bob@example.test über die Wazuh-OPA-Bridge (OPA-Ebene + Ziti-Identität deaktiviert)
Ziti-Identität nach Eindämmung	disabled: true, disabledAt: 2026-06-04T12:06:11Z, hasApiSession: false
Time to Contain (TTC)	Ca. 1 Sekunde (12:06:11Z bis 12:06:11Z)
Zugriffsergebnis nach Eindämmung	Ziti-Identität deaktiviert; neue Sessions können nicht mehr geöffnet werden.
Auswirkungen auf legitime Benutzer	Nur bob@example.test betroffen; Server-Blocklisten blieben leer.
Relevante Logs oder Screenshots	evaluation/A2/A2-ZT-001/ziti-service-list.json, ziti-identity-status-after.json, keycloak-token-result.json, contain-block-response.json, timestamps.txt.
Abweichungen oder Einschränkungen	Der Ziti-ext-jwt-Signer musste vor dem Lauf korrigiert werden: Die konfigurierte Issuer-URL (http://localhost:8080) stimmte nicht mit dem tatsächlichen Token-Issuer (https://localhost) überein. Die Abweichung entstand durch den VPS-Compose-Override (-hostname=https://localhost) und wurde per Management-API behoben.

Tabelle C.9.: Protokollfelder für A2 in der Zero-Trust-Architektur

C.11. Protokoll A3 – Exponierter und verwundbarer Cloud-Dienst

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario A3. Es wird kein realer Angriff auf eine Schwachstelle durchgeführt. Stattdessen wird die kompromittierte Seite des Szenarios durch zwei kontrollierte Schritte simuliert: erstens eine Policy-Inspektion zur Ermittlung der Pivot-Möglichkeiten, zweitens eine direkte Blockierung der Server-Identität über die Wazuh-OPA-Bridge.

Zielservers: `ov-626e92` (VPS1), Ziti-Host-Identität: `intranet-host`.

C.11.1. Zero-Trust-Architektur

1. Die Zero-Trust-Dienste werden gestartet; der Ausgangszustand aller Host-Identitäten wird als Baseline dokumentiert.
2. **Pivot-Potenzial:** Die Ziti-Management-API wird auf Dial-Policies der Identität `intranet-host` geprüft. Ergebnis: Nur Bind-Policy vorhanden, keine Dial-Policy. Kein Ziti-basiertes Lateral Movement möglich.
3. **Serverblock:** Die Bridge-API `/block-server` wird aufgerufen, um die kompromittierte Server-Identität zu deaktivieren (simuliert Wazuh-Active-Response).
4. Alle Host-Identitäten werden überprüft: nur `intranet-host` ist deaktiviert, alle anderen bleiben aktiv.
5. Manuelle Service-Verfügbarkeitsprüfung aus Benutzerperspektive.
6. Restore: Blockierung aufgehoben.

Feld	Wert
Run-ID	A3-ZT-001
Datum und Uhrzeit	04.06.2026, 12:37:10–12:37:38 UTC
Architekturvariante	Zero Trust
Zielserver	ov-626e92 / Ziti-Identität: intranet-host
Simulierter Angriffsvektor	Kompromittierung eines exponierten Cloud-Dienstes; keine reale Schwachstellenausnutzung
Pivot-Potenzial (Dial-Policies)	Keine Dial-Policies vorhanden. intranet-host kann über Ziti keine anderen Dienste erreichen. Lateral Movement via Ziti nicht möglich.
Serverblock erfolgreich	Ja. Bridge blockierte ov-626e92 / intranet-host.
Intranet-Host nach Eindämmung	disabled: true, disabledAt: 2026-06-04T12:37:14Z, hasApiSession: false
Andere Host-Identitäten	Alle disabled: false – keine Kollateralschäden.
Time to Contain (TTC)	Ca. 1 Sekunde (12:37:14Z bis 12:37:14Z)
Tatsächlich nicht erreichbar nach Block	intranet.ziti:80: nicht erreichbar (manuelle Prüfung bestätigt).
Tatsächlich erreichbar nach Block	kuma.ziti:80, ot-dashboard.ziti:80, accounting.ziti:80, mqtt.ziti:1883, ot-dht11.ziti:502, guacamole.ziti:80, win-rdp.ziti:3389 und alle SSH-Services: erreichbar (manuelle Prüfung bestätigt).
Auswirkungen auf legitime Nutzer	Nur der Intranet-Dienst betroffen; alle anderen Dienste unverändert verfügbar.
Relevante Logs oder Screenshots	evaluation/A3/A3-ZT-001/ziti-pivot-check.json, block-server-response.json, all-host-identity-status.json, service-results.csv, timestamps.txt.
Abweichungen oder Einschränkungen	wazuh-manager und wazuh-alerts-tailer nicht aktiv (Indexer-Abhängigkeit). Der Bridge-Endpunkt /block-server wurde direkt aufgerufen und entspricht dem Code-Pfad der Wazuh-Active-Response. Vor dem Lauf mussten OPA-Bundle-URL, Envoy-DNS-Konfiguration und intranet-oauth2-proxy-Issuer-Verifikation für die lokale Lab-Umgebung korrigiert werden.

Tabelle C.10.: Protokollfelder für A3 in der Zero-Trust-Architektur

C.12. Protokoll B1 – Hinzufügen eines neuen Clients

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario B1 auf der real verteilten Infrastruktur. Der Lauf wurde unter `evaluation/B1/B1-ZT-REAL-001/` abgelegt.

Feld	Wert
Run-ID	B1-ZT-REAL-001
Datum und Uhrzeit	2026-06-05, 08:15:13–08:15:17 UTC
Neuer Client	b1-maint-client
Änderung	Neue OpenZiti-Identität mit Enrollment-Datei und eine explizite Dial-Policy b1-maint-client-intranet-dial.
Zielservice	intranet
Objektänderung	Identitäten: 23 → 24, Service-Policies: 24 → 25, Services unverändert bei 11.
Gemessene Einrichtungszeit	ca. 2 Sekunden (08:15:14Z bis 08:15:16Z).
Ergebnis der Zugriffssichtbarkeit	b1-maint-client sah ausschliesslich intranet. Die Sichtbarkeit von bob und service-technician blieb unverändert.
Wiederherstellung	Erfolgreich; nach Restore wieder 23 Identitäten und 24 Service-Policies, keine b1-*-Objekte vorhanden.
Relevante Artefakte	baseline-snapshot.json, verify-snapshot.json, post-restore-snapshot.json, b1-maint-client.identity.json, timestamps.txt.

Tabelle C.11.: Praktisches Protokoll für B1 in der Zero-Trust-Architektur

C.13. Protokoll B2 – Hinzufügen eines neuen Cloud-Dienstes

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario B2 auf der real verteilten Infrastruktur. Der Lauf wurde unter `evaluation/B2/B2-ZT-REAL-001/` abgelegt.

Feld	Wert
Run-ID	B2-ZT-REAL-001
Datum und Uhrzeit	2026-06-05, 08:15:17–08:15:23 UTC
Neuer Dienst	b2-cloud-service mit Intercept b2-cloud.ziti:80.
Temporärer Client	b2-cloud-client
Änderung	Zwei Configs, ein Service, eine Client-Identität, eine Bind-Policy und eine Dial-Policy.
Bind/Dial	b2-cloud-service-bind bindet an intranet-host; b2-cloud-service-dial erlaubt nur b2-cloud-client.
Objektänderung	Configs: 24 → 26, Identitäten: 23 → 24, Service-Policies: 24 → 26, Services: 11 → 12.
Gemessene Einrichtungszeit	ca. 3 Sekunden (08:15:18Z bis 08:15:21Z).
Ergebnis der Zugriffssichtbarkeit	b2-cloud-client sah ausschliesslich b2-cloud-service. Die Sichtbarkeit von bob und service-technician blieb unverändert.
Wiederherstellung	Erfolgreich; nach Restore wieder 24 Configs, 23 Identitäten, 24 Service-Policies und 11 Services, keine b2-*-Objekte vorhanden.
Relevante Artefakte	baseline-snapshot.json, verify-snapshot.json, post-restore-snapshot.json, b2-cloud-client.identity.json, timestamps.txt.

Tabelle C.12.: Praktisches Protokoll für B2 in der Zero-Trust-Architektur

C.14. Protokoll B3 – Hinzufügen eines neuen Sensors

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario B3 auf der real verteilten Infrastruktur. Der Lauf wurde unter `evaluation/B3/B3-ZT-REAL-001/` abgelegt.

