HSLU Hochschule Luzern

Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte (DIVA.EXCHANGE)

Bachelorarbeit

Philipp Hauswirth

03. Januar 2024

Bachelorarbeit an der Hochschule Luzern – Informatik

Titel:	Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte („DIVA.EXCHANGE“)
Studentin/Student:	Philipp Hauswirth
Studiengang:	BSc Information & Cyber Security
Jahr:	2023
Betreuungsperson:	Dr. Dieter Arnold
Expertin/Experte:	Max Suter
Auftraggeberin/Auftraggeber:	Konrad Bächler

Codierung / Klassifizierung der Arbeit:
☒ Öffentlich (Normalfall)
☐ Vertraulich

Eidesstattliche Erklärung
Ich erkläre hiermit, dass ich/wir die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt haben, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben haben, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht haben, das Vertraulichkeitsinteresse des Auftraggebers wahren und die Urheberrechtsbestimmungen der Hochschule Luzern respektieren werde.

Ort / Datum, Unterschrift _____

Abgabe der Arbeit auf der Portfolio Datenbank
Bestätigungsvisum Studentin/Student
Ich bestätige, dass ich die Bachelorarbeit korrekt gemäss Merkblatt auf der Portfolio Datenbank ablege. Die Verantwortlichkeit sowie die Berechtigungen gebe ich ab, so dass ich keine Änderungen mehr vornehmen oder weitere Dateien hochladen kann.

Ort / Datum, Unterschrift _____

Verdankung

An dieser Stelle möchte ich die Gelegenheit nutzen, mich bei den Personen zu bedanken, welche massgeblich zum Gelingen meiner Bachelorarbeit beigetragen haben.

Ein besonderer Dank gilt Dieter Arnold, der meine Bachelorarbeit betreut und begutachtet hat, für seine Unterstützung, wertvollen Ratschläge und seine konstruktive Kritik. Seine engagierte Betreuung haben meine Arbeit in entscheidendem Masse geprägt.

Herzlichen Dank auch an Konrad Bächler von “DIVA.EXCHANGE”, für das Vertrauen in meine Fähigkeiten und die Möglichkeit, an diesem spannenden Projekt zu arbeiten. Seine fachliche Expertise, klaren Zielvorgaben und offene Kommunikation haben die Grundlage für eine erfolgreiche Zusammenarbeit geschaffen.

Ein weiterer Dank gebührt Max Suter, der mir wertvolle Inputs bei der Zwischenpräsentation gegeben hat. Seine Fragen und Anregungen während der Zwischenpräsentation haben mir geholfen, mögliche Hindernisse frühzeitig zu erkennen.

Zusammenfassung

Die vorliegende Arbeit knüpft an die Bachelorarbeit von Brian Boss und Marco Purtschert an, welche die De-Anonymisierung von Teilnehmern im I2P-basierten Handelsnetzwerk “DIVA.EXCHANGE” untersucht hat. Der Fokus liegt auf der praktischen Umsetzung der in ihrer Arbeit entwickelten theoretischen Angriffe. Der gemeinnützige Verein “DIVA.EXCHANGE” setzt sich für barrierefreie und kollaborative Ansätze ein, um eine dezentrale Plattform für das Finanzwesen zu schaffen.

Ziel dieser Arbeit ist es, die De-Anonymisierungsmöglichkeiten der Netzwerkteilnehmer aufzuzeigen. Der Hauptfokus liegt dabei auf dem Aufbau einer Laborumgebung in Form eines I2P-Testnetzwerks, um die Durchführbarkeit von De-Anonymisierungs-Attacken zu demonstrieren. Hierzu werden verschiedene bestehende I2P-Testnetzwerke untersucht, und eine Automatisierung für das Bereitstellen eines I2P-Testnetzwerks in Kubernetes entwickelt. Dabei werden diverse Herausforderungen und deren Lösungen beschrieben, die bei der Entwicklung aufgetreten sind.

In einem zweiten Schritt werden bestehende Angriffsideen in der Laborumgebung umgesetzt und verifiziert. Die Arbeit fokussiert sich auf die Durchführung und Verifizierung von Angriffen in einer kontrollierten Umgebung. Es werden keine neuen Angriffsmethoden entwickelt, und es erfolgt keine detaillierte Analyse von Angriffen auf spezifische I2P-Implementationen.

Die Entwicklung einer voll funktionsfähigen Laborumgebung und die gewonnenen Erkenntnisse stellen die Grundlage für weitere Forschung im Zusammenhang mit I2P und deren Angriffe dar.

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

“DIVA.EXCHANGE” ist ein gemeinnütziger Verein in Baar, welcher auf barrierefreie und kollaborative Ansätze setzt, um quelloffene Banking-Technologie zu entwickeln. Ihr Ziel ist es, eine Blockchain-basierte, vollständig dezentrale Plattform für das Finanzwesen der Zukunft zu schaffen, die die Privatsphäre aller Nutzer schützt. Durch die transparent Kommunikation und der Erstellung von quelloffener Software kann jeder sich an diesem Projekt beteiligen. Zurzeit arbeitet «DIVA.EXCHANGE» mit der Open-Source-Gemeinschaft an einer Banking-Technologie («DIVA»), welches für die Kommunikation auf I2P aufbaut. Ausserdem unterstützt «DIVA.EXCHANGE» wissenschaftliche Forschungsarbeiten. Unter anderem wurden mit der Hochschule Luzern bereits Forschungsarbeiten durchgeführt. (*Freie Banking-Technologie für alle*, 2020).

Diese Arbeit soll eine Weiterführung der gleichnamigen Bachelorarbeit «Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte („DIVA.EXCHANGE“)» von Brian Boss und Marco Purtschert sein. In ihrer Arbeit wurden theoretische Angriffe entwickelt und beschrieben, aber durch das Fehlen einer Laborumgebung bzw. eines Testnetzwerk wurden diese nie praktisch durchgeführt (Boss & Purtschert, 2022).

1.2 Problemstellung

Die von «DIVA.EXCHANGE» entwickelte Software «DIVA» basiert auf dem Anonymisierungsnetzwerk I2P. I2P steht für das “Invisble Internt Project” und ist eine vollständig verschlüsselte Netzwerkschicht welches die Aktivitäten und den Standort der Nutzer schützt (*I2P Anonymous Network*, o. J.). Es soll aufgezeigt werden, wie die Teilnehmer des Netzwerkes de-anonymisiert werden können. De-Anonymisierung ist definiert, als Zuordnung eines LeaseSets zu einer RouterInfo. Die Zuordnung ermöglicht es ein Ziel wie z.B. eine I2P-Webseite einem I2P-Router zuzuordnen. Dies wiederum ermöglicht den Standort und möglicherweise den Benutzer oder Betreiber ausfindig zu machen.

1.3 Fragestellung

Aufgrund der gegebenen Ausgangslage und der Problemstellung wurde die nachfolgende Fragestellung zusammen mit den entsprechenden Unterfragen abgeleitet.

1. Kann ein unabhängiges I2P Testnetzwerk automatisch und reproduzierbar aufgebaut werden?

- Gibt es bestehende Projekte für den automatischen Aufbau von einem unabhängigen I2P Netzwerk?
- Welche Komponenten werden dazu benötigt?
- Welcher I2P-Client eignet sich dazu am besten?

2. Inwiefern könnten theoretisch beschriebene Angriffe, wie sie in anderen Arbeiten dargestellt wurden, tatsächlich durchgeführt werden?

- Was sind die Voraussetzungen für diese Angriffe?
- Welches spezifische Zugriffslevel (z. B. Netzwerk- oder Applikationszugriff) wäre erforderlich, damit ein potenzieller Angreifer die theoretischen Angriffe umsetzen kann?
- Wie kann das Resultat eines Angriffs validiert werden?

1.4 Ziel der Arbeit

Das Ziel der Arbeit ist es eine Laborumgebung in Form eines I2P-Testnetzwerkes aufzubauen, um die Durchführbarkeit von De-Anonymisierungs-Attacken aufzuzeigen. Dazu soll auf der bestehenden theoretischen Arbeiten der HSLU und «DIVA.EXCHANGE» aufgebaut werden. Die Grundlage dafür stellt die gleichnamige Bachelorarbeit «Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte („DIVA.EXCHANGE“)» von Brian Boss und Marco Purtschert. Mit dem I2P-Testnetzwerk sollen Angriffe in einem kontrollierten Umfeld praktisch durchgeführt, getestet und validiert werden.

1.5 Abgrenzung

Diese Arbeit konzentriert sich darauf, bestehende Ansätze und Projekte für den automatischen Aufbau eines allgemeinen I2P-Testnetzwerks zu untersuchen und die Durchführbarkeit theoretisch beschriebener Angriffe im Kontext von I2P zu bewerten. Es werden keine neuen Angriffsmethoden entwickelt, und es erfolgt keine detaillierte Analyse von Angriffen auf bestimmte I2P-Implementierungen (I2P-Router, I2P-Applikationen, etc.).

2 Stand der Technik

2.1 Invisible Internet Project (I2P)

Abbildung 1: I2P Logo (geti2p.net)

Das Invisible Internet Project (I2P) ist ein Open-Source, dezentrales und anonymes Peer-to-Peer-Netzwerk, das eine sichere und anonyme Kommunikation über das Internet ermöglicht. Es verwendet mehrere Schichten von Verschlüsselung und Routing, um die Identitäten und den Datenverkehr seiner Benutzer zu verschleiern und so ein höheres Mass an Privatsphäre zu gewährleisten. Als Overlay-Netzwerk funktioniert I2P über dem regulären Internet und bietet hauptsächlich Dienste wie Web-Browsing, Messaging und File-Sharing («DevX Glossray: I2P», o. J.). Über den eigenen Transport-Layer ist es in der Lage, herkömmliche TCP/IP-Anwendungen über I2P zu streamen. Dies ermöglicht die Kommunikation von diversen TCP und UDP basierenden Applikationen und Protokolle (*I2P Anonymous Network*, o. J.).

I2P verbirgt die Quell- und Zielsysteme voneinander. Das bedeutet, dass z.B. ein Websitebetreiber nicht weiss woher seine Benutzer kommen und die Benutzer nicht wissen wo die Website gehostet wird. Niemand kann sehen, woher der Datenverkehr kommt, wohin er geht, oder was der Inhalt ist. Des Weiteren findet der gesamte I2P-Verkehr innerhalb des I2P-Netzwerks statt und interagiert nicht direkt mit dem Internet [1]. I2P ist lediglich eine Schicht, die über dem Internet liegt. Da sich das Netzwerk auf die Peers verlässt, um den Datenverkehr zu leiten und transportierten, wird das Geoblocking oder das Filtern von Internetverkehr von Regierungen oder Internet-Service-Provider (ISP) erschwert (*I2P Anonymous Network*, o. J.).

[1] Das Leiten von Datenverkehr zum Clearnet ist nicht Teil der I2P-Architektur und das I2P-Projekt betreibt auch keine Proxys in das Clearnet. Jedoch gibt es Drittanbieter wie *StormyCloud Inc.*, welche diese Funktionalität anbieten (*Web Browser Configuration - I2P*, o. J.).