Feld	Wert
Run-ID	B3-ZT-REAL-001
Datum und Uhrzeit	2026-06-05, 08:15:23–08:15:29 UTC
Neuer Sensor-Dienst	b3-ot-sensor mit Intercept b3-ot-sensor.ziti:1883.
Temporärer Client	b3-ot-sensor-client
Änderung	Zwei Configs, ein Service, eine Client-Identität, eine Bind-Policy und eine Dial-Policy.
Bind/Dial	b3-ot-sensor-bind bindet an onprem-services-host; b3-ot-sensor-dial erlaubt nur b3-ot-sensor-client.
Objektänderung	Configs: 24 → 26, Identitäten: 23 → 24, Service-Policies: 24 → 26, Services: 11 → 12.
Gemessene Einrichtungszeit	ca. 3 Sekunden (08:15:24Z bis 08:15:27Z).
Ergebnis der Zugriffssichtbarkeit	b3-ot-sensor-client sah ausschliesslich b3-ot-sensor. Die Sichtbarkeit von bob und service-technician blieb unverändert.
Wiederherstellung	Erfolgreich; nach Restore wieder 24 Configs, 23 Identitäten, 24 Service-Policies und 11 Services, keine b3-*-Objekte vorhanden.
Relevante Artefakte	baseline-snapshot.json, verify-snapshot.json, post-restore-snapshot.json, b3-ot-sensor-client.identity.json, timestamps.txt.

Tabelle C.13.: Praktisches Protokoll für B3 in der Zero-Trust-Architektur

C.15. Protokoll A4 – Insider-Threat durch externen Service-Techniker

Dieses Protokoll dokumentiert die Zero-Trust-basierte Durchführung von Szenario A4. Ein externer Service-Techniker besitzt legitime Zugangsdaten und versucht, über seine Wartungsrolle hinaus auf IT- und Administrationsdienste zuzugreifen. Im Unterschied zu A1 und A2 handelt es sich nicht um gestohlene Zugangsdaten: Die Identität `service-technician` ist korrekt authentifiziert. Der Test prüft, ob Zero Trust die Rollengrenze auch für eine legitim authentifizierte Identität durchsetzt.

C.15.1. Zero-Trust-Architektur

1. Die Zero-Trust-Dienste werden gestartet; der Ausgangszustand wird als Baseline dokumentiert.
2. Der Service-Techniker authentifiziert sich mit seinen legitimen Zugangsdaten bei Keycloak.
3. Mit dem erhaltenen OIDC-Token wird eine Ziti-Client-API-Session geöffnet.
4. Die Session enumeriert die dialisierbaren Dienste: nur die 4 der Rolle `service-technicians` zugeordneten Dienste erscheinen.
5. IT- und Administrationsdienste sind in der Session nicht sichtbar.
6. Die Identität `service-technician@example.test` wird über die Wazuh-OPA-Bridge blockiert.

7. Restore: Blockierung aufgehoben.

Feld	Wert
Run-ID	A4-ZT-001
Datum und Uhrzeit	04.06.2026, 12:57:51–12:58:03 UTC
Architekturvariante	Zero Trust
Identität	service-technician / service-technician@example.test
Zugewiesene Rolle	service-technicians
Angriffsvektor	Missbrauch legitimer Zugangsdaten (kein Credential Theft)
Keycloak-Authentifizierung	Erfolgreich; HTTP-Status 200. Tokens redigiert.
Dienste in der Ziti-Session	Exakt 4 Dienste: guacamole-svc, mqtt-broker, ot-dashboard, ot-dht11-modbus.
IT-/Admin-Dienste sichtbar	Nein. intranet, accounting, uptime-kuma-direct, win-rdp, SSH-Services erscheinen nicht.
Tatsächlich erreichte Dienste (manuell)	guacamole.ziti:80, mqtt.ziti:1883, ot-dashboard.ziti:80, ot-dht11.ziti:502 – alle 4 Rollendienste als erreichbar bestätigt.
Tatsächlich blockierte Dienste (manuell)	intranet.ziti:80, accounting.ziti:80, kuma.ziti:80, win-rdp.ziti:3389, vps1-ssh.ziti:22, vps2-ssh.ziti:22, onprem-ssh.ziti:22 – alle 7 nicht autorisierten Dienste als verweigert bestätigt.
Eindämmung	Blockierung von service-technician@example.test über die Wazuh-OPA-Bridge.
Ziti-Identität nach Eindämmung	disabled: true, disabledAt: 2026-06-04T12:58:01Z, hasApiSession: false
Time to Contain (TTC)	Ca. 1 Sekunde (12:58:01Z bis 12:58:02Z)
Auswirkungen auf andere Benutzer	Keine; andere Identitäten unberührt.
Relevante Logs	evaluation/A4/A4-ZT-001/keycloak-token-result.json, ziti-service-list.json, contain-block-response.json, ziti-identity-status-after.json, timestamps.txt
Abweichungen	Keycloak-Realm-Export enthält service-technician nicht; Benutzer wurde vor dem Lauf per Admin-API angelegt. Ziti-ExternalId auf neue Keycloak-UUID aktualisiert.

Tabelle C.14.: Protokollfelder für A4 in der Zero-Trust-Architektur

D. README der Infrastruktur

Dieser Anhang enthält die zentrale README-Datei der Infrastructure-as-Code-Umgebung. Sie beschreibt die Voraussetzungen, die notwendigen Anpassungen für eine eigene Umgebung sowie die schrittweise Inbetriebnahme der Laborinfrastruktur.

```
1 # Zero Trust Lab Infrastructure
2
3 This directory contains the infrastructure-as-code and service code for a Zero Trust bachelor thesis lab.
4   The setup builds a private service fabric with OpenZiti and adds identity, authorization, monitoring,
5   SIEM, and optional perimeter-based comparison components.
6
7 The repository is not meant to be started unchanged on a new environment. Before deployment, replace all
8   example domains, IP addresses, passwords, client secrets, and hostnames with values for your own hosts.
9
10 ## Architecture
11
12 The lab is split across several machines:
13
14 | Host | Purpose |
15 | --- | --- |
16 | Platform host | Runs the central Zero Trust platform: OpenZiti, Keycloak, Caddy, OPA bundle server,
17   intranet demo app, optional Wazuh, and optional Guacamole. |
18 | Monitoring host | Runs the private Uptime Kuma stack and OpenZiti monitoring client sidecars. |
19 | On-prem host | Runs the OT dashboard, accounting demo app, Mosquitto broker, Envoy, OPA, and OpenZiti host
20   tunnel. |
21 | DHT11 Raspberry Pi | Runs the DHT11 Modbus TCP service and hosts it into OpenZiti. |
22 | Fan Raspberry Pi | Runs the MQTT fan client and dials the private MQTT broker through OpenZiti. |
23 | Optional Windows host | Can be reached through Guacamole/RDP if the `win-rdp` OpenZiti service is
24   configured. |
25
26 Core traffic flow for protected web services:
27
28 ```text
29 User or machine identity -> OpenZiti -> oauth2-proxy -> Envoy -> OPA -> private workload
30 ```
31
32 Core traffic flow for OT protocol services:
33
34 ```text
35 Machine identity -> OpenZiti -> private protocol endpoint
36 ```
37
38 ## Directory Layout
39
40 ```text
41 compose/      Docker Compose entry points and overlays for the platform host
42 platform/    Shared infrastructure configuration for OpenZiti, Keycloak, Caddy, OPA, Wazuh, Guacamole, and
43   proxies
44 services/    Protected workloads and host-specific service bundles
45 monitoring/   Uptime Kuma monitor provisioning
46 policy/      Source access matrix for identities, roles, services, and policy generation
47 scripts/     Deployment, generation, enrollment, and maintenance helpers
48 remote-services/ Optional perimeter comparison compose files
49 ```
50
51 ## Prerequisites
52
53 On every Docker-based host:
54
55 - Linux server with SSH access
56 - Docker Engine and Docker Compose plugin
```

```
50 - Git, `curl`, `openssl`, and `python3`
51 - A user that can run Docker directly or through `sudo`
52
53 On the platform host:
54
55 - Public DNS records pointing to the platform host for your own domain names, for example:
56 - `auth.<your-domain>` for Keycloak
57 - `ziti.<your-domain>` for the OpenZiti edge client API
58 - `router.<your-domain>` for the OpenZiti edge router
59 - `wazuh.<your-domain>` if Wazuh dashboard is enabled
60 - `guacamole.<your-domain>` if Guacamole is enabled
61 - `opa-bundles.<your-domain>` for OPA bundle distribution
62 - Inbound firewall rules for `80/tcp`, `443/tcp`, OpenZiti ports, and Wazuh agent ports if Wazuh agents will
   connect remotely
63
64 On Raspberry Pi devices:
65
66 - Python 3 and `systemd --user` support
67 - DHT11 sensor connected to GPIO4 for the DHT11 service
68 - Required hardware libraries for the fan display if using the fan bundle
69
70 ## Important Before You Start
71
72 This repository still contains deployment-specific example values. For a fresh setup, review and replace
   them before deploying.
73
74 At minimum, check these files:
75
76 ```text
77 .env.vps.example
78 .env.example
79 platform/caddy/Caddyfile
80 platform/keycloak/realm-export.json
81 compose/vps.yml
82 compose/guacamole.yml
83 docker-compose.perimeter.yml
84 services/uptime-kuma/.env.example
85 services/onprem/.env.example
86 monitoring/uptime-kuma/monitors.yml
87 scripts/deploy-wazuh-agent-remote.sh
88 ```
89
90 Use your own values for:
91
92 - Public domain names
93 - Public IP addresses
94 - OAuth2 redirect URLs
95 - Caddy virtual hosts
96 - OpenZiti advertised addresses
97 - Keycloak client secrets
98 - Keycloak user passwords
99 - OPA bundle basic-auth credentials
100 - Wazuh API and indexer credentials
101 - SSH targets used by helper scripts
102
103 Do not commit generated identities, enrollment tokens, certificates, `.env` files, OPA bundles, or runtime
   secrets.
104
105 ## 1. Prepare The Platform Host
106
107 Copy or clone this repository onto the platform host and enter the code directory:
108
109 ```bash
110 cd <repo>/code
111 ```
112
113 Create the platform environment file:
114
115 ```bash
116 cp .env.vps.example .env
117 ```
118
```

```
119 Edit `.env` and set values for your environment. Important variables include:
120
121 ```text
122 KEYCLOAK_HOSTNAME=auth.<your-domain>
123 KEYCLOAK_PUBLIC_URL=https://auth.<your-domain>
124 ZITI_CTRL_EDGE_ADVERTISED_ADDRESS=ziti.<your-domain>
125 ZITI_CTRL_EDGE_IP_OVERRIDE=<platform-public-ip>
126 ZITI_ROUTER_ADVERTISED_ADDRESS=router.<your-domain>
127 CADDY_EMAIL=<admin-email>
128 OPA_BUNDLE_BASE_URL=https://opa-bundles.<your-domain>
129 ```
130
131 Generate strong secrets before filling `.env`:
132
133 ```bash
134 openssl rand -hex 32
135 openssl rand -base64 32
136 ```
137
138 For OPA bundle basic auth, choose a username and password, then create the values needed by OPA and Caddy:
139
140 ```bash
141 printf '%s' '<opa-user>:<opa-password>' | base64 -w0
142 docker run --rm caddy:2.9-alpine caddy hash-password --plaintext '<opa-password>'
143 ```
144
145 Set the first command output as `OPA_BUNDLE_AUTH_HEADER` and the second command output as `
    OPA_BUNDLE_BASIC_AUTH_HASH`.
146
147 ## 2. Replace Public Hostnames
148
149 Update Caddy with your public hostnames:
150
151 ```text
152 platform/caddy/Caddyfile
153 ```
154
155 The Caddyfile should contain your own domains for Keycloak, OpenZiti, router WebSocket forwarding, Wazuh,
    Guacamole, and OPA bundle hosting.
156
157 Update Docker Compose overrides that contain host-specific DNS helpers:
158
159 ```text
160 compose/vps.yml
161 compose/guacamole.yml
162 ```
163
164 If your host can resolve your public domains normally, the `extra_hosts` entries can usually be removed. If
    you need local hairpin routing to the Docker host, replace the domain names with your own domains.
165
166 Update Keycloak redirect URLs and client secrets:
167
168 ```text
169 platform/keycloak/realm-export.json
170 ```
171
172 At minimum, replace public redirect URLs for the intranet, Uptime Kuma, OT dashboard, accounting, and
    Guacamole clients. Also replace the demo client secrets and default user passwords.
173
174 ## 3. Review The Access Matrix
175
176 The source of truth for OpenZiti identities, roles, private services, and generated policy data is:
177
178 ```text
179 policy/access-matrix.json
180 ```
181
182 The default matrix creates identities for:
183
184 - Human users such as `alice`, `bob`, `service-technician`, and `ops-admin`
185 - Service hosts such as `intranet-host`, `uptime-kuma-host`, `onprem-services-host`, and `ot-dht11-host`
186 - Machine clients such as `uptime-kuma-monitor`, `ot-dashboard-client`, `ot-fan-client`, and `guac-rdp-
```

```
client`
187 - Private services such as `intranet`, `uptime-kuma-direct`, `ot-dashboard`, `accounting`, `mqtt-broker`, `
    ot-dht11-modbus`, SSH services, and RDP
188
189 If you change users, roles, services, ports, or intercept hostnames, edit `policy/access-matrix.json` first.
190
191 Validate the matrix:
192
193 ```bash
194 ./scripts/access-matrix.py validate
195 ```
196
197 Generate all derived artifacts:
198
199 ```bash
200 ./scripts/access-matrix.py render-all
201 ```
202
203 This regenerates:
204
205 ```text
206 policy/POLICY_MATRIX.md
207 platform/opa/policies/access-matrix.json
208 services/uptime-kuma/opa/policies/access-matrix.json
209 services/onprem/opa/policies/access-matrix.json
210 platform/opa/bundles/*.tar.gz
211 platform/openziti/generated/access-matrix-openziti.sh
212 ```
213
214 ## 4. Generate Wazuh Runtime Files
215
216 If you will run Wazuh, generate certificates and dashboard configuration before starting the SIEM stack:
217
218 ```bash
219 ./scripts/generate-wazuh-certs.sh
220 ./scripts/render-wazuh-dashboard-config.sh
221 ```
222
223 These files are runtime artifacts and must stay out of git.
224
225 ## 5. Start The Central Zero Trust Platform
226
227 Start the platform without optional Guacamole and Wazuh:
228
229 ```bash
230 docker compose \
231   -f compose/zero-trust.yml \
232   -f compose/vps.yml \
233   --profile vps \
234   up -d --build
235 ```
236
237 Start the platform with Guacamole and Wazuh:
238
239 ```bash
240 docker compose \
241   -f compose/zero-trust.yml \
242   -f compose/vps.yml \
243   -f compose/guacamole.yml \
244   -f compose/wazuh.yml \
245   --profile vps \
246   --profile siem \
247   up -d --build
248 ```
249
250 Check service status:
251
252 ```bash
253 docker compose \
254   -f compose/zero-trust.yml \
255   -f compose/vps.yml \
256   --profile vps \
```

```
257  ps
258  ...
259
260  Check the OpenZiti bootstrap logs:
261
262  ```bash
263  docker compose \
264    -f compose/zero-trust.yml \
265    -f compose/vps.yml \
266    --profile vps \
267    logs ziti-bootstrap
268  ```
269
270  The bootstrap container creates OpenZiti services, service policies, identities, and identity JSON files in
    the `ziti-data` Docker volume.
271
272  ## 6. Sync Keycloak Users And Roles
273
274  After Keycloak is reachable, sync human users and realm roles from the access matrix:
275
276  ```bash
277  ./scripts/sync-keycloak-users.py
278  ```
279
280  This script reads `.env` and `policy/access-matrix.json`. If you changed human identity IDs in the matrix,
    keep those IDs aligned with Keycloak users so OpenZiti external JWT authentication can match the
    correct users.
281
282  If Guacamole is enabled, create or update its Keycloak client:
283
284  ```bash
285  KEYCLOAK_GUACAMOLE_CLIENT_SECRET=<secret> \
286  GUACAMOLE_PUBLIC_HOST=guacamole.<your-domain> \
287  ./scripts/configure-guacamole-keycloak-client.sh
288  ```
289
290  If Uptime Kuma is enabled, create or update its Keycloak client:
291
292  ```bash
293  KEYCLOAK_KUMA_CLIENT_SECRET=<secret> \
294  KUMA_INTERCEPT_HOSTNAME=kuma.ziti \
295  ./scripts/configure-uptime-kuma-keycloak-client.sh
296  ```
297
298  ## 7. Export OpenZiti Identities
299
300  Remote services cannot connect until their identity JSON files are exported from the platform host and
    installed on the correct machines.
301
302  Export identities with:
303
304  ```bash
305  ./scripts/export-ziti-identity.sh <identity-name> ./<identity-name>.json
306  ```
307
308  Common identities to export:
309
310  ```bash
311  ./scripts/export-ziti-identity.sh uptime-kuma-host ./uptime-kuma-host.json
312  ./scripts/export-ziti-identity.sh uptime-kuma-monitor ./uptime-kuma-monitor.json
313  ./scripts/export-ziti-identity.sh onprem-services-host ./onprem-services-host.json
314  ./scripts/export-ziti-identity.sh ot-dashboard-client ./ot-dashboard-client.json
315  ./scripts/export-ziti-identity.sh ot-dht11-host ./ot-dht11-host.json
316  ./scripts/export-ziti-identity.sh ot-fan-client ./ot-fan-client.json
317  ./scripts/export-ziti-identity.sh vps1-ssh-host ./vps1-ssh-host.json
318  ./scripts/export-ziti-identity.sh vps2-ssh-host ./vps2-ssh-host.json
319  ./scripts/export-ziti-identity.sh onprem-ssh-host ./onprem-ssh-host.json
320  ```
321
322  Transfer each JSON file to the host that needs it using your preferred secure copy method. Keep file
    permissions restrictive, for example `0600`.
323
```

```
324 ## 8. Deploy Uptime Kuma On The Monitoring Host
325
326 Copy the repository to the monitoring host and enter the service directory:
327
328 ```bash
329 cd <repo>/code/services/uptime-kuma
330 ```
331
332 Create the service environment file:
333
334 ```bash
335 cp .env.example .env
336 ```
337
338 Edit `.env` and replace:
339
340 ```text
341 KEYCLOAK_ISSUER=https://auth.<your-domain>/realms/ziti
342 KEYCLOAK_CLIENT_SECRET=<uptime-kuma-client-secret>
343 OAUTH2_PROXY_COOKIE_SECRET=<random-secret>
344 OPA_BUNDLE_BASE_URL=https://opa-bundles.<your-domain>
345 OPA_BUNDLE_AUTH_HEADER=<base64-opa-user-colon-password>
346 ```
347
348 Install the Uptime Kuma host identity:
349
350 ```bash
351 ./scripts/install-ziti-host-identity.sh /path/to/uptime-kuma-host.json
352 ```
353
354 Install the monitoring identity used by the OpenZiti proxy sidecars:
355
356 ```bash
357 mkdir -p identities
358 install -m 0600 /path/to/uptime-kuma-monitor.json identities/uptime-kuma-monitor.json
359 ```
360
361 Start the stack:
362
363 ```bash
364 ./scripts/deploy.sh
365 ```
366
367 Validate:
368
369 ```bash
370 docker compose ps
371 docker compose logs uptime-kuma-ziti-host uptime-kuma-direct-oauth2-proxy uptime-kuma-envoy uptime-kuma-opa
372 ```
373
374 The private Uptime Kuma UI is reached through OpenZiti at:
375
376 ```text
377 http://kuma.ziti
378 ```
379
380 ## 9. Deploy The On-Prem Services
381
382 Copy the repository to the on-prem host and enter the service directory:
383
384 ```bash
385 cd <repo>/code/services/onprem
386 ```
387
388 Create the environment file:
389
390 ```bash
391 cp .env.example .env
392 ```
393
394 Edit `.env` and replace:
395
```

```
396 ```text
397 KEYCLOAK_ISSUER=https://auth.<your-domain>/realms/ziti
398 KEYCLOAK_OT_DASHBOARD_CLIENT_SECRET=<ot-dashboard-client-secret>
399 OT_DASHBOARD_OAUTH2_PROXY_COOKIE_SECRET=<random-secret>
400 KEYCLOAK_ACCOUNTING_CLIENT_SECRET=<accounting-client-secret>
401 ACCOUNTING_OAUTH2_PROXY_COOKIE_SECRET=<random-secret>
402 OPA_BUNDLE_BASE_URL=https://opa-bundles.<your-domain>
403 OPA_BUNDLE_AUTH_HEADER=<base64-opa-user-colon-password>
404 ```
405
406 Install required OpenZiti identities:
407
408 ```bash
409 ./scripts/install-ziti-identity.sh onprem-services-host /path/to/onprem-services-host.json
410 ./scripts/install-ziti-identity.sh ot-dashboard-client /path/to/ot-dashboard-client.json
411 ```
412
413 Start the stack:
414
415 ```bash
416 ./scripts/deploy.sh
417 ```
418
419 Validate locally:
420
421 ```bash
422 docker compose ps
423 docker compose exec ot-dashboard python -c 'import urllib.request; print(urllib.request.urlopen("http
424 ://127.0.0.1:8080/api/readings").read().decode())'
425 ```
426
427 The private OpenZiti services exposed by this host are:
428
429 ```text
430 http://ot-dashboard.ziti
431 http://accounting.ziti
432 mqtt.ziti:1883
433 ```
434
435 ## 10. Deploy The DHT11 Raspberry Pi Service
436
437 Copy `services/ot-dht11` to the Raspberry Pi connected to the DHT11 sensor.
438
439 For a system-wide installation, run from that directory:
440
441 ```bash
442 ./scripts/install-modbus-service.sh
443 ./scripts/install-ziti-cli.sh
444 ./scripts/install-ziti-host-identity.sh /path/to/ot-dht11-host.json
445 ```
446
447 For a user-level installation, create or use a Python virtual environment first, then run:
448
449 ```bash
450 INSTALL_DIR="${HOME}/bin" ./scripts/install-ziti-cli.sh
451 ./scripts/install-user-services.sh
452 ./scripts/install-user-ziti-host-identity.sh /path/to/ot-dht11-host.json
453 ```
454
455 Enable lingering if the user services should start at boot:
456
457 ```bash
458 sudo loginctl enable-linger <pi-user>
459 ```
460
461 Validate:
462
463 ```bash
464 systemctl --user status ot-dht11-modbus.service
465 systemctl --user status ot-dht11-ziti-host.service
466 ss -ltnp | grep 1502
467 ```
```

```
467 Expected result: Modbus listens on `127.0.0.1:1502` and is hosted into OpenZiti as `ot-dht11-modbus`.
468
469
470 ## 11. Deploy The Fan Raspberry Pi Service
471
472 Copy `services/ot-fan` to the Raspberry Pi connected to the fan display.
473
474 Run from that directory:
475
476 ```bash
477 ./scripts/install-ziti-cli.sh
478 ./scripts/install-user-services.sh
479 ./scripts/install-user-ziti-identity.sh /path/to/ot-fan-client.json
480 ```
481
482 Enable lingering if the user services should start at boot:
483
484 ```bash
485 sudo loginctl enable-linger <pi-user>
486 ```
487
488 Validate:
489
490 ```bash
491 systemctl --user status ot-fan-ziti-proxy.service
492 systemctl --user status ot-fan-mqtt.service
493 ss -ltnp | grep ':11883 '
494 ```
495
496 The fan client dials the private `mqtt-broker` service through OpenZiti and uses these MQTT topics:
497
498 ```text
499 ot/fan/cmd
500 ot/fan/state
501 ```
502
503 ## 12. Optional SSH Services Through OpenZiti
504
505 The access matrix defines private SSH services for managed hosts. To enable them, export the matching host
506 identity and run the SSH host tunneler on each host.
507
508 Example for one host:
509
510 ```bash
511 docker compose \
512   -f services/ssh-hosts/docker-compose.yml \
513   --profile vps1 \
514   up -d
515 ```
516
517 Set the identity path through environment variables if the default path does not match your host:
518
519 ```text
520 VPS1_SSH_HOST_IDENTITY=/path/to/vps1-ssh-host.json
521 VPS2_SSH_HOST_IDENTITY=/path/to/vps2-ssh-host.json
522 ONPREM_SSH_HOST_IDENTITY=/path/to/onprem-ssh-host.json
523 ```
524
525 ## 13. Optional Wazuh Agents
526
527 If the Wazuh stack is enabled, install Wazuh agents on monitored hosts after the Wazuh manager is reachable.
528
529 For the local host:
530
531 ```bash
532 WAZUH_AGENT_MANAGER=<platform-hostname-or-ip> \
533 WAZUH_AGENT_NAME=<agent-name> \
534 ./scripts/setup-wazuh-host-agent.sh
535 ```
536
537 For remote hosts, either run the same setup locally on each host or adapt the remote deployment script with
538 your own SSH targets:
```

```
537 ```bash
538 ./scripts/deploy-wazuh-agent-remote.sh \
539   --host <user>@<host> \
540   --agent-name <agent-name> \
541   --manager <platform-hostname-or-ip>
542 ```
543
544 Do not run the remote script without reviewing its defaults first.
545
546 ## 14. Optional Uptime Kuma Monitor Provisioning
547
548 After the Uptime Kuma stack is running, update monitoring targets for your own domains in:
549
550 ```text
551 monitoring/uptime-kuma/monitors.yaml
552 ```
553
554 Install provisioning dependencies from the repository root on the host that can access the Uptime Kuma
555 Docker Compose project:
556
557 ```bash
558 python3 -m venv .venv-uptime-kuma
559 . .venv-uptime-kuma/bin/activate
560 pip install -r monitoring/uptime-kuma/requirements.txt
561 ```
562
563 Preview changes:
564
565 ```bash
566 python monitoring/uptime-kuma/provision.py --dry-run
567 ```
568
569 Apply changes:
570
571 ```bash
572 python monitoring/uptime-kuma/provision.py
573 ```
574
575 ## 15. Optional Perimeter-Based Comparison
576
577 The repository also contains WireGuard and perimeter-style Compose files for comparison with the Zero Trust
578 setup.
579
580 WireGuard entry point:
581
582 ```bash
583 docker compose -f compose/wireguard.yml up -d
584 ```
585
586 Perimeter intranet entry point:
587
588 ```bash
589 docker compose -f docker-compose.perimeter.yml up -d
590 ```
591
592 Remote perimeter comparison files are in:
593
594 ```text
595 remote-services/
596 ```
597
598 Before using them, replace all example domains, OAuth2 redirect URLs, Docker network names, and hostnames.
599
600 ## Validation Checklist
601
602 After deployment, verify each layer in order:
603
604 1. DNS resolves all public hostnames to the intended public IP addresses.
605 2. Caddy obtains certificates and serves the configured virtual hosts.
606 3. Keycloak realm is reachable at `https://auth.<your-domain>/realms/<realm>`.
607 4. OpenZiti edge API is reachable at `https://ziti.<your-domain>/edge/client/v1/version`.
```

```
607 5. `ziti-bootstrap` completed successfully.
608 6. OPA bundle server requires basic auth and serves generated bundles.
609 7. Remote identity JSON files exist on the correct hosts with restrictive permissions.
610 8. Remote service stacks show healthy containers with `docker compose ps`.
611 9. Private OpenZiti URLs such as `http://intranet.ziti`, `http://kuma.ziti`, `http://ot-dashboard.ziti`, and
    `http://accounting.ziti` work only for authorized identities.
612 10. Wazuh agents appear in the Wazuh dashboard if SIEM is enabled.
613 11. Uptime Kuma monitors report the expected private and public checks.
614
615 ## Local Development
616
617 For local experimentation without public DNS, use the local example environment:
618
619 ```bash
620 cp .env.example .env
621 ./scripts/access-matrix.py render-all
622 docker compose -f compose/zero-trust.yml --profile local up -d
623 ```
624
625 Local URLs:
626
627 ```text
628 Keycloak: http://localhost:8080
629 OpenZiti console: https://localhost:8443
630 Intranet through local Ziti proxy: http://localhost:8088
631 ```
```