2.2 I2P Protocol Stack

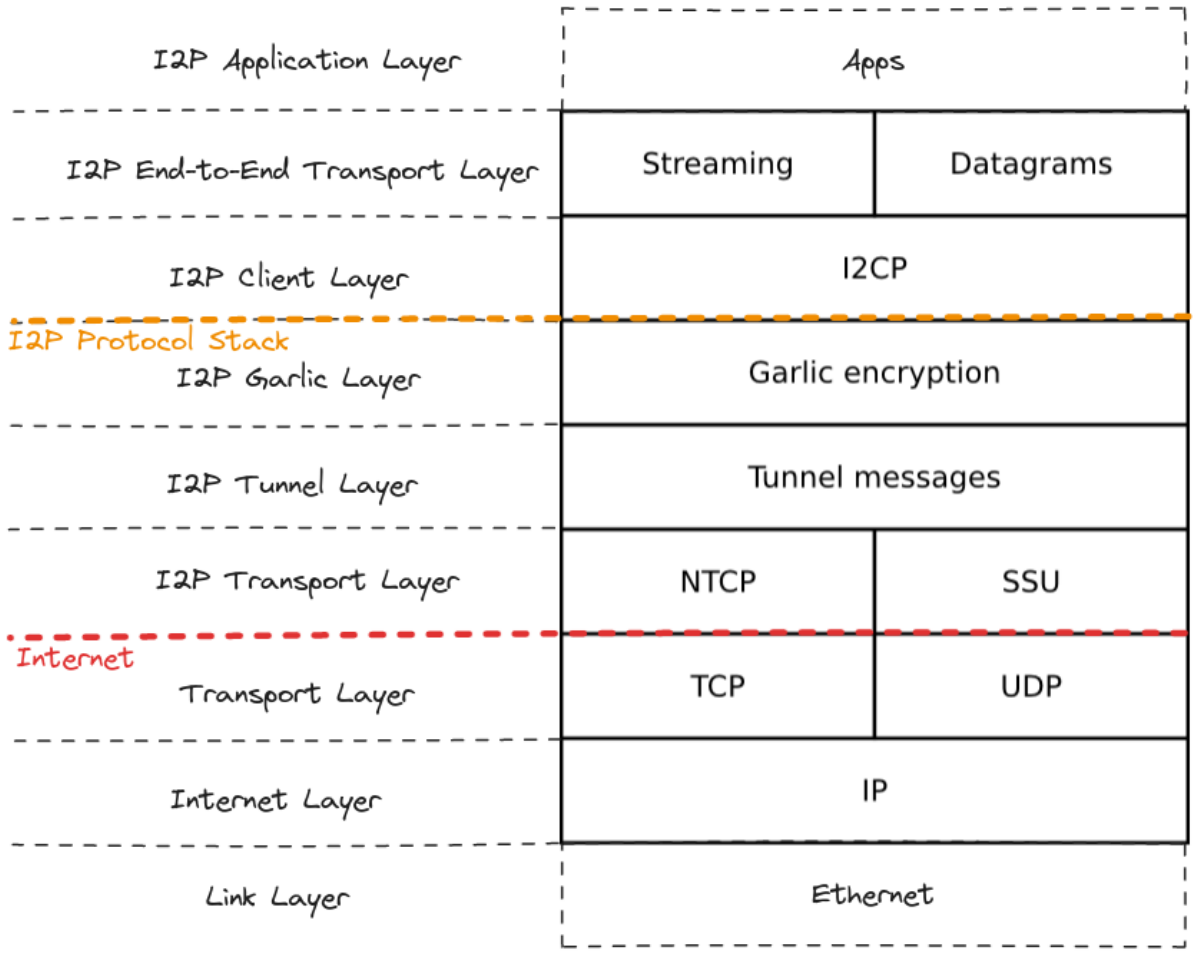

Abbildung 2: I2P Netzwerkschichten

I2P ist in mehreren Schichten aufgebaut. Jede Schicht des Protokollstapels hat seine eigene Funktion. In der oberen Abbildung der Protokollstapel mit den einzelnene Schichten skizziert. Im Anschluss werden die relevanten Schichten von unaufgelistet, beginnend mit der untersten Ebene des Protokollstapels.

2.2.1 Internet Layer & Transport Layer

Zu den untersten Schichten im Protokollstapel (*I2P Protocoll Stack*) sind der *Internet Layer* (IP) und *Transport Layer* (TCP, UDP). Beide Schichten sind Teil des normalen Internets und weit verbreitet. Aus diesem Grund werden diese nur sehr kurz erklärt.

Der *Internet Layer*, auch *Network Layer* (im OSI-Modell) genannt, ist eine entscheidende Komponente

des TCP/IP-Modells, die die Kommunikation zwischen Geräten in verschiedenen Netzen ermöglicht. Sie ist für die logische Adressierung zuständig und verwendet unter anderen das Internet-Protokoll (IP), um Geräten eindeutige Adressen zur Identifizierung und Weiterleitung zuzuweisen. IP-Adressen ermöglichen es der Internet-Schicht, Datenpakete über zusammengeschaltete Netze von der Quelle zum Ziel zu leiten. Ausserdem sorgt der *Internet Layer* für die Fragmentierung und Wiederzusammensetzung von Datenpaketen, um eine effiziente Übertragung zu gewährleisten. Zu den bekannten Protokollen, die auf dieser Schicht arbeiten, gehören das Internet-Protokoll (IPv4 oder IPv6) und das *Internet Control Message Protocol* (ICMP) (*TCP/IP Model - GeeksforGeeks*, o. J.). Der *Internet Layer* ist auf den *Link Layer* (auch *Data Link Layer* genannt) angewiesen, damit dieser ordnungsgemäss funktioniert. Der *Link Layer* ist die Schicht unter dem *Internet Layer* im TCP/IP-Modell. Dieser fällt ausserhalb des Themenbereichs und wurde ausgelassen.

Die Protokolle in dem *Transport Layer* stellen End-to-End Kommunikationsdienste für Anwendungen bereit. Zu den bekannten Protokollen, die auf dieser Schicht arbeiten, gehören das *Tansmission Control Protocol* (TCP) und das *User Datagram Protocol* (UDP). TCP ist ein verbindungsorientiertes Protokoll, das eine zuverlässige und geordnete Datenübertragung gewährleistet. Es stellt eine Verbindung her, bevor es Daten austauscht, und bietet eine Fehlerprüfung, eine erneute Übertragung verlorener Pakete und eine Flusskontrolle, um eine effiziente Datenübertragung zu gewährleisten. UDP ist ein verbindungsloses Protokoll, wodurch es im Vergleich zu TCP schneller ist und eine geringere Latenz aufweist. Es garantiert weder die Zustellung noch die Reihenfolge der Pakete, sodass es ein “Best-Effort”-Protokoll ist.

2.2.2 I2P Transport Layer

Die erste I2P spezifische Schicht ist der *I2P Transport Layer* mit den Protokollen NTCP und SSU. Der *I2P Transport Layer* bietet verschlüsselte Verbindungen zwischen zwei I2P-Routern. Diese Verbindungen sind noch nicht anonym und es handelt sich ausschliesslich um eine Hop-to-Hop-Verbindung. Zu den Protokollen im *I2P Transport Layer* gehören NTCP und SSU. NTCP baut auf TCP auf, während SSU UDP verwendet (*Protocol Stack - I2P*, o. J.). Das I2P-Projekt hat für NTCP und SSU mittlerweile eine neue Spezifikation definiert (NTCP2 und SSU2), welche die Alten ablöst[2] (*NTCP (NIO-based TCP) - I2P*, o. J.) (*SSU2 - I2P*, o. J.). Aus Gründen der Vereinfachung und Lesbarkeit wird im Text auf die allgemeine Bezeichnung von NTCP oder SSU verwiesen und keine spezifischen Versionen ausdrücklich genannt.

NTCP (*NIO-based TCP*) ist ein authentifiziertes Schlüsselvereinbarungsprotokoll, welches auf dem Noise Protocol basiert. NTCP2 ist ausschliesslich für den Punkt-zu-Punkt-Transport (Router-zu-Router) von I2NP Messages definiert. Es handelt sich nicht um eine allgemeine Datenleitung (*NTCP 2 - I2P*, o. J.).

[2] Welche Protokollversionen unterstützt sind, hängt stark vom I2P-Client ab. NTCP wurde grundsätzlich durch NTCP2 ersetzt und wird nicht mehr unterstützt (*NTCP (NIO-based TCP) - I2P*, o. J.). Im Gegensatz dazu unterstützt i2pd beispielsweise immer noch SSU1 (stand Dezember 2023), während bei i2p.i2p SSU1 standardmässig deaktiviert ist (*SSU2 - I2P*, o. J.).

Wie der NTCP-Transport bietet SSU einen verschlüsselten, verbindungsorientierten Punkt-zu-Punkt-Datentransport. SSU (*Secure Semireliable UDP*) wird als "semireliable" bezeichnet, weil es unbestätigte Nachrichten wiederholt sendet, aber nur bis zu einer bestimmten Anzahl von Malen. Danach wird die Nachricht verworfen. Das UDP basierte SSU Protkoll bietet im Gegensatz zu NTCP IP-Erkennung und UDP-NAT-Traversal-Dienste (*Secure Semireliable UDP (SSU) - I2P*, o. J.). UDP-NAT-Traversal, allgemein bekannt als *UDP Hole Punching*, ist eine Technik, die verwendet wird, um die Kommunikation zwischen zwei Geräten herzustellen, die sich hinter *Network Address Translation* (NAT) Routern befinden. Das NAT verhindert normalerweise die direkte Kommunikation zwischen Geräten in einem privaten Netz und Geräten im Internet. Durch das sogenannte *UDP Hole Punching* können diese Geräte miteinander kommunizieren, indem sie die Art und Weise ausnutzen, wie NAT-Router ausgehenden und eingehenden UDP-Verkehr behandeln. Dies ist vor allem sehr nützlich, wenn beispielsweise der I2P-Router in einem Heimnetzwerk steht, welches sich hinter einem NAT befindet.

Der *I2P Transport Layer* ist verschlüsselt (siehe Kapitel Tunnel Encryption).

2.2.3 I2P Tunnel Layer

Der *I2P Tunnel Layer* bietet vollständig verschlüsselte Tunnelverbindungen von Inbound- und Outbound-Tunnel (genauere Informationen sind dazu im Kapitel Tunnel beschrieben). Im *I2P Tunnel Layer* werden sogenannte Tunnel Messages transportiert. (*Protocol Stack - I2P*, o. J.)

Der *I2P Tunnel Layer* ist verschlüsselt (siehe Kapitel Tunnel Encryption).

Tunnel Messages sind eine Form von grossen I2NP Messages, welche mehrere verschlüsselte I2NP Messages (siehe unten) von einem anderen Typ und verschlüsselte Anweisungen für deren Zustellung enthalten (*Protocol Stack - I2P*, o. J.). Während I2NP Messages eine variable Grösse von 0 bis fast 64 KB haben, haben *Tunnel Messages* eine feste Grösse von etwa 1 KB (1028 Byte gemäss der I2NP-Dokumentation). Die feste Nachrichtengrösse schränkt verschiedene Arten von Angriffen ein, die durch Beobachtung der Nachrichtengrösse möglich sind. Damit diese Grösse eingehalten wird, verarbeitet der *Tunnel Gateway* (erste Knoten in einem Tunnel) die I2NP Messages vor, indem er diese, je nach Grösse, fragmentiert und zu *Tunnel Messages* zusammenfasst (*Tunnel Message Specification - I2P*, o. J.).

I2NP Messages werden definiert in dem *I2P Network Protocol*. Sie können für One-Hop-, Router-to-Router-, Punkt-zu-Punkt-Verbindungen verwendet werden. Durch die Verschlüsselung und Einbettung von Nachrichten in andere Nachrichten können diese auf sichere Weise über mehrere Zwischenstationen zum endgültigen Ziel gesendet werden. Wie bereits erwähnt sind sie ein wichtiger Teil von verschiedenen Schichten im I2P-Protkoll-Stack und können in folgende Kategorien aufgeteilt werden (*I2P Network Protocol (I2NP) - I2P*, o. J.):

- **Network Database Messages**: Diese Kategorie beihnhaltet Nachrichtentypen für die Interaktion mit der Network Database. Dazu gehören die *DatabaseLookupMessage*. *DatabaseSearchReply-Message* und *DatabaseStoreMessage*. Genauere Infos werden in dem Kapitel Network Database beschrieben.
- **Data Messages**: In dieser Kategorie befinden sich die Nachrichtentypen, welche die eigentlichen Daten Transportieren. Dazu gehören die *DataMessage*, *DeliveryStatusMessage* und die *GarlicMessage*. Grundsätzlich ist die *DeliveryStatusMessage* in einer *GarlicMessage* und die *GarlicMessage* in einer *DataMesssage* verpackt. *Garlic Encryption* wird genauer im Kapitel Tunnel Encryption erklärt.
- **Tunnel Messages**: In der letzten Kategorie befindet sich die bereits oben erwähnte *Tunnel Message* (*TunnelDataMessage*) und diverse Nachrichtentypen für den Aufbau von Tunnel.