Listing D.1: README der Infrastructure-as-Code-Umgebung

E. Playbook zur Erkundung der Laborumgebung

Dieses Playbook dient als praktische Anleitung, um die umgesetzte Laborumgebung aus Sicht der wichtigsten Benutzerrollen nachzuvollziehen. Es beschreibt nicht den technischen Aufbau der Infrastruktur, sondern die Bedienung der fertigen Umgebung. Ziel ist, dass eine lesende Person die zentralen Eigenschaften der Zero-Trust-Architektur selbst überprüfen kann: identitätsbasierte Dienstsichtbarkeit, Single Sign-on über Keycloak, rollenbasierte Autorisierung, private Anwendungszugriffe, Monitoring, OT-Interaktion sowie administrativer Zugriff.

Die Anleitung setzt voraus, dass die Infrastruktur gemäss README der Infrastructure-as-Code-Umgebung aufgebaut wurde (vgl. Anhang D). Die verwendeten Benutzernamen entsprechen den Identitäten aus der Access-Matrix. Passwörter werden bewusst nicht dokumentiert und sind durch die Betreiberin oder den Betreiber der Umgebung bereitzustellen.

E.1. Allgemeine Voraussetzungen

Für die Durchführung der folgenden Schritte wird ein Client-System mit Webbrowser benötigt. Für die Rollen `alice`, `bob` und `ops-admin` muss zusätzlich der OpenZiti Desktop Edge Client installiert und mit der Laborumgebung verbunden werden können. Für administrative Tests wird ausserdem ein SSH-Client benötigt.

Die folgenden Zugangsdaten werden im Playbook verwendet:

Tabelle E.1.: Benutzerrollen im Playbook

Benutzername	Passwort	Rolle im Playbook
<code>ops-admin</code>	<code><ops-admin-password></code>	Administrativer Benutzer für SSH-Zugriff auf die verwalteten Hosts.
<code>alice</code>	<code><alice-password></code>	Buchhaltungsbenedutzerin mit Zugriff auf Intranet und Accounting.
<code>bob</code>	<code><bob-password></code>	Technischer Benutzer mit Zugriff auf Intranet, Uptime Kuma und OT-Dashboard.
<code>service-technician</code>	<code><technician-password></code>	Service-Techniker mit Zugriff auf den browserbasierten Wartungszugang über Guacamole.

Für die Zero-Trust-Dienste werden die privaten OpenZiti-Namen aus Tabelle E.2 verwendet. Nicht autorisierte Dienste sollten im OpenZiti Desktop Edge Client nicht sichtbar sein oder nicht erreichbar sein.

Tabelle E.2.: Wichtige Dienstadressen im Playbook

Dienstadresse	Zweck
http://intranet.ziti	Intranet-Demoanwendung.
http://accounting.ziti	Accounting-Demoanwendung <i>PayYourPeople</i> .
http://kuma.ziti	Private Uptime-Kuma-Oberfläche.
http://ot-dashboard.ziti	OT-Dashboard mit Sensorwerten und Lüftersteuerung.
vps1-ssh.ziti:22	Privater SSH-Dienst des ersten Plattformhosts.
vps2-ssh.ziti:22	Privater SSH-Dienst des Monitoringhosts.
onprem-ssh.ziti:22	Privater SSH-Dienst des On-Prem-Hosts.
https://guacamole.prench-fress.xyz	Öffentlicher Guacamole-Zugang für den Service-Techniker.

E.2. OpenZiti Desktop Edge Client verwenden

Die Rollen `ops-admin`, `alice` und `bob` verwenden den OpenZiti Desktop Edge Client. Der Client stellt nach erfolgreicher Anmeldung private Dienste lokal bereit und zeigt nur jene Dienste an, für die die angemeldete Identität eine Dial-Policy besitzt.

1. OpenZiti Desktop Edge Client auf dem Client-System starten.
2. Eine neue Identität beziehungsweise den Keycloak-gestützten Login der Laborumgebung hinzufügen.
3. Im Browserfenster des Login-Flows mit dem vorgesehenen Benutzernamen anmelden.
4. Nach erfolgreicher Anmeldung prüfen, dass der Client verbunden ist.
5. Die angezeigten Dienste mit den erwarteten Diensten der jeweiligen Rolle vergleichen.

Abbildung E.1 zeigt die erwartete Benutzerperspektive im OpenZiti Desktop Edge Client. Entscheidend ist nicht nur, dass Dienste erreichbar sind, sondern auch, dass nicht autorisierte Dienste gar nicht in der Dienstsicht erscheinen.

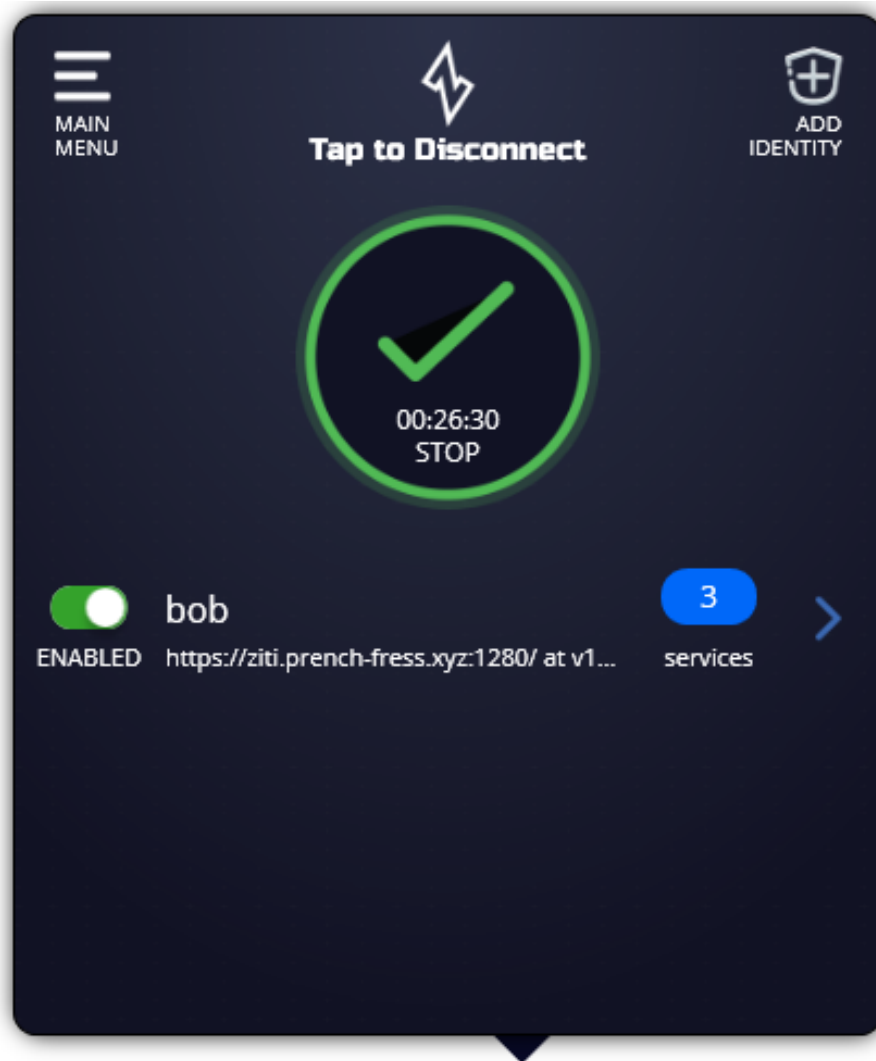


Abbildung E.1.: OpenZiti Desktop Edge Client mit sichtbaren privaten Diensten

E.3. Rolle ops-admin

E.3.1. Zweck der Rolle

Die Identität `ops-admin` ist für die Administration der Laborumgebung vorgesehen. Sie besitzt Zugriff auf die privaten SSH-Dienste der verwalteten Hosts. Diese Rolle demonstriert, dass auch Administrationszugänge als private, explizit freigegebene Dienste modelliert werden und nicht allgemein im Netzwerk erreichbar sein müssen.

E.3.2. Schritte

1. OpenZiti Desktop Edge Client starten.
2. Mit dem Benutzer `ops-admin` und dem Passwort `<ops-admin-password>` anmelden.
3. Prüfen, dass die privaten SSH-Dienste sichtbar sind: `vps1-ssh`, `vps2-ssh` und `onprem-ssh`.

4. Einen SSH-Client öffnen.
5. Eine Verbindung zum Plattformhost über OpenZiti herstellen:

```
ssh <admin-user>@vps1-ssh.ziti
```

Listing E.1: SSH-Zugriff auf den Plattformhost über OpenZiti

6. Auf dem Zielhost den Zustand der Docker-Dienste prüfen, beispielsweise mit `docker compose ps` im entsprechenden Projektverzeichnis.
7. Die Prüfung für `vps2-ssh.ziti` und `onprem-ssh.ziti` wiederholen, sofern diese Dienste in der Umgebung aktiv sind.

E.3.3. Erwartetes Ergebnis

Der Benutzer `ops-admin` kann die privaten SSH-Dienste erreichen und die Hosts administrieren. Geschäftsanwendungen wie `accounting.ziti`, `kuma.ziti` oder `ot-dashboard.ziti` gehören nicht automatisch zur Admin-Rolle. Damit wird sichtbar, dass administrative Berechtigungen und Fachanwendungsberechtigungen getrennt modelliert sind.

E.4. Rolle `alice`

E.4.1. Zweck der Rolle

`alice` repräsentiert eine Benutzerin aus der Buchhaltung. Sie benötigt Zugriff auf das Intranet und auf die Accounting-Anwendung, aber nicht auf Monitoring-, OT- oder Administrationsdienste.

E.4.2. Schritte

1. OpenZiti Desktop Edge Client starten.
2. Mit dem Benutzer `alice` und dem Passwort `<alice-password>` anmelden.
3. Prüfen, dass `intranet` und `accounting` als private Dienste verfügbar sind.
4. Im Browser `http://intranet.ziti` öffnen.
5. Falls Keycloak zur Anmeldung auffordert, erneut mit `alice` anmelden.
6. Prüfen, dass die Intranet-Seite angezeigt wird.

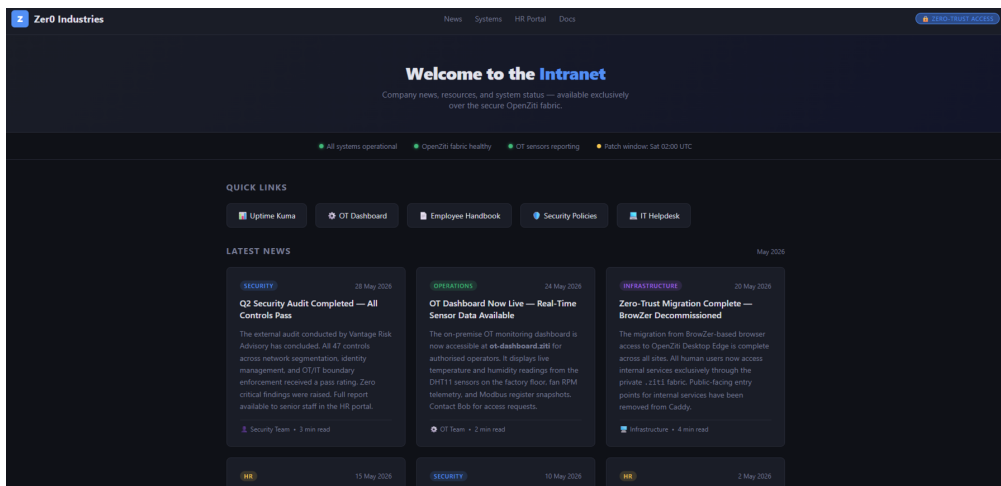


Abbildung E.2.: Intranet-Demoanwendung aus Benutzersicht

7. Im Browser <http://accounting.ziti> öffnen.
8. Prüfen, dass die Accounting-Anwendung *PayYourPeople* angezeigt wird.
9. Einen Testwert eingeben und die Oberfläche auslösen, um die Bedienbarkeit zu prüfen.

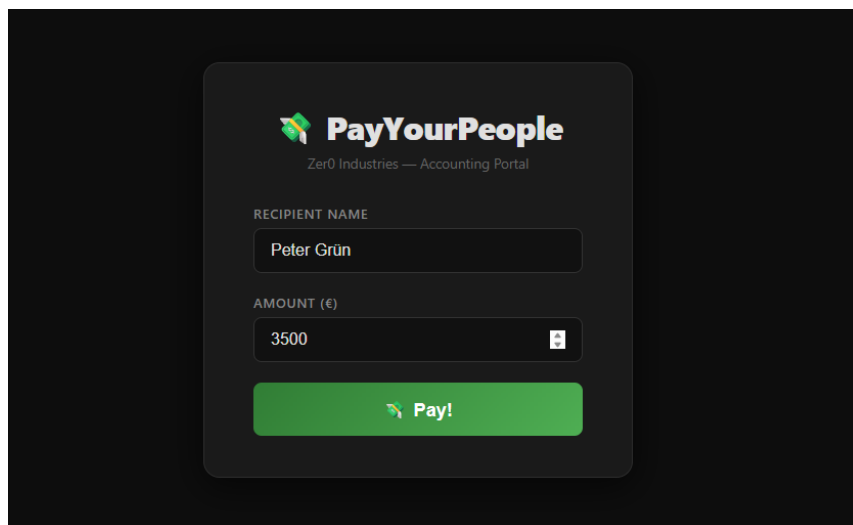


Abbildung E.3.: Accounting-Demoanwendung *PayYourPeople*

E.4.3. Negative Zugriffstests

1. <http://kuma.ziti> öffnen.
2. <http://ot-dashboard.ziti> öffnen.
3. Prüfen, ob SSH-Dienste, Guacamole, MQTT oder Modbus im OpenZiti Desktop Edge Client sichtbar sind.

E.4.4. Erwartetes Ergebnis

alice kann Intranet und Accounting nutzen. Uptime Kuma, OT-Dashboard, Guacamole, MQTT, Modbus und die SSH-Dienste sind für diese Rolle nicht vorgesehen und sollten nicht erreichbar sein. Damit wird die fachliche Trennung zwischen Buchhaltung und technischen Betriebsrollen sichtbar.

E.5. Rolle bob

E.5.1. Zweck der Rolle

bob repräsentiert einen technischen Benutzer beziehungsweise Maschinenbediener. Er benötigt Zugriff auf das Intranet, auf das Monitoring und auf das OT-Dashboard. Er erhält keinen Zugriff auf Accounting oder administrative SSH-Dienste.

E.5.2. Schritte

1. OpenZiti Desktop Edge Client starten.
2. Mit dem Benutzer bob und dem Passwort <bob-password> anmelden.
3. Prüfen, dass intranet, uptime-kuma-direct und ot-dashboard als private Dienste verfügbar sind.
4. Im Browser <http://intranet.ziti> öffnen und prüfen, dass die Intranet-Seite angezeigt wird.
5. Im Browser <http://kuma.ziti> öffnen.
6. Prüfen, dass die Uptime-Kuma-Oberfläche erreichbar ist und die Monitorübersicht angezeigt wird.

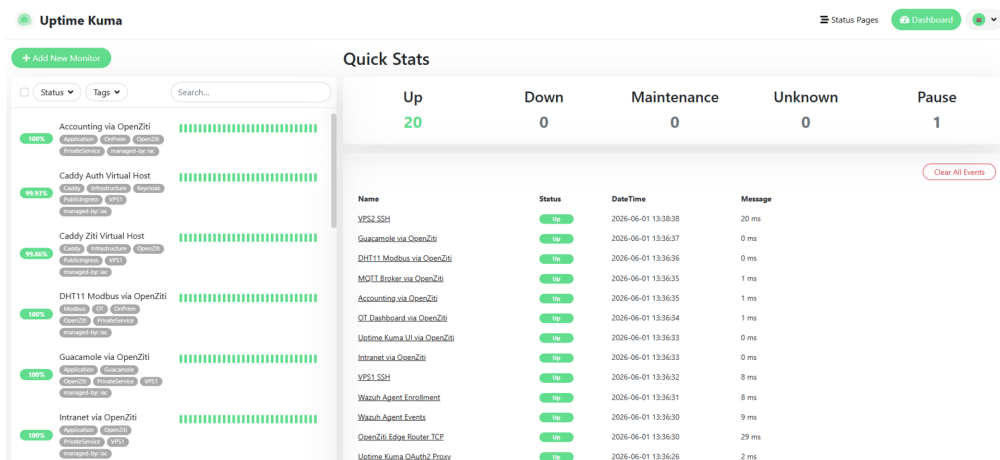


Abbildung E.4.: Uptime Kuma als privater Monitoring-Dienst

7. Im Browser <http://ot-dashboard.ziti> öffnen.
8. Prüfen, dass Temperatur, Luftfeuchtigkeit, Sensorstatus und Lüfterstatus angezeigt werden.