Jeder Nachrichtentyp hat eine spezifische Priorität, mit welcher der I2P-Router die Nachricht behandeln soll. Dabei machen die *Tunnel Messages* (*TunnelDataMessages*) mit einer Piorität von 400 den Grossteil des Datenverkehrs aus. Das heisst, dass eigentlich alles über der Priorität von 400, im Grunde eine hohe Priorität, und alles darunter, eine niedrige Priorität hat (*I2P Network Protocol (I2NP) - I2P*, o. J.).

2.2.4 I2P Garlic Layer

Der *I2P Garlic Layer* bietet verschlüsselte und anonyme Ende-zu-Ende Nachrichtenübermittlung zwischen der eigentlichen Quelle und dem Ziel. Dabei werden wie bei den vorherigen Schichten verschachtelte I2NP Messages übermittelt. Die *Garlic Encryption* wird genauer im Kapitel Tunnel Encryption erklärt.

2.2.5 I2P Client Layer & I2P End-to-End Transport Layer

Über dem *I2P Garlic Layer* befinden sich der *I2P Client Layer* (I2CP) und der *I2P End-to-End Transport Layer* (Streaming, Datagrams). Die folgenden Schichten sind streng genommen nicht mehr Teil des *I2P Protocol Stack* und auch nicht Teil der Kernfunktionalität des I2P-Routers. Sie sind aber ein wichtiger Teil um die Kommunikation von verschiedenen Applikationen über I2P zu gewährleisten (*Protocol Stack - I2P*, o. J.).

Der *I2P Client Layer* mit dem *I2P Client Protocol (I2CP)* sorgt für eine strikte Trennung zwischen dem Router und jedem Client, der über das Netzwerk kommunizieren möchte. Zusammen mit der *Streaming Library* und der *Datagram Library* im *I2P End-to-End Transport Layer* ermöglicht I2CP eine TCP- oder UDP-ähnliche Schnittstelle zur Verfügung zu stellen. Dies ermöglicht, dass bestehende Anwendungen und Applikationen über I2P kommunizieren können, ohne mit irgendeiner Art von Kryptografie arbeiten zu müssen. Sie sind im Grunde genommen der Klebstoff zwischen I2P-Router und Applikationen die über I2P kommunizieren wollen (*Protocol Stack - I2P*, o. J.) (*I2CP - I2P*, o. J.).

2.2.6 I2P Application Layer

Abschliessend würde nun der *I2P Application Layer* kommen, mit den verschiedenen Applikationen und Anwendungen welche auf I2P Aufbauen. Darunter zählen Standardapplikationen wie z.B. Apache, OpenSSH oder Susimail, aber auch I2P spezifische Applikationen wie z.B. I2PSnark (*Protocol Stack - I2P*, o. J.) oder das zukünftige Produkt "DIVA" von "DIVA.EXCHANGE" (*DIVA.EXCHANGE*, o. J.).

2.3 Tunnel

Das I2P-Netz besteht aus einer Reihe von Knoten. Bei den Knoten redet man von sogenannten I2P-Routern. Jeder Router initiiert eine Reihe von virtuellen Tunnel. Diese virtuellen Tunnel erstrecken sich meist über mehrere I2P-Router, welche vom Initiator gewählt werden. I2P-Tunnel sind unidirektional und übertragen die eigentlichen Daten nur in eine Richtung. Um bidirektionale Kommunikation zu erreichen, verwendet I2P Inbound- und Outbound-Tunnel. Wie der Name schon sagt, wird der Inbound-Tunnel vom Initiator verwendet für die Übertragung von eingehenden Daten und der Outbound-Tunnel für die Übertragung von ausgehenden Daten (Herrmann & Grothoff, 2011).

2.3.1 Tunnel Pool

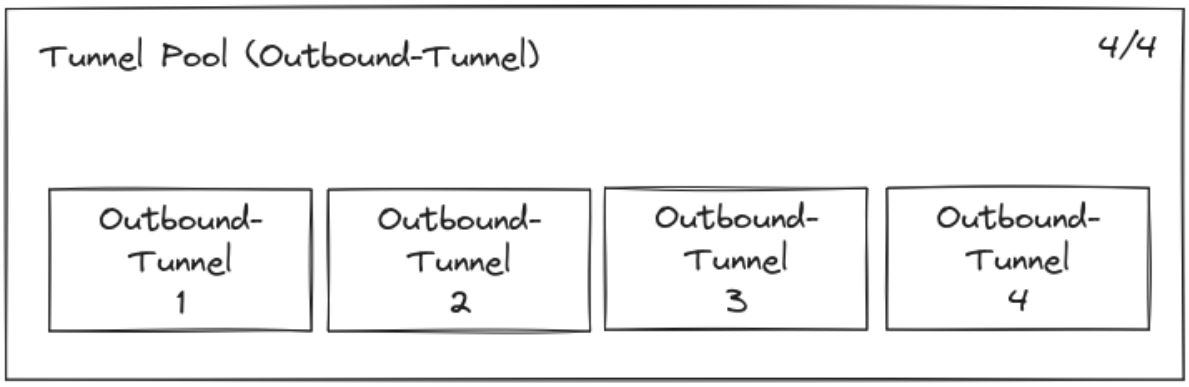

Abbildung 3: Tunnel Pool

Der I2P-Router erstellt und unterhält immer eine Reihe von Tunnel-Pools, die jeweils eine Gruppe von Tunneln für einen bestimmten Zweck mit einer eigenen Konfiguration verwalten. Das bedeutet, wenn ein Inbound- oder Outbound-Tunnel benötigt wird, wählt der Router einen aus dem entsprechenden Pool nach dem Zufallsprinzip aus. Dies ermöglicht einen effizienten Betrieb. Zudem können Inbound- und Outbound-Tunnel für verschiedene Verbindungen über einen gewissen Zeitraum wiederverwendet werden. (*Tunnel Implementation - I2P*, o. J.)

Jeder Tunnel-Pool verfügt über einige Schlüsseleinstellungen. Darunter fallen beispielsweise folgende Eigenschaften:

- Die Anzahl an Tunnel, die aktiv bleiben sollen (*Tunnel Implementation - I2P*, o. J.).
- Die Anzahl an Backup-Tunnel, die aktiv bleiben sollen (*Tunnel Implementation - I2P*, o. J.).
- Wie lange die einzelnen Tunnel sein bzw. wie viele I2P-Router für einen Tunnel verwendet werden sollen (*Tunnel Implementation - I2P*, o. J.).

Darüber hinaus gibt es auch Einstellung die z.B. die Tunnellänge randomisieren (*I2CP - I2P*, o. J.). Was konfiguriert werden kann unterscheidet sich stark von der I2P-Router-Implementation.

2.3.2 Beispiel HTTP-Request

Ein vereinfachtes Beispiel des eingehenden und ausgehenden Tunnel-Nachrichtenflusses könnte wie folgt aussehen: (siehe Abbildung 4)

1. Alice benutzt einen Browser, welcher ihren lokalen I2P-Router als HTTP-Proxy konfiguriert hat und ruft die I2P-Website von Bob via `GET /index.html HTTP/1.1` auf.
2. Der Webserver von Bob erhält die HTTP-Request und antwortet mit `HTTP/1.1 200 OK`.

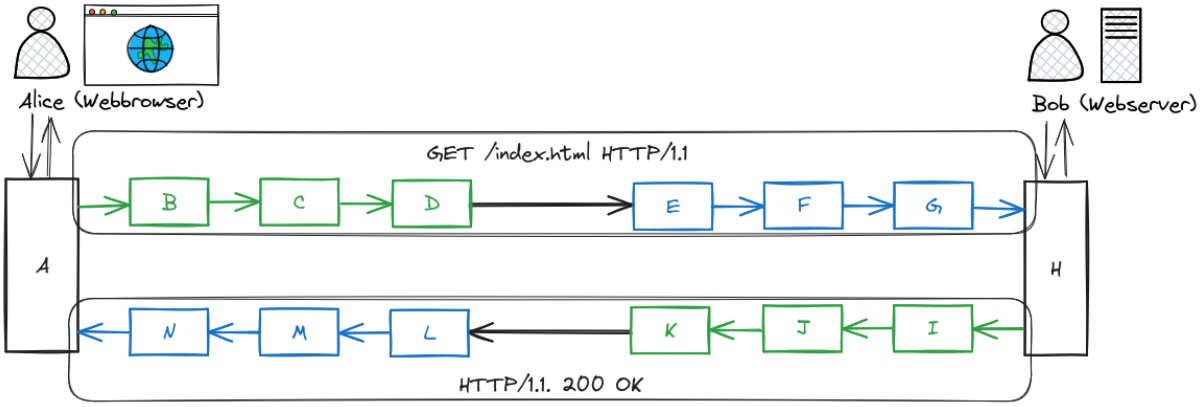

Abbildung 4: Beispiel zu Inbound- und Outbound-Tunnel

In der oberen Abbildung wird jeweils ein Inbound- und Outbound-Tunnel von Alice und Bob dargestellt, welche für die HTTP-Request und HTTP-Response gewählt wurden. Dabei wird eine Tunnellänge von 3 dargestellt, welche zurzeit dem Standard entspricht für diese Tunnel (*I2CP - I2P*, o. J.).

I2P-Router:

- **A bis N**: I2P-Router im I2P-Netzwerk
- **A**: Lokaler I2P-Router von Alice, welcher ein HTTP-Proxy zur Verfügung stellt

- **H**: Lokaler I2P-Router von Bob, welcher den Webserver I2P zur Verfügung stellt

Pfad der HTTP-Request von Alice:

Für die HTTP-Request wird ein *Outbound-Tunnel* von **A** aus dem entsprechenden Tunnel-Pool gewählt. Zudem wählt **A** einen *Inbound Gateway* (**E**), welcher im LeaseSet von Bob definiert ist.

- Outbound-Tunnel von Alice
  - **A**: *Outbound Gateway*
  - **B-C**: *Outbound Participant*
  - **D**: *Outbound Endpoint*
- Inbound-Tunnel von Bob
  - **E**: *Inbound Gateway*
  - **F-G**: *Inbound Participant*
  - **H**: *Inbound Endpoint*

Pfad der HTTP-Response von Bob:

Für die HTTP-Response wird ein *Outbound-Tunnel* von **H** aus dem entsprechenden Tunnel-Pool gewählt. Zudem wählt **H** einen *Inbound Gateway* (**L**), welcher im LeaseSet von Alice definiert ist.

- *Outbound-Tunnel* von Bob
  - **H**: *Outbound Gateway*
  - **I-J**: *Outbound Participant*
  - **K**: *Outbound Endpoint*
- *Inbound-Tunnel* von Alice
  - **L**: *Inbound Gateway*
  - **M-N**: *Inbound Participant*
  - **A**: *Inbound Endpoint*

2.3.3 Tunnel Kategorien

Neben *Inbound-* und *Outbound-Tunnel* wird noch zwischen Exploratory-Tunnel und Client-Tunnel unterschieden. Grundsätzlich werden die vom I2P-Router selbst verwendeten Tunnel als Exploratory-Tunnel und die von Anwendungen genutzten Tunnel als Client-Tunnel bezeichnet (*Peer Profiling and Selection - I2P*, o. J.).