9. Die Schaltflächen *Stop*, *Low*, *Medium* und *Max* verwenden.
10. Den Slider für die Lüftergeschwindigkeit bewegen.
11. Beobachten, ob der Zielwert und der aktuelle Lüfterstatus aktualisiert werden.
12. Prüfen, ob die gesetzte beziehungsweise aktuelle Drehzahl auch auf der LED-Matrix des Lüfters sichtbar ist.



Abbildung E.5.: OT-Dashboard mit DHT11-Sensordaten und Lüftersteuerung

E.5.3. Negative Zugriffstests

1. <http://accounting.ziti> öffnen.
2. Prüfen, ob SSH-Dienste im OpenZiti Desktop Edge Client sichtbar sind.
3. Prüfen, ob Guacamole, MQTT oder der direkte DHT11-Modbus-Dienst als verfügbare Dienste erscheinen.

E.5.4. Erwartetes Ergebnis

bob kann Intranet, Uptime Kuma und OT-Dashboard verwenden. Im OT-Dashboard kann er Sensordaten betrachten und die Lüftersteuerung auslösen. Die Weboberfläche veröffentlicht dazu MQTT-Steuerbefehle; die physische LED-Matrix des Lüfters sollte die gewählte beziehungsweise aktuelle Drehzahl sichtbar anzeigen. Accounting, Guacamole, MQTT, Modbus und SSH-Dienste sind für diese Rolle nicht vorgesehen. Damit zeigt die Umgebung, dass technische Benutzer Monitoring- und OT-Funktionen nutzen können, ohne gleichzeitig Zugriff auf Finanz- oder Administrationsdienste zu erhalten.

E.6. Rolle service-technician

E.6.1. Zweck der Rolle

`service-technician` repräsentiert einen externen Service-Techniker. Diese Rolle verwendet im Playbook bewusst nicht den OpenZiti Desktop Edge Client. Der Zugriff erfolgt stattdessen über den browserbasierten Guacamole-Zugang. Damit wird ein kontrollierter Wartungszugang demonstriert, bei dem der Techniker nur über eine bereitgestellte Jump-Host- beziehungsweise Remote-Desktop-Sitzung arbeitet.

E.6.2. Schritte

1. Einen Browser öffnen.
2. Die URL <https://guacamole.prench-fress.xyz> aufrufen.
3. Im vorgeschalteten Login mit dem Benutzer `service-technician` anmelden. Als Passwort wird der Platzhalter `<technician-password>` verwendet.
4. Prüfen, dass die Guacamole-Oberfläche geladen wird.
5. Die konfigurierte Remote-Desktop- oder Jump-Host-Verbindung auswählen.
6. Prüfen, dass die Remote-Sitzung im Browser geöffnet wird.
7. Innerhalb der Sitzung die vorgesehenen Wartungswerkzeuge oder Zielsysteme verwenden.

E.6.3. Erwartetes Ergebnis

Der Benutzer `service-technician` erreicht die Wartungsumgebung über Guacamole. Der Zugriff erfolgt über einen zentral kontrollierten Browserpfad und nicht über eine direkte Netzwerkanbindung des Technikergeräts an das OpenZiti-Fabric. Dadurch kann die Umgebung Wartungszugriffe bereitstellen, ohne dem externen Client allgemeine Sichtbarkeit auf interne Dienste zu geben.

E.6.4. Negative Zugriffstests

1. Auf dem Technikergerät keinen OpenZiti Desktop Edge Client verwenden.
2. `http://accounting.ziti`, `http://kuma.ziti` oder private SSH-Dienste direkt vom Technikergerät aus aufrufen.
3. Prüfen, dass diese Dienste ohne passende private Verbindung nicht erreichbar sind.

E.7. Use Case: Malware-Test mit EICAR-Datei

Dieser Use Case ist rollenübergreifend gültig und prüft die dynamische Zugriffseinschränkung über Wazuh und die Wazuh-OPA-Bridge. Er kann mit einem beliebigen Benutzer durchgeführt werden, sofern der verwendete Windows-Client durch einen Wazuh-Agent überwacht wird und der Agent in der Umgebung eindeutig einem Benutzer zugeordnet ist.

Die EICAR-Datei ist eine standardisierte, harmlose Testdatei für Antiviren- und Sicherheitslösungen. Sie enthält keinen echten Schadcode, wird aber von Schutzsystemen wie eine Malware-Erkennung behandelt. Dadurch kann geprüft werden, ob Erkennung, Alarmierung und Reaktion funktionieren, ohne echte Schadsoftware einzusetzen.

E.7.1. Voraussetzungen

Vor der Durchführung müssen folgende Voraussetzungen erfüllt sein:

- Der Wazuh-Agent ist auf dem Windows-Client installiert, konfiguriert und beim Wazuh-Manager enrolled. Die Skripte zur Installation und Konfiguration liegen im Repository unter `code/scripts/`. Für Linux-Hosts fasst `code/scripts/setup-wazuh-host-agent.sh` die Installation, Konfiguration und Registrierung zusammen. Für Windows-Clients ist das entsprechende Windows-Agent-Skript aus demselben Skriptverzeichnis zu verwenden.
- Der Wazuh-Agentname des Windows-Clients ist einem Benutzer zugeordnet. Diese Zuordnung erfolgt in `code/platform/wazuh-opa-bridge/user-map.json`. Die Datei `code/platform/wazuh-opa-bridge/user-map.example.json` dient als Vorlage für das Mapping.
- Die Wazuh-OPA-Bridge läuft und kann Alerts vom Wazuh-Manager beziehungsweise vom Alert-Tailer empfangen.
- Der getestete Benutzer ist vor dem Test nicht bereits gesperrt.

Das Mapping ist entscheidend, weil der Wazuh-Alert zunächst nur den auslösenden Agent beziehungsweise Host beschreibt. Die Wazuh-OPA-Bridge verwendet `user-map.json`, um daraus die betroffene Benutzeridentität zu bestimmen und gezielt diese Identität zu sperren.

Abbildung E.6 zeigt die Agent-Übersicht im Wazuh-Dashboard. Vor dem EICAR-Test sollte hier geprüft werden, ob der Windows-Client als angebundener Agent sichtbar ist und Daten an den Wazuh-Manager liefert.

ID	Name	IP address	Group(s)	Operating system	Cluster node	Version	Status	Actions
001	ov-626e92	127.0.0.1	default	Debian GNU/Linux 13	manager	v4.13.1	active	[Icons]
002	demeter	192.168.2.201	default	Ubuntu 26.04 LTS	manager	v4.13.1	active	[Icons]
003	LAB2	192.168.1.177	default	Microsoft Windows 11 Pro 10.0.22631.6199	manager	v4.13.1	active	[Icons]

Abbildung E.6.: Wazuh-Dashboard mit Übersicht der angebotenen Agents

E.7.2. Schritte

1. Auf dem Windows-Client mit einem der Laborbenutzer anmelden, zum Beispiel `alice`, `bob` oder `service-technician`.
2. Einen erlaubten Zugriff testen, beispielsweise `http://intranet.ziti` für `alice` oder `bob`. Dadurch ist klar, dass der Benutzer vor dem Test grundsätzlich Zugriff besitzt.
3. Die EICAR-Testdatei auf dem Windows-Client herunterladen oder erstellen. Dafür kann die offizielle EICAR-Testzeichenfolge verwendet werden.
4. Warten, bis der Wazuh-Agent den Fund meldet und der Alert im Wazuh-Manager verarbeitet wurde.
5. Prüfen, ob die Wazuh-OPA-Bridge den Alert verarbeitet und den über `user-map.json` gemappten Benutzer sperrt.
6. Den zuvor funktionierenden Zugriff erneut versuchen.

E.7.3. Erwartetes Ergebnis

Der Wazuh-Agent erkennt die EICAR-Datei als Test-Malware und erzeugt einen Alert. Sofern der Wazuh-Agent korrekt installiert, konfiguriert und einem Benutzer zugeordnet ist, verarbeitet die Wazuh-OPA-Bridge den Alert und sperrt die gemappte Benutzeridentität. Danach kann der betroffene Benutzer nicht mehr auf die zuvor erlaubten Zero-Trust-Dienste zugreifen. Andere Benutzer sollten nicht betroffen sein.

Die Sperre demonstriert die kontinuierliche Verifikation der Umgebung: Zugriff wird nicht nur beim Login und anhand statischer Rollen entschieden, sondern kann durch Sicherheitsereignisse nachträglich eingeschränkt werden.

E.7.4. Entsperrn nach dem Test

Nach dem Test können gesperrte Benutzer und Server über das Skript `code/scripts/unblock-all.sh` wieder freigegeben werden. Das Skript leert die OPA-Blocklisten und aktiviert die betroffenen OpenZiti-Identitäten wieder.

```
./scripts/unblock-all.sh
```

Listing E.2: Alle Benutzer und Server nach einem Test wieder entsperren

Falls das Skript nicht auf dem Plattformhost ausgeführt wird, muss der passende SSH-Zielhost als Argument übergeben werden.

E.8. Zusammenfassende Feature-Tests

Tabelle E.3 fasst die wichtigsten Merkmale zusammen, die mit dem Playbook geprüft werden.