- **Client-Tunnel**: Die Client-Tunnel werden von den Anwendungen im *I2P Application Layer* verwendet für das Transportieren von Nachrichten innerhalb des I2P-Netzwerks. Bei dem oberen Beispiel mit der Abfrage des Webservers handelt es sich um vier *Client-Tunnel* (zwei Inbound- und zwei Outbound-Client-Tunnel) (*Peer Profiling and Selection - I2P*, o. J.).
- **Explorator-Tunnel**: Exploratory Tunnel dienen verschiedenen administrativen Funktionen des I2P-Routers, darunter dem *Network Databse* Verkehr und dem Testen von *Client-Tunnel*. Sie dienen dem Aufbau von Verbindungen mit zuvor nicht verbundenen I2P-Routern, was ihnen die Bezeichnung “Exploratory” einbrachte. Wie Client-Tunnel sind Exploratory-Tunnel auch unidirektional. (*Peer Profiling and Selection - I2P*, o. J.).

Beide Arten von Tunnel sind technisch gleich aufgebaut. Jedoch unterscheiden sie sich in der *Peer Selection*, also welche Knoten von einem I2P-Router für den Tunnel gewählt werden (*Peer Profiling and Selection - I2P*, o. J.). Je nach I2P-Router-Implementation unterscheiden sie sich auch in der standardmässig konfigurierten Länge. i2p.i2p und i2pd verwenden standardmässig für Exploratory Tunnel eine Länge von zwei und für Client Tunnel eine Länge von drei Knoten (*Configuration File Specification - I2P*, o. J.) (*Configuring - i2pd documentation*, o. J.).

2.4 Peer Selection

Im oberen Abschnitt wurde erwähnt, dass sich die Exploratory Tunnel und die Client Tunnel in ihrer *Peer Selection* unterscheiden. Bei der *Peer Selection* geht es darum, die I2P-Router im Netz auszuwählen, welche für unsere Tunnel verwenden wollen. Um dies zu erreichen, wird die Leistung jedes I2P-Routers aufgezeichnet (bekannt unter *Peer Profiling*) um abzuschätzen, wie schnell ein Router ist, wie oft er Anfragen annimmt und ob er überlastet ist. Die *Peer Selection* erfolgt recht häufg, da ein I2P-Router eine grosse Anzahl an Tunnel bzw. Tunnel Pools unterhält und die Lebensdauer eines Tunnels nur 10 Minuten beträgt (*Peer Profiling and Selection - I2P*, o. J.).

2.5 Tunnel Encryption

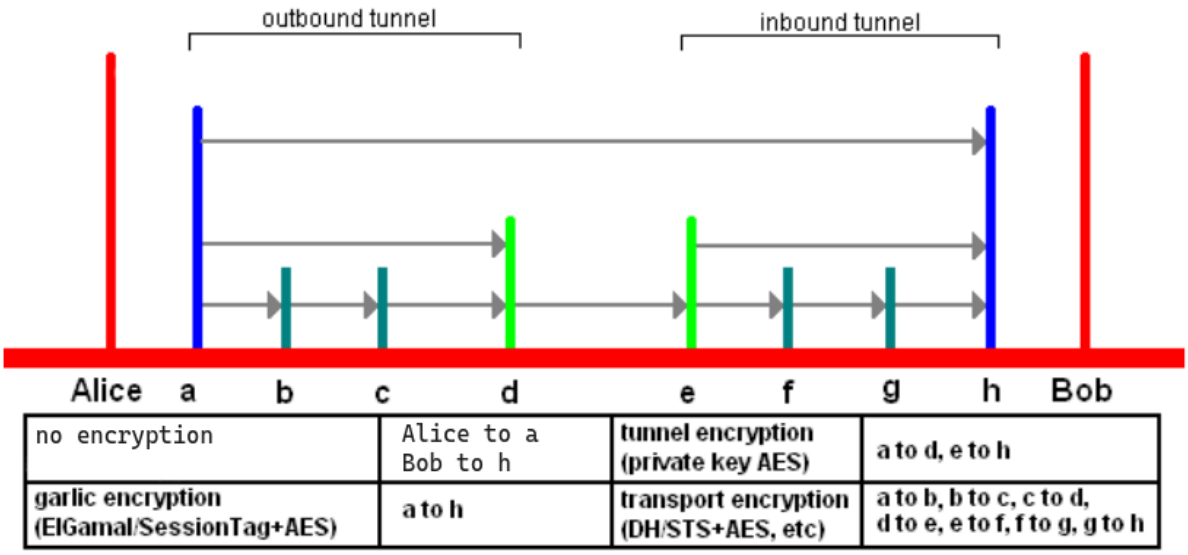

Abbildung 5: I2P Tunnel Encryption

Inhalte, die über das I2P-Netzwerk gesendet werden, werden auf drei Ebenen verschlüsselt. Jede Ebene entspricht einer Schicht in unserem Protokoll-Stack. Im Kapitel I2P Protocol Stack wurden die drei Schichten *I2P Transport Layer*, *I2P Tunnel Layer* und *I2P Garlic Layer* beschrieben, welche verschlüsselt sind. Jede dieser Schichten verwendet eine andere Art der Verschlüsselung (*A Gentle Introduction to How I2P Works - I2P*, o. J.).

- **Transport Enyption** (*I2P Transport Layer*): Wird verwendet für die Verschlüsselung zwischen den einzelnen I2P-Routern.
- **Tunnel Encryption** (*I2P Tunnel Layer*): Wird verwendet für die Verschlüsselung aller Nachrichten in einem Tunnel, dabei wird vom Tunnel-Gateway bis zum Tunnel-Endpoint verschlüsselt.
- **Garlic Encryption** (*I2P Garlic Layer*): Wird für die End-to-End Verschlüsselung zwischen Sender und Empfänger verwendet

Um den Rahmen nicht zu sprengen, werden die Arten der Verschlüsselungen in den folgenden Abschnitten nur sehr grob beschrieben. Dabei wird mehr Wert auf die eigentlichen Konzepte als die kryptografischen Chiffren oder Algorithmen gesetzt.

2.5.1 Transport Encryption

Bei der *Transport Encryption* wird der Verkehr zwischen jedem Knoten bzw. I2P-Router verschlüsselt. Beide Protokolle NTCP und SSU im *I2P Transport Layer* verwenden Verschlüsselung. Beide verwenden eine Kombination von Diffie-Hellman-Schlüsselaustausch (DH) und symmetrischer Verschlüsselung. NTCP verwendet zusätzlich noch das *Noise Protocol* (*Low-level Cryptography Specification - I2P*, o. J.).

2.5.2 Tunnel Encryption

Bei der *Tunnel Encryption* handelt es sich auch um eine *Layered Encryption* welche *Garlic Routing* verwendet.

Garlic Routing ist eine Variante des *Onion Routing*. (*Garlic Routing - I2P*, o. J.). *Onion Routing* ist eine Technik zum Aufbau von Pfaden oder Tunneln durch eine Reihe von Knoten und zur anschliessenden Nutzung dieser Tunnel. Die Nachrichten werden vom Absender mehrmals verschlüsselt (in Schichten) und dann von jedem Knoten entschlüsselt. Während der Aufbauphase werden jedem Peer nur die Routing-Anweisungen (*Garlic Clove Delivery Instructions*) für den nächsten Knoten mitgeteilt. Während der Betriebsphase werden die Nachrichten durch den Tunnel geleitet, und die Nachricht und ihre Routing-Anweisungen sind nur dem Endpunkt des Tunnels zugänglich (*Garlic Routing - I2P*, o. J.).

Wie oben erwähnt beginnt die *Tunnel Encryption* beim Tunnel-Gateway und endet bei dem Tunnel-Endpoint. Wie die Verschlüsselung am Tunnel-Gateway erfolgt, hängt davon ab, ob es sich um einen *Inbound-* oder *Outbound-Tunnel* handelt (*Tunnel Implementation - I2P*, o. J.)

Outbound-Tunnel:

Rolle	Verschlüsselungsoperation
Outbound Gateway (Initiator)	iteratives Verschlüsseln für jeden Knoten im Tunnel
Outbound Participant	Entschlüsseln
Outbound Endpoint	Entschlüsseln

Der Outbound-Tunnel ist dazu da, ausgehende Daten von dem Initiator des Tunnels zu transportieren. Aus diesem Grund verschlüsselt der *Outbound Gateway* die *Tunnel Message* iterativ für jeden Knoten im Tunnel. Am Ende des *Outbound-Tunnels* wird die *Tunnel Message* im Klartext aufgedeckt.

Inbound-Tunnel:

Rolle	Verschlüsselungsoperation
Inbound Gateway	Verschlüsseln
Inbound Participant	Verschlüsseln
Inbound Endpoint (Initiator)	vterative Entschlüsseln für jeden Knoten im Tunnel

Der Inbound Tunnel ist dazu da, eingehende Daten von dem Initiator des Tunnels zu transportieren. Jeder Knoten verschlüsselt die erhaltenen Daten erneut und nur der *Inbound Endpoint*kann die *Tunnel Message* mit iterativem entschlüsseln aufdecken.

Der Initiator des Tunnels wählt genau die Knoten aus, die jeweiligen Inbound- oder Outbound-Tunnel teilnehmen sollen. Die Länge eines Tunnels soll weder für *Tunnel Participants* noch für Dritte feststellbar sein. Durch die *Tunnel Encryption* haben die *Tunnel Participants* nie Zugriff auf die *Tunnel Messages* im Klartext. Da I2P für die *Tunnel Encryption* eine symmetrische AES-Verschlüsselung verwendet und neben dem verschlüsselten Daten auch den verschlüsselten Initialisierungsvektor (IV) zwischen den Knoten transportiert, ist die Verschlüsselung und Entschlüsselung für ein *Tunnel Participant* die selbe Operation. Somit wissen die *Tunnel Participants* nicht, ob sie sich in einem *Inbound-* oder *Outbound-Tunnel* befinden, da sie immer “verschlüsseln” für den nächsten Knoten (*Tunnel Implementation - I2P*, o. J.).

2.5.3 Garlic Encryption

Während die Tunnel selbst über eine *Layered Encryption* verfügen, um eine unbefugte Weitergabe an die Knoten innerhalb des Netzes zu verhindern (ebenso wie die Transportschicht selbst, um eine unbefugte Weitergabe an Knoten ausserhalb des Netzes zu verhindern), ist es notwendig, eine zusätzliche End-to-End Verschlüsselung hinzuzufügen, um die eigentliche Nachricht vor dem *Outbound Endpoint* und dem *Inbound Gateway* zu verbergen. Diese *Garlic Encryption* ermöglicht es dem I2P-Router des Senders, mehrere Nachrichten in eine einzige “Garlic Message” zu verpacken, die mit einem bestimmten Public-Key verschlüsselt ist, so dass die Knoten weder feststellen können, wie viele Nachrichten in dem *Garlic Clove* enthalten sind, noch was diese Nachrichten beinhalten oder wohin die einzelnen *Garlic Messages* gehen sollen. Für eine typische Ende-zu-Ende-Kommunikation zwischen dem Sender und dem Empfänger wird die *Garlic Message* mit dem Public-Key verschlüsselt, der in dem LeaseSet des Empfängers veröffentlicht wurde. Im Endefeckt handelt es sich bei der *Garlic Encryption* nur um eine einmaligen Verschlüsselung der eigentlichen Nachricht.

2.6 Network Database

Die *Network Database* oder kurz NetDB von I2P ist eine verteilte Datenbank, die zwei Arten von Daten enthält (*The Network Database - I2P*, o. J.):

1. *RouterInfos*: Eine RouterInfo speichert Informationen über einen bestimmten I2P-Router und wie man ihn kontaktieren kann (*I2P*, o. J.).
2. *LeaseSets*: Ein LeaseSet speichert Informationen über ein bestimmtes Ziel (z.B. I2P-Website) (*I2P*, o. J.).

Jeder Teil der Daten wird von der entsprechenden Partei signiert und von jedem, der sie verwendet oder speichert, überprüft. Darüber hinaus enthalten die Einträge in der NetDB Informationen über ihre Lebensdauer, so dass irrelevante Einträge gelöscht werden können, neuere Einträge ältere ersetzen können und ein Schutz gegen bestimmte Angriffsklassen besteht (*The Network Database - I2P*, o. J.).

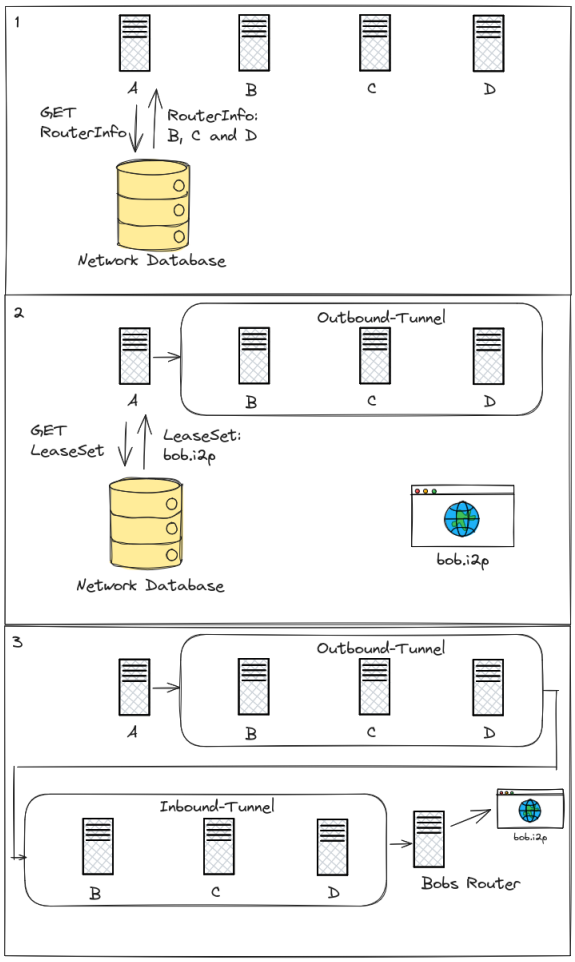

Abbildung 6: Beispiel Network Database

Im I2P wird die Verteilung von NetDB-Informationen mithilfe der Store- und Lookup-Nachrichten realisiert. Dafür gibt es drei Arten von I2NP Messages.

- **Lookup:** *DatabaseLookupMessage* und *DatabaseSearchReplyMessage*
- **Store:** *DatabaseStoreMessage*

Die Store-Funktion ist dazu da, um ein LeaseSet im I2P-Netzwerk bekannt zu machen. Dies geschieht, indem das entsprechende LeaseSet an eine bestimmte Anzahl von Floodfills gesendet wird. Ein I2P-Router übermittelt die *DatabaseStoreMessage* an eine definierte Anzahl von Floodfills. Diese Floodfills speichern das LeaseSet in ihrer NetDB, sodass es von anderen Teilnehmern im Netzwerk gefunden werden kann (*The Network Database - I2P*, o. J.).

Die Lookup-Funktion wird benötigt, wenn ein I2P-Router eine Verbindung mit einem I2P-Dienst herstellen möchte. Falls diese Information nicht in seiner NetDB vorhanden ist, kann er die Kontaktinformationen mithilfe der *DatabaseLookupMessage* bei einem Floodfill anfragen. Der Floodfill, der die *DatabaseLookupMessage* erhält, überprüft seine eigene NetDB, ob er die angefragte Identität kennt. Falls nicht, antwortet der Floodfill mit einer *DatabaseSearchReplyMessage*, die eine Liste mit weiteren Floodfills (*The Network Database - I2P*, o. J.).

2.6.1 Floodfill

Bei Floodfill wird meist von einem Floodfill-I2P-Router geredet. Die einzigen zusätzlichen Aufgaben eines Floodfill-Routers sind die Annahme von *DatabaseLookupMessage* und*DatabaseStoreMessage*. Die Floodfill-Funktionalität wird automatisch nur auf I2P-Routern aktiviert, die mit hohen Bandbreitenlimits konfiguriert sind. Darüber hinaus kann diese auch manuell aktiviert werden (*The Network Database - I2P*, o. J.).

2.6.2 Reseeding

Ein wichtiger Aspekt des I2P-Netzwerks ist das sogenannte “Reseeding”, was den Initialisierungsprozess der NetDB beschreibt. Ohne ausreichende Einträge in der NetDB kann die ordnungsgemässe Funktionalität des I2P-Routers nicht gewährleistet werden. Ohne Einträge in der NetDB ist es nicht möglich, Tunnel für die Kommunikation zu erzeugen oder Kontaktinformationen von anderen I2P-Routern abzufragen. Daher ist das Reseeding von entscheidender Bedeutung, um sicherzustellen, dass jeder Router die notwendigen Informationen über aktive Peers erhält und somit am I2P-Netzwerk teilnehmen kann.

2.7 Angriffe auf I2P

Die Entwicklung von I2P begann im Jahr 2003, und das Desgin profitierte wesentlich von der Forschung, die zu dieser Zeit veröffentlich wurde. Darunter zählt Onion Routing, Freenet und Tor. I2P benutzt mit dem Garlic-Routing eine Form von Onion-Routing, sodass es weiterhin von dem grossen akademischen Interesse an Tor profitiert. Das I2P-Threat-Model unterscheidet zwischen einer Reihe von verschiedenen Angriffen. Dazu gehören auch einige Angriffe welche möglicherweise einzigartig für I2P sind (*I2P's Threat Model - I2P*, o. J.). Diese können in folgende Arten eingeteilt werden:

- **Brute-Force-Angriffe:** Bei einem Brute-Force-Angriff geht es grundsätzlich darum, Nachrichten zwischen den Knoten zu beobachten und herauszufinden, welche Nachrichten welchem Pfad folgen. Dazu kann ein Angreifer beispielsweise sehr viel Datenverkehr an ein I2P-Dienst senden und Knoten für einen Pfad ausschliessen, welche nicht eine grosse Menge von Daten erhalten haben (*I2P's Threat Model - I2P*, o. J.).
- **Timing-Angriffe:** Wie der Name schon annehmen lässt, geht es bei einem Timing-Angriff darum, Muster in der Frequenz von Nachrichten zu erkennen (*I2P's Threat Model - I2P*, o. J.).
- **Überschneidungs-Angriffe:** Bei Überschneidungs-Angriffen geht es darum abzugleichen, welche Knoten sich im Netzwerk befinden.Im Laufe der Zeit kann der Angreifer durch die Abwanderung von Knoten erhebliche Informationen über das Ziel gewinnen, indem er einfach die Gruppen von Knoten überschneidet, die online sind, wenn eine Nachricht erfolgreich übermittelt wird (*I2P's Threat Model - I2P*, o. J.).
- **Denial-of-Service-Angriffe:** Bei den Denial-of-Service-Angriffen geht es um die Überlastung von Ressourcen (*I2P's Threat Model - I2P*, o. J.).
- **Tagging-Angriffe:** Unter Tagging-Angriff versteht man die Änderung einer Nachricht, damit sie später im weiteren Verlauf des Weges identifiziert werden kann (*I2P's Threat Model - I2P*, o. J.).
- **Partitionierungs-Angriffe:** Bei einem Partitionierungs-Angriff geht es um die Suche nach einer Möglichkeit I2P-Router im Netzwerk, technisch oder analytisch zu trennen bzw. zu partitionnieren (*I2P's Threat Model - I2P*, o. J.).
- **Vorgänger-Angriffe:** Vorgänger-Angriffe beziehen sich auf das Sammeln von Informationen um herauszufinden welche Knoten sich in der Nähe eines Ziels befinden. Ein Angreifer könnte dafür an dem Tunneln eines Ziels teilnehmen und den vorherigen (Vorgänger) oder nächsten Hop verfolgen (*I2P's Threat Model - I2P*, o. J.).
- **Harvesting-Angriffe:** Beim "Harvesting" geht es um das Zusammenstellen einer Liste von I2P-Benutzern. Es kann zur Unterstützung anderer Angriffe verwendet werden, indem man einfach einen I2P-Router startet, sieht, mit wem er sich verbindet, und alle Referenzen zu anderen I2P-Router sammelt, die er finden kann.

- **Identifizierung durch Verkehrsanalyse:** Bei der Verkehrsanalyse liegt der Fokus auf der Inspektion des ein- und ausgehenden Datenverkehrs eines I2P-Routers und der Identifikation, ob auf einem Computer I2P-Softwäre läuft (*I2P's Threat Model - I2P*, o. J.).
- **Sybil-Angriffe:** Ein bekannter Angriff auf Anonymitätssysteme wie I2P ist der sogenannte Sybil-Angriff (Egger et al., 2013). Sybil beschreibt eine Kategorie von Angriffen, bei denen der Angreifer eine grosse Anzahl von Knoten kontrolliert und die erhöhte Anzahl dazu verwendet, andere Angriffe zu starten. Wenn ein Angreifer beispielsweise in einem Netz, in dem die Teilnehmer nach dem Zufallsprinzip ausgewählt werden, eine 80-prozentige Chance haben möchte, einer dieser Teilnehmer zu sein, erstellt er einfach die fünffache Anzahl der im Netz vorhandenen Knoten (*I2P's Threat Model - I2P*, o. J.). Peer-to-Peer-Systeme wie I2P stützen sich in der Regel auf die Existenz mehrerer, unabhängiger Entitäten. Da es keine zentrale vertrauenswürdige Autorität im I2P-Netzwerkwerk gibt, die für die einzelnen Knoten bürgen kann, ist es immer möglich, dass eine unbekannte Gruppe oder Einheit die Kontrolle über mehrere Knoten vorweisen kann (Douceur, 2002).
- **Buddy-Exhaustion-Angriffe:** Beim Buddy-Exchaustion-Angriff ist das Ziel, sicherzustellen, dass ein Tunnel ausschliesslich aus von einem Angreifer kontrollierten I2P-Router besteht. Dabei weigern sich vom Angreifer kontrollierten I2P-Router, Anfragen zum Tunnelaufbau anzunehmen oder weiterzuleiten, wenn andere (nicht vom Angreifer kontrollierte) I2P-Router beteiligt sind (Wang et al., 2010) (*I2P's Threat Model - I2P*, o. J.).
- **Kryptographische Angriffe:** Bei kryptografischen Angriffen werden Schwachstellen kryptografischen Algorithmen ausgenutzt, um die Vertraulichkeit, Integrität oder Authentizität (CIA) von Informationen zu gefährden (*I2P's Threat Model - I2P*, o. J.).
- **Floodfill- und NetDB-Angriffe:** Floodfill- und NetDB-Angriffe beziehen sich auf das Betreiben von böswilligen Floodfill-Router, welche schlechte, falsche, langsame oder gar keine Antworten zu liefern (*I2P's Threat Model - I2P*, o. J.).
- **Angriffe auf zentrale Ressourcen:** Diese Angriffe beschreiben zentralisierte oder begrenzte Ressourcen innerhalb oder ausserhalb des I2P-Netzwerkes als Angriffsvektor zu verwenden. Zu diesen Ressourcen zählen beispielsweise die System, welche den I2P-Source-Code bereitstellen.
- **Angriffe auf Entwicklung und Implementierung (Bugs):** Diese Angriffe zielen auf Schwachstellen (absichtlich oder unabsichtlich) in I2P-Software.