Tabelle E.3.: Feature-Tests der Laborumgebung

Merkmal	Prüfung
Dienstsichtbarkeit pro Identität	OpenZiti Desktop Edge zeigt je nach Benutzer nur die erlaubten Dienste an.
Single Sign-on	Webdienste leiten bei Bedarf zur Keycloak-Anmeldung weiter.
Rollenbasierte Autorisierung	Nicht vorgesehene Dienste sind nicht sichtbar oder nicht erreichbar.
Intranet und Accounting	alice kann beide Fachanwendungen nutzen.
Monitoring	bob kann Uptime Kuma über kuma.ziti öffnen.
OT-Dashboard	bob sieht DHT11-Werte und kann Lüfterbefehle auslösen.
Administrativer Zugriff	ops-admin erreicht SSH-Dienste über private OpenZiti-Adressen.
Wartungszugang	service-technician nutzt Guacamole als browserbasierten Jump-Host-Zugang.
Dynamische Zugriffssperre	Eine EICAR-Datei auf einem überwachten Windows-Client löst einen Wazuh-Alert aus; die Wazuh-OPA-Bridge sperrt den gemapten Benutzer.

E.9. Troubleshooting

Wenn ein Schritt nicht funktioniert, sollten die folgenden Punkte geprüft werden:

- **OpenZiti Desktop Edge ist nicht verbunden:** Prüfen, ob der Client angemeldet ist und die Keycloak-Anmeldung erfolgreich abgeschlossen wurde.
- **Ein Dienst ist nicht sichtbar:** Prüfen, ob der angemeldete Benutzer laut Access-Matrix eine Dial-Policy für diesen Dienst besitzt.
- **Browser zeigt eine Login-Seite:** Mit dem gleichen Benutzer über Keycloak anmelden und prüfen, ob der Benutzer die passende Rolle besitzt.
- **Browser zeigt 401 oder 403:** Die Netzwerkverbindung kann vorhanden sein, die Anwendungsebene verweigert aber den Zugriff über oauth2-proxy, Envoy oder OPA.
- **OT-Dashboard zeigt keine Sensordaten:** Prüfen, ob der DHT11-Modbus-Dienst läuft und die Identität ot-dashboard-client den Dienst ot-dht11-modbus erreichen kann.
- **Lüfterzustand aktualisiert nicht:** Prüfen, ob Mosquitto läuft und ob die Topics ot/fan/cmd und ot/fan/state verwendet werden.
- **Uptime Kuma zeigt Monitore als down:** Prüfen, ob uptime-kuma-monitor installiert ist und die Proxy-Sidecars laufen.
- **Guacamole öffnet keine Sitzung:** Prüfen, ob der Guacamole-Stack läuft, der Keycloak-Client korrekt konfiguriert ist und die Zielverbindung innerhalb von Guacamole vorhanden ist.
- **SSH ist nicht erreichbar:** Prüfen, ob die passende SSH-Host-Identity installiert ist und der lokale SSH-Dienst auf dem Zielhost läuft.
- **EICAR-Test erzeugt keine Sperre:** Prüfen, ob der Wazuh-Agent installiert, gestartet und beim Wazuh-Manager registriert ist. Zusätzlich prüfen, ob der Agentname in code/platform/wazuh-opa-bridge/user-map.json dem richtigen Benutzer zugeordnet ist.

- **Benutzer bleibt nach dem Test gesperrt:** `code/scripts/unblock-all.sh` ausführen und danach den Zugriff erneut testen.

Dieses Playbook zeigt die Laborumgebung aus Benutzersicht. Die technischen Artefakte, mit denen diese Zugriffe erzeugt werden, sind in der README der Infrastructure-as-Code-Umgebung und in der Policy-Matrix dokumentiert.

Abbildungsverzeichnis

1.	Physischer Laboraufbau mit den Client-Arbeitsplätzen der Benutzerrollen Alice und Bob sowie der Lüfter-Simulation auf einer LED-Matrix	iii
2.	Vereinfachter Zugriffspfad der Zero-Trust-Architektur: Eine Anfrage durchläuft den privaten Netzwerkzugriff (OpenZiti), die Benutzeranmeldung (Keycloak via oauth2-proxy) und den Richtlinienentscheid (OPA), bevor sie die Anwendung erreicht	iii
3.	OT-Dashboard der Laborumgebung zur Anzeige der Sensordaten (Temperatur, Luftfeuchtigkeit) und zur Steuerung der Lüfter-Simulation	iv
4.	OpenZiti Desktop Edge Client aus Benutzersicht: Sichtbar sind ausschliesslich die der angemeldeten Identität explizit freigegebenen privaten Dienste	v
1.1.	Ablauf einer Zugriffsanfrage im NIST Architekturmodell	13
2.1.	Physischer Laboraufbau mit den Client-Arbeitsplätzen von Alice und Bob sowie der Lüfter-Simulation	17
2.2.	Komponentenübersicht des eingesetzten Technologie-Stacks	19
2.3.	Benutzeroberflächen der IT-Dienste in der Laborumgebung	24
2.4.	OT-Dashboard zur Anzeige von Sensordaten und Lüfter-Zustand	25
2.5.	Übersicht der implementierten Zero-Trust-Architektur	28
2.6.	Vereinfachter Zugriffspfad der implementierten Zero-Trust-Architektur (Schritt 4 bezeichnet die Autorisierungsanfrage von Envoy an OPA)	29
2.7.	Browserbasierter Wartungszugriff über Guacamole und OpenZiti	29
2.8.	Zusammenspiel von Keycloak-Identitäten und OpenZiti-Service-Policies	30
2.9.	OpenZiti Desktop Edge Client mit verfügbaren privaten Diensten	31
2.10.	Trennung von Policy Enforcement, Policy Decision und Policy-Daten	33
2.11.	Dynamische Zugriffseinschränkung über Wazuh, OPA und OpenZiti	34
2.12.	Einbindung privater On-Prem- und OT-Dienste über OpenZiti	34
2.13.	Übersicht der implementierten perimeterbasierten Architektur	36
3.1.	Eingesetzter DHT11-Temperatur- und Luftfeuchtigkeitssensor	40
3.2.	LED-Matrix zur Simulation der Lüfter-Komponente	41
A.1.	3x3-Risikomatrix mit vier identifizierten Risiken	81
A.2.	Aktualisierte Risikomatrix zum Projektabschluss	82
E.1.	OpenZiti Desktop Edge Client mit sichtbaren privaten Diensten	115
E.2.	Intranet-Demoanwendung aus Benutzersicht	117
E.3.	Accounting-Demoanwendung <i>PayYourPeople</i>	117
E.4.	Uptime Kuma als privater Monitoring-Dienst	118
E.5.	OT-Dashboard mit DHT11-Sensordaten und Lüftersteuerung	119
E.6.	Wazuh-Dashboard mit Übersicht der angebundenen Agents	121

Tabellenverzeichnis

2.1. Zuordnung der Zero-Trust-Rollen zu den eingesetzten Technologien	20
2.2. Auszug zentraler Identitäten der Laborumgebung	23
2.3. IT-Dienste der gemeinsamen Laborumgebung	23
2.4. OT-Dienste der gemeinsamen Laborumgebung	25
3.1. Registermodell des DHT11-Modbus-Dienstes	40
3.2. Durchführungsschritte Szenario A1	49
3.3. Durchführungsschritte Szenario A2	50
3.4. Durchführungsschritte Szenario A3	51
3.5. Durchführungsschritte Szenario A4	52
3.6. Durchführungsschritte Szenario B1	53
3.7. Durchführungsschritte Szenario B2	53
3.8. Durchführungsschritte Szenario B3	54
3.9. Übersicht der Hypothesenbewertungen	55
A.1. Zeitplan des Projekts anhand der Meilensteine	79
C.1. Protokollfelder für A1 in der Zero-Trust-Architektur	87
C.2. Protokollfelder für A1 in der perimeterbasierten Architektur	88
C.3. Protokollfelder für A2 in der perimeterbasierten Architektur	89
C.4. Protokollfelder für A3 in der perimeterbasierten Architektur	90
C.5. Protokollfelder für A4 in der perimeterbasierten Architektur	91
C.6. Protokollfelder für B1 in der perimeterbasierten Architektur	92
C.7. Protokollfelder für B2 in der perimeterbasierten Architektur	93
C.8. Protokollfelder für B3 in der perimeterbasierten Architektur	94
C.9. Protokollfelder für A2 in der Zero-Trust-Architektur	96
C.10. Protokollfelder für A3 in der Zero-Trust-Architektur	98
C.11. Praktisches Protokoll für B1 in der Zero-Trust-Architektur	99
C.12. Praktisches Protokoll für B2 in der Zero-Trust-Architektur	100
C.13. Praktisches Protokoll für B3 in der Zero-Trust-Architektur	101
C.14. Protokollfelder für A4 in der Zero-Trust-Architektur	102
E.1. Benutzerrollen im Playbook	113
E.2. Wichtige Dienstadressen im Playbook	114
E.3. Feature-Tests der Laborumgebung	123