Viele der hier beschriebenen Angriffe setzen andere Angriffe voraus oder sind nur in der Kombination mit anderen Angriffe möglich.

2.8 I2P-Clients

Die offenen Spezifikationen des I2P-Protokolls bieten eine Grundlage für die Entwicklung verschiedener Clients, die eine Vielfalt von Implementierungen ermöglichen und es Entwicklern erlauben, Anwendungen zu erstellen, die auf unterschiedliche Plattformen und Anwendungsfälle zugeschnitten sind. I2P verfügt mittlerweile über mehre I2P-Router-Implementationen, sogenannten I2P-Clients. Das Wesen von Open-Source erlaubt es den Entwicklern, in verschiedene Richtungen zu gehen, was die Flexibilität und Kreativität innerhalb des I2P-Ökosystems fördert. Die Entwickler haben die Freiheit, neue Funktionen zu erforschen, bestehende zu verbessern und mit neuen Ansätzen zu experimentieren, um die Gesamtfunktionalität von I2P zu verbessern. Aus diesem Grund, kann es sein, dass verschiedene Clients unterschiedliche Verhaltensweisen zeigen und unterschiedliche Sicherheitsmassnahmen anbieten können.

i2p.i2p

Die primäre und offizielle Implementierung von I2P (bekannt als *i2p.i2p* oder *Java I2P*) ist in Java geschrieben. Sie ist die erste I2P-Implementation und wurde von den die Schöpfer des I2P-Protokolls entwickelt (sadie, o. J.).

i2pd

I2pd (kurz für I2P-Daemon) ist eine vollwertige C++-Implementierung des I2P-Clients, welche von dem PurpleI2P-Team und under Community entwickelt wird. Nach der I2P-Dokumentation ist i2pd inzwischen stabil genug, um in der “Produktion” eingesetzt zu werden, und implementiert seit Sommer 2016 alle I2P-APIs vollständig (*Alternative I2P clients - I2P*, o. J.).

Andere

Die I2P-Community entwickelt darüber hinaus an vorläufigen Versionen und Prototypen wie z.B. go-i2p oder i2p-rs) und pflegt verschiedene Forks wie z.B. kovri, ein Fork von i2pd oder I2P+, ein Fork von i2p.i2p.

3 Ideen & Konzept

In diesem Kapitel wird ein genauerer Blick auf die im Verlauf der Auseinandersetzung mit dem definierten Problemfeld entstandenen Ideen und Konzepte geworfen. Der Fokus liegt dabei darauf, wie diese Ideen in Bezug auf das vorliegende Problem stehen und welche innovativen Elemente sie mit sich bringen.

Bevor den Lösungsansätzen Aufmerksamkeit geschenkt wird, ist es wichtig, einen klaren Bezug zum zugrundeliegenden Problem herzustellen. Ein tieferes Verständnis dieses Problems bildet die Grundlage für die darauf aufbauenden Lösungsvorschläge. Es wird herausgearbeitet, wie sich die entwickelten Ideen aus der Problemstellung heraus entwickelt haben und inwiefern sie geeignet sind, die identifizierten Herausforderungen zu bewältigen.

3.1 I2P-Testnetzwerk

Ein erster Ansatzpunkt besteht in der Recherche nach bestehenden Implementierungen von I2P-Testnetzwerken. Dabei ist es entscheidend, Projekte zu identifizieren, die sich auf den automatischen Aufbau und die Reproduzierbarkeit eines I2P-Testnetzwerks fokussieren. Hierbei könnten existierende Open-Source-Repositories oder Forschungsprojekte eine vielversprechende Ausgangsbasis bieten.

Die Untersuchung potenzieller Projekte sollte auch die Analyse der genutzten Komponenten einschliessen, um herauszufinden welche “Bausteine” für einen erfolgreichen Aufbau erforderlich sind. Ein tiefer Einblick in die Architektur solcher Projekte ist essenziell, um eine fundierte Entscheidung über die Auswahl der benötigten Ressourcen treffen zu können. Im Kontext des I2P-Testnetzwerkes sollte besonders darauf geachtet werden, welcher Client bzw. welche I2P-Router-Implementation sich am besten für den automatischen und reproduzierbaren Aufbau eines Testnetzwerks eignet. Dies könnte Aspekte wie Skalierbarkeit, Konfigurierbarkeit und Unterstützung für Automatisierungsskripte oder bestehender Orchestrators beinhalten.

Des Weiteren ist es wichtig, mögliche Herausforderungen im Rahmen des I2P-Testnetzwerkes zu identifizieren. Dies könnten potenzielle technische Schwierigkeiten oder Limitierungen der aktuellen Implementierungen einschliessen. Ein umfassendes Verständnis dieser Herausforderungen ermöglicht es, geeignete Lösungsansätze zu entwickeln und in das Gesamtkonzept zu integrieren. Um mögliche “Show-Stopper” zu vermeiden sollen jeweils

Zusammengefasst bildet die Analyse von bestehenden Projekten, die Auswahl der notwendigen Komponenten, die Identifikation des optimalen I2P-Clients und die Bewertung potenzieller Herausforderungen eine solide Grundlage für die Entwicklung eines Konzepts zum automatischen und reproduzierbaren Aufbau eines unabhängigen I2P-Testnetzwerks.

Boss und Purtschert haben in ihrer Arbeit bereits eine Implementation eines unter dem Pseudonym "LNS" oder "l-n-s" bekannten Github-Benutzers (LNS, o. J.-b) genauer angeschaut (Boss & Purtschert, 2022). Die Implementation von "LNS" soll als Startpunkt dieser Untersuchung dienen.

3.2 Verkehrs im I2P-Testnetzwerk

Nachdem die grundlegende Struktur für den Aufbau eines I2P-Testnetzwerks festgelegt wurden, fokussiert sich ein weiterer entscheidender Aspekt auf die Simulation von Netzwerkverkehr im Testnetzwerk. Dazu sollen Möglichkeiten erforscht werden, wie die I2P-Router im Testnetzwerk verwendet werden können um Verkehr hin und her zu schicken. Dabei ist die Idee sich hauptsächlich auf HTTP-Verkehr zu konzentrieren, um den Zugriff auf I2P-Webseiten zu simulieren. Das bedeutet, dass neben den I2P-Routern auch Webserver teil des Testnetzwerkes sein sollen und man in der Lage sein sollte, jegliche Knoten im Testnetzwerk als Einstiegspunkt zu verwenden.

3.3 Datengewinnung im I2P-Testnetzwerk

Um Angriffe in dem I2P-Testnetzwerk überwachen und nachvollziehen zu können, sollen diverse Daten gesammelt werden.

1. **Netzwerkverkehr:** Netzwerkverkehr soll, wenn möglich, auf die I2P-Router im Netzwerk zugeordnet werden können.
2. **Logs:** Jegliche Logs der einzelnen I2P-Router soll gesammelt werden.
3. **APIs:** Es soll untersucht werden, welche alternativen Möglichkeiten von den I2P-Router-Implementationen zur Verfügung gestellt werden, um Daten über das Verhalten oder Zustand zu gewinnen.

3.4 Angriffs-Simulation

Nach der Etablierung der Grundlagen und dem erfolgreichen Aufbau der Laborumgebung (I2P-Testnetzwerks) liegt die Absicht in der Durchführung von Angriffen. Wie in der Abgrenzung erwähnt liegt der Fokus nicht auf der Ausarbeitung von neuen Angriffsideen. In diesem Teil sollen bestehende Angriffsideen praktisch durchgeführt und wenn möglich validiert werden. Die Grundlage dafür wurde einerseits in der Arbeit von Brian Boss und Marco Purtschert und andererseits in Sitzungen mit Konrad Bächler und Dieter Arnold während der Projektphase geschaffen. Das Ziel der Angriffe soll die Deanonymisierung von Teilnehmern im Netzwerk sein. Dass heisst es geht nicht darum, was für Daten genau im Netzwerk transportiert werden. Viel mehr soll Quelle oder das Ziel einer Kommunikation ausfindig gemacht werden. Dazu wurden zwei Angriffsideen ausgewählt.

3.4.1 Tunnelkontrolle

Eine der Angriffsideen mit welcher sich Boss und Purtschert auseinandergesetzt haben, ist die Tunnelkontrolle. Im Kontext davon haben sie sich zwei Varianten überlegt. Die vollständige Tunnelkontrolle und die partielle Tunnelkontrolle (Boss & Purtschert, 2022).

In der vollständigen Tunnelkontrolle geht es grundsätzlich darum alle Hops bzw. I2P-Router eines vom Ziel erstellten Tunnels zu kontrollieren. Zusätzlich muss der Sender oder Empfänger der eigentlichen Daten ein I2P-Router sein, welcher vom Angreifer kontrolliert wird und es muss während der Kommunikation der kontrollierte Tunnel vom Ziel verwendet werden. Somit müssen folgende Voraussetzungen gegeben sein (Boss & Purtschert, 2022).

Voraussetzungen:

1. Der Angreifer muss entweder der **Sender** oder **Empfänger** einer Nachricht sein (das Gegenstück zum Opfer).
2. Dies wiederum setzt voraus, …
    - **Angreifer ist Sender**: dass das Opfer ein unverschlüsseltes LeaseSet in der Network Database veröffentlicht hat, damit der Angreifer eine Verbindung mit dem Opfer herstellen kann.
    - **Angreifer ist Empfänger**: dass der Angreifer einen I2P-Dienst betreibt, das unverschlüsselte LeaseSet in der Network Database veröffentlicht hat und das Opfer auf den vom Angreifer kontrollierten I2P-Dienst zugreifen will.
3. Der Angreifer muss die Kontrolle über die eigenen Inbound- und Outbound-Tunnel haben.
4. Der Angreifer muss alle I2P-Router bzw. Knoten in einem Inbound- oder Outbound-Tunnel des Opfers kontrollieren.
5. Der vom Angreifer kontrollierte Opfer-Tunnel muss für den Transport zwischen Angreifer und Opfer verwendet werden.

In der partiellen Tunnelkontrolle würde der Angreifer nicht alle I2P-Router in dem von dem Opfer erstellten Tunnel kontrollieren. Sondern, dass bei einer Tunnelgrösse von drei Hops es bereits ausreicht, wenn der Angreifer den ersten (Gateway) und letzten (Endpoint) I2P-Router eines Tunnels kontrolliert (Boss & Purtschert, 2022). Für die genaue Beschreibung der Tunnelkontrolle-Angriffsidee wird auf die Arbeit von Boss und Purtschert verwiesen.

Durchführbarkeit im Testnetzwerk:

Nach Boss und Purtschert gibt es keine Hinweise die auf eine Verunmöglichung dieser Idee hindeuten würden (Boss & Purtschert, 2022). Auch in der Recherche dieser Arbeit wurden bis jetzt keine Informationen gesammelt, die das grundsätzlich widerlegen würden. Nach initialer Recherche sollten die Voraussetzungen in einer Laborumgebung (I2P-Testnetzwerk) erreicht werden können.

Für die Umsetzung im Testnetzwerk stellt sich jedoch die Frage, wie gut die Datengewinnung von "kontrollierten" I2P-Router mit den bestehenden I2P-Router-Implementationen sein wird. Die Herausforderung wird darin bestehen, herauszufinden, ob die von der I2P-Router-Implementation bereitgestellten Informationen, sei es durch Logs oder alternative Quellen, genug sind, um diese für eine Deanonymisierung zu kombinieren. Selbst wenn umfassende Logs von der I2P-Router-Implementation verfügbar sind, stellt sich die Frage zu deren Relevanz und Kontext.

Falls die von der I2P-Router-Implementation bereitgestellten Daten nicht umfassend genug sind um Informationen zum Tunnelaufbau oder zu den *Tunnel Messages* zu erlangen, müsste eine andere Implementation gewählt werden, die diese bereitstellt oder die I2P-Router-Implementation angepasst werden. Dies hätte zur Folge, dass der Sourcecode einer I2P-Router-Implementation umfassend untersucht werden muss, was den Zeitrahmen für diese Arbeit übersteigen würde.

3.4.2 Mustererkennung Datenverkehr

Eine andere Angriffsidee mit welcher sich Boss und Purterschert auseinandergesetzt haben, ist die Mustererkennung des Datenverkehrs. I2P ist ein Overlay-Network und baut auf der bestehenden Internet-Infrastruktur auf (siehe I2P Protocol Stack). Somit fliesst im Endeffekt jeglicher Datenverkehr über zentrale Anbieter wie z.B. Internet-Service-Provider (ISP) oder Unterseekabel-Betreiber, welche den Datenverkehr aufzeichnen können. Boss und Purtschert beschreiben, dass in dieser Angriffsidee Daten gezielt versendet werden sollen um ein Muster zu generieren, welches von einem Angreifer verfolgt werden kann. Der Fokus bei dieser Idee liegt bei der Analyse des Datenverkehrs. Der eigentliche Inhalt des Verkehrs in diesem Fall nicht relevant. Ein Angreifer müsste einen grossen Teil des Internetverkehrs überwachen können oder bereits Informationen über den Standort des Opfers haben um gezielt Verkehr aufzuzeichnen. In diesem Fall müssten folgende Voraussetzungen gegeben sein (Boss & Purtschert, 2022).

Voraussetzungen:

1. Der Angreifer muss einen grossen Teil des Tunnel-Verkehrs überwachen können.
2. Der Angreifer muss ein I2P-Router kontrollieren um gezielten Datenverkehr zu versenden.

Durchführbarkeit im Testnetzwerk:

Wie Boss und Purtschert erwähnt haben, werden in diesem Angriff nicht direkt I2P-Protokolle angegriffen, sondern die darunterliegende Infrastruktur. Da in einer Laborumgebung alle Netze in denen sich I2P-Router befinden kontrolliert werden, sollte die Überwachung und Aufzeichnung des Datenverkehrs kein Problem darstellen. Da es sich bei dem Datenverkehr in Tunnel um mehrfach verschlüsselte und fragmentierte oder zusammengesetzte I2NP Messages handelt ist noch unklar, welche Erkenntnisse daraus gezogen werden können.

4 Methoden

Das vorliegende Kapitel widmet sich der detaillierten Darstellung der angewandten Methoden, die im Rahmen dieser Forschungsarbeit zum Einsatz kamen.

4.1 Literaturrecherche

Die Literaturrecherche stellt eine Schlüsselmethode im Forschungsprozess dar. Sie diente nicht nur als Mittel zur Informationsbeschaffung, sondern auch als kritische Grundlage für das Verständnis von I2P-Protokollen und Konzepte.

Eine zentrale Rolle bei der Literaturrecherche spielte die Wissensdatenbank geti2p.net. Diese Ressource erwies sich als fundamental für das Verständnis innerhalb des I2P-Ökosystems. Durch gezielte Recherchen in dieser Datenbank konnten umfassende Informationen über I2P abgerufen werden, angefangen bei grundlegenden Konzepten bis hin zu Protokoll-Spezifikationen. Dabei waren die Informationen in der Datenbank immer sehr aktuell, was z.B. das Verhalten von I2P-Router oder Spezifikationen betrifft und hat auch Informationen zu von den Entwicklern geplanten Veränderungen gegeben. Darüber hinaus wurde auch auf nicht erforschte oder unbekannte Themen in I2P aufmerksam gemacht. Die Wissensdatenbank geti2p.net wurde im Laufe der Forschungsarbeit als kontinuierliches Nachschlagewerk verwendet.

Neben geti2p.net wurde eine Bandbreite von Forschungsarbeiten studiert, von umfassenden Zusammenfassungen über I2P bis hin zu spezifischen Angriffsbeschreibungen. Vor rallem für das Verständnis von verschiedenen Angriffen waren die Arbeiten von Boss und Purtschert (Boss & Purtschert, 2022), Christoph Egger (Egger et al., 2013) und John R. Douceur (Douceur, 2002) sehr wichtig.

4.2 Skript- und Code-Analysen

Ein entscheidender Schritt für das technische Verständnis und die praktische Umsetzung bestand darin, öffentliche Code-Repositories zu untersuchen. Dieser Prozess wurde erheblich durch die Verfügbarkeit von freiem und quelloffenem Source-Code erleichtert. Die Möglichkeit, den Source-Code einzusehen und zu analysieren, spielte eine zentrale Rolle, um zu verstehen, wie abstrakte Konzepte in konkrete Implementationen umgesetzt wurden.

Die Einblicke in öffentliche Code-Repositories ermöglichten nicht nur einen tieferen Einblick in bewährte Praktiken, sondern halfen auch dabei, weniger dokumentierte Aspekte zu verstehen. Das Verständnis dieser weniger offensichtlichen und oft undokumentierten Teile des Codes war von entscheidender Bedeutung, für die Umsetzung eines I2P-Testnetzwerkes.

Dabei wurden nicht nur verschiedene I2P-Clients begutachtet, sondern auch bestehende I2P-Testnetzwerke studiert. Im Kapitel Realisierung wird die Untersuchung von I2P-Testnetzwerken genauer beschrieben.

4.3 Prototyping

Für die Entwicklung der Laborumgebung wurde eine Form von Prototyping verfolgt. Beim Prototyping wird eine Vorabversion eines Anwendungssystems entwickelt und evaluiert (Wilde & Hess, 2007).

Die Methode des Prototyping wurde iterativ angewendet, um verschiedene Prototypen eines I2P-Testnetzwerkes zu erstellen. Dieser Ansatz wurde gewählt, weil keine Literatur oder Anleitung für den Aufbau eines I2P-Testnetzwerkes mit einer aktuellen I2P-Router-Version verfügbar war. Die Schwierigkeit, Ressourcen wie Literatur oder Anleitungen zu finden, erforderte einen experimentellen Ansatz, bei dem durch die Schaffung von Prototypen praktische Erfahrungen gesammelt werden konnten.

Während des Prozesses der Prototypenerstellung wurden verschiedene Konfigurationen und Infrastrukturanpassungen vorgenommen. Dies umfasste beispielsweise Änderungen an den Adressräumen im Netzwerk oder der I2P-Router-Konfiguration, die für den Aufbau eines funktionsfähigen I2P-Testnetzwerkes relevant sind. Darüber hinaus wurden sogar verschiedene Prototypen erstellt mit verschiedenen I2P-Clients (i2pd und i2p.i2p). Die Idee, verschiedene I2P-Clients zu verwenden, lag dabei, dass bestimmte Probleme oder Herausforderungen beim Aufbau des I2P-Testnetzwerkes auftraten. Durch die Erkundung verschiedener I2P-Clients wurde versucht, spezifische Schwierigkeiten zu umgehen oder zu bewältigen, die durch die Einschränkungen oder Inkompatibilitäten eines I2P-Clients verursacht wurden.

4.4 Computer-Ressourcen

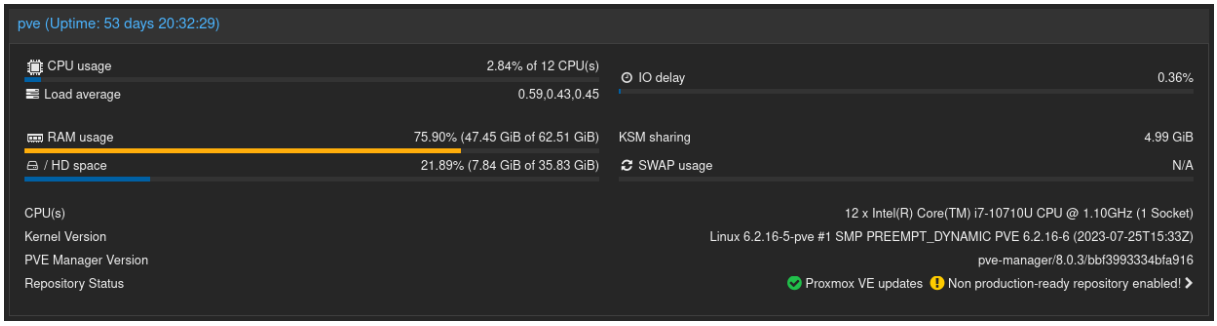

Abbildung 7: Proxmox Ressourcen Zusammenfassung

Für die Entwicklung der Prototypen wurde ein Mini-PC von Intel (Intel NUC 10th Generation) eingesetzt. Dieser Mini-PC war mit der Virtualisierungssoftware Proxmox ausgestattet. Die Verwendung dieses Systems ermöglichte die effiziente Erstellung und den Betrieb von virtuellen Maschinen sowie die Konfiguration von Netzwerken in einem experimentellen Umfeld.

4.5 Projektorganisation

In dem Verlauf der Projektphase wurden vermehrt Statusmeetings abgehalten, um den aktuellen Stand des Projekts zu besprechen. Diese Meetings dienten dazu, den Fortschritt und mögliche Vorgehensweisen zu diskutieren. Ausserdem halfen sie die Ziele des Auftraggebers besser zu verstehen. Durch diese regelmässigen Treffen konnten alle Projektbeteiligten auf dem Laufenden gehalten werden und eine effektive Kommunikation sowie Zusammenarbeit gewährleistet werden. Die Besprechungen ermöglichten es zudem, eventuelle Herausforderungen frühzeitig zu identifizieren und entsprechende Massnahmen zur Lösung zu erarbeiten. Diese strukturierten Statusmeetings trugen somit massgeblich zur erfolgreichen Umsetzung des Projekts bei.

Folgende Parteien waren am Projekt beteiligt:

- Dieter Arnold (Betreuungsperson)
- Max Suter (Experte)
- Konrad Bächler (Auftraggeber DIVA.EXCHANGE)
- Philipp Hauswirth (Student)

5 Realisierung

5.1 Entwicklung des I2P-Testnetzwerk

Im Rahmen dieser Arbeit wurde ein voll funktionsfähiges I2P-Testnetzwerk mit der I2P-Router-Implementation i2pd in Kubernetes aufgebaut. Die Implementation in Kubernetes ermöglichte eine skalierbare und flexible Bereitstellung des I2P-Testnetzwerks. Der Entwicklungsprozess bestand darin, eine Untersuchung bestehender I2P-Testnetzwerke durchzuführen, einen Entwurf zu erstellen und umzusetzen.

5.1.1 Untersuchung bestehender I2P-Testnetzwerke

Im Zuge der Untersuchung wurden verschiedene bestehende I2P-Testnetzwerk-Implementationen analysiert, um Erkenntnisse über verwendete Komponenten und mögliche Herausforderungen zu gewinnen. Die identifizierten Komponenten dieser Implementierungen bildeten die Grundlage für die eigene Umsetzung.

testnet.py von LNS Die erste Implementation die angeschaut wurde, war das Github-Repository testnet.py von LNS. Bei dieser Implementation handelte es sich um ein Python-Skript mit minimalen Abhängigkeiten und Overhead. Das Skript verwendet Linux-Namespaces, um virtuelle Netzwerkstacks zu erstellen und die I2P-Router-Prozesse darin zu starten (siehe Linux Namespaces). Für den Gebrauch wird eine Installation von i2pd oder i2p.i2p benötigt. (LNS, o. J.-b).

Das Skript verwendet eine JSON-Datei als Konfiguration. Im Repository wird ein `example.json` als Beispiel für eine Konfiguration gegeben. Das Beispiel definiert verschiedene Netzwerke mit ihren entsprechenden Subnetzen und Gateways.

```
{
    "networks": {
        "france":  {"subnet": "10.4.0.0/24",      "gateway":
        ↪  "10.4.0.1/24"},
        "germany": {"subnet": "192.168.132.0/24", "gateway":
        ↪  "192.168.132.1/24"},
        "us":      {"subnet": "10.11.0.0/24",     "gateway":
        ↪  "10.11.0.1/24"},
        "russia":  {"subnet": "192.168.74.0/24",  "gateway":
        ↪  "192.168.74.1/24"}
    }
}
```

Die Subnetze befinden sich alle in dem privaten IPv4-Adressraum (RFC1918). Somit scheinen alle Knoten mit privaten IP-Adressen zu agieren. Neben den Netzen werden auch die einzelnen Knoten bzw. I2P-Router definiert.

```
{
    "nodes": [
        {"router": "i2p", "name": "reseed1", "network": "france", "ip":
        ↪  "10.4.0.21", "floodfill": true, "reseed": true},
        {"router": "i2p", "name": "hans1", "network": "germany", "ip":
        ↪  "192.168.132.145", "floodfill": true},
        {"router": "i2p", "name": "ivan0", "network": "russia", "ip":
        ↪  "192.168.74.174", "floodfill": false},
    ]
}
```

Jedem Knoten wird ein Netzwerk und eine statische IP-Adresse zugewiesen. Ausserdem wird jedem Knoten ein boolescher Wert für `floodfill` und `reseed` zugewiesen. Der `floodfill`-Wert steuert, ob ein I2P-Router die Rolle eines Floodfills übernimmt.

```
  # im Fall von i2pd
  if "floodfill" in node and node["floodfill"] == True:
      args += ["--floodfill"]
```

Alle Knoten, welche den `reseed`-Wert auf `true` gesetzt haben, werden für das Reseeding verwendet. Im Skript ist zu sehen, wie dafür von diesen Knoten die RouterInfo in Form einer Datei in den `workspace`-Ordern kopiert wird.

```
def make_netdb_dir(conf, reseed_nodes):
    """Create reseed netDb dir and populate it with router.info's from
    ↪  reseed nodes"""
    netdb_dir = os.path.abspath(os.path.join(conf["workspace"], "netDb"))
    os.mkdir(netdb_dir)

    for n in reseed_nodes:
        ri_path = os.path.join(conf["workspace"], n["name"],
↪   "router.info")
        for x in range(1200): # two minutes max
            if is_ready(ri_path):
                ri_hash = ri_b64hash(ri_path)
                t_dir = os.path.join(netdb_dir, "r{}".format(ri_hash[0]))
                if not os.path.exists(t_dir): os.mkdir(t_dir)
                shutil.copy(ri_path, os.path.join(t_dir,
                                "routerInfo-{}.dat".format(ri_hash)))
                break
            else:
                time.sleep(0.1)
```

Beim Start jedes Routers wird der `workspace`-Ordern danach dazu verwendet, um die Network Database zu initialisieren. Dazu werden einfach die Router Info Dateien in die `netDb`-Ordner der I2P-Router gelegt.

```
shutil.copytree(os.path.join(config["workspace"], "netDb"),
                os.path.join(datadir, "netDb"))
```

Damit das ganze aufgeht, werden die Knoten in zwei Schritten gestartet. Zuerst werden alle Reseed-Knoten gestartet. Danach werden die RouterInfos von den Reseed-Knoten gesammelt, um die restlichen Knoten damit zu initialisiert.

```
# Start aller Reseed-Knoten
reseed_nodes = [n for n in conf["nodes"] if "reseed" in n and n["reseed"]
↪  == True]
for n in reseed_nodes: start_router(conf, n)

# Sammeln aller RouterInfos der Reseed-Knoten
make_netdb_dir(conf, reseed_nodes)

# Start aller restlichen Knoten
for n in conf["nodes"]:
    if n not in reseed_nodes: start_router(conf, n)
```

i2p-testnet von Mikal Villa Bei der zweiten Implementation handelte es sich um einen Blogbeitrag (0xcc.re) von Mikal Villa, welcher in der `Readme`-Datei der LNS-Implementation vorzufinden war. In dem Blogbeitrag wurde ein Bash-Skript beschrieben, welches mit Linux-Namespaces, mehrere isolierte i2pd-Prozesse startet (i2p.i2p wird nicht unterstützt) (Mikal, o. J.-a) (Mikal, o. J.-b).

Ähnlich wie bei der Implementation von LNS verwendet das Skript eine Konfiguration, die verschiedene Netzwerke und Knoten konfiguriert. In einem Beispiel werden mehrere virtuelle Netze (`vbr0` bis `vbr4`) in im privaten IPv4-Adressraum (RFC1918) definiert.

```
bridgedev vbr0 vbr0eth2 vbr0eth3 10.23.23.254/24
bridgedev vbr1 vbr1eth4 vbr1eth5 10.45.45.254/24
bridgedev vbr2 vbr2eth6 vbr2eth7 192.168.24.1/24
bridgedev vbr3 vbr3eth8 vbr3eth9 10.78.17.254/24
bridgedev vbr4 vbr4eth10 vbr4eth11 10.78.100.254/24
```

Jedem I2P-Router wird eine statische IP-Adresse in einer der virtuellen Netze zugewiesen.

```
host host01
  dev veth0 10.0.0.1/24
  route default via 10.0.0.254
  i2pdnode 15
```

Grundsätzlich ist der Ablauf sehr ähnlich, wie im Skript von LNS. Im Gegensatz zu LNS wird das Reseeding aber anders implementiert. In dem Skript von Mikal Villa wird das Program i2p-tools verwendet. i2p-tools stellt einen Webserver bereit, welcher von I2P-Routern für das Reseeding verwendet werden kann. Dieser wird neben den eigentlichen I2P-Routern zusätzlich gestartet.

```
echo "Setting up reseed"
ip netns exec $ns i2p-tools reseed \
    --numRi 20 \
    --key $RESEED_DIR/${SIGNER_FNAME}.pem \
    --netdb $RESEED_DIR/netDb \
    --tlsHost $RESEED_DIR/${RESEEDSERVER_IP} \
    --tlsCert $RESEED_DIR/${RESEEDSERVER_IP}.crt \
    --tlsKey $RESEED_DIR/${RESEEDSERVER_IP}.pem \
    --signer $RESEED_DIR/${SIGNER_FNAME} &
cd -
```

Für den Betrieb dieses Reseed-Webservers werden alle RouterInfo-Dateien der Router gesammelt und dem Webserver bereitgestellt. Beim Start der I2P-Router bzw. der i2pd-Prozesse wird dann einfach nur noch die URL dieses Reseeding-Webservers übergeben.

```
I2PDWRAPPER=$(cat <<EOF
#!/usr/bin/env bash\n
ip netns exec \$2 i2pd \
  --datadir=$DATADIR/testnode\$1 \
  --log stdout \
  --floodfill \
  --ipv4=1 \
  --host=\$3 \
  --ifname4=veth0 \
  --pidfile=$PIDDIR/router\$1.pid \
  --port=4764 \
  --reseed.urls=$RESEEDSERVER | tee -a $LOGDIR/router\$1.log >>
↪  $LOGDIR/all-routers.log &
EOF
)
```

docker-testnet von LNS Als Letztes wurde noch ein weiteres Github-Repository (docker-testnet) von LNS untersucht. Bei dieser Implementation handelt es sich um ein Python-Skript, welches per Docker, I2P-Router als Container startet. Für die Container wird das von den i2pd-Entwicklern bereitgestellte Container-Image purplei2p/i2pd verwendet (LNS, o. J.-a).

Im Gegensatz zu den anderen beiden Implementationen wird beim Start nur ein einziges Netzwerk definiert und erstellt. Dafür wird im `ctl.py` die `create_network()` Methode ausgeführt.

```
# ctl.py
class TestnetCtl(object):

    def start(self, args):
        """Start testnet"""
        if not self.testnet.NODES:
            self.testnet.create_network()
            cid = self.testnet.run_i2pd(args=' --reseed.threshold=0 ',
                    floodfill=True, with_seed=False)
            print(cid)
            self.testnet.make_seed(cid)

            if args.floodfills: self._batch_run(args.floodfills, True)
            if args.nodes:      self._batch_run(args.nodes,      False)
        log.info("*** Testnet is running")
```

Die `create_network()` Methode erstellt dann im Hintergrund ein internes Docker-Netzwerk.

```
# testnet.py
class Testnet(object):

    def create_network(self):
        """Create isolated docker network"""
        self.cli.networks.create(self.NETNAME, driver="bridge",
        ↪  internal=True)
```

Weder die Dokumentation, noch das Python-Skript im Github-Repository lassen darauf schliessen, dass der Docker-Adress-Pool verändert wird. Somit handelt es sich bei dem erstellten Docker-Netzwerk um ein Private-Range-Netzwerk (RFC1918). Beim Start der Container ist zu sehen, dass jeder Container dem gleichen Docker-Netzwerk zugeordnet wird.

```
 # testnet.py: Testnet.run_i2pd()
 cont = self.cli.containers.run(self.I2PD_IMAGE, i2pd_args,
         volumes=["{}:/seed.zip".format(self.SEED_FILE)],
         labels=labels, network=self.NETNAME, detach=True, tty=True)
```

Was bei der `run()` Methode auffällt, ist, dass beim Start der Container, keine spezifische Netzwerk-Konfiguration mitgegeben wird. Standardmässig erhält der Container eine IP-Adresse für jedes Docker-Netzwerk, mit dem er verbunden ist. Der Docker-Daemon führt eine dynamische Subnetz- und IP-Adresszuweisung für Container durch (*Networking overview | Docker Docs*, o. J.). Das heisst, dass die Container bei jedem Start eine neue IP-Adresse erhalten.

Neben der Netzwerk-Konfiguration unterscheidet sich auch das Reseeding zu den anderen Implementationen. In diesem Fall wird dem Container beim Start ein Pfad zu einer Zip-Datei für das Reseeding übergeben.

```
i2pd_args += " --reseed.zipfile=/seed.zip "
```

Das i2pd-Argument `--reseed.zipfile` ermöglicht die Übergabe eines Pfades zu einer lokalen Zip-Datei. Diese Zip-Datei wird aus der Router Info eines spezifischen Containeres im Testnetzwerk über die `make_seed()` Methode erstellt.

```
    def make_seed(self, cid):
        """creates a zip reseed file"""
        ri_path = "/home/i2pd/data/router.info"

        with tempfile.TemporaryFile() as fp:
            while True:
                try:
                    arc_data =
↪   self.NODES[cid].container.get_archive(ri_path)[0].read()
                    break
                except:
                    time.sleep(0.1)

            fp.write(arc_data)
            fp.seek(0)
            ri_file = tarfile.open(fileobj=fp, mode='r')\
                    .extractfile("router.info").read()
            tf = tempfile.mkstemp()[1]
            with open(tf, 'wb') as f: f.write(ri_file)
            zf = zipfile.ZipFile(self.SEED_FILE, "w")
            zf.write(tf, "ri.dat")
            zf.close()
            os.remove(tf)
```

In der `start()` Methode der Testnet-Klasse ist zu sehen, dass für den Start des Testnetzwerkes, zuerst ein einziger Floofill-Router gestartet wird. Die Router Info dieses Floofill-Routers wird dann per `--reseed.zipfile` an die weiteren I2P-Router verteilt, die danach gestartet werden.

```
# ctl.py
class TestnetCtl(object):

    def start(self, args):
        """Start testnet"""
        if not self.testnet.NODES:
            self.testnet.create_network()
            cid = self.testnet.run_i2pd(args=' --reseed.threshold=0 ',
                    floodfill=True, with_seed=False)
            print(cid)
            self.testnet.make_seed(cid)

            if args.floodfills: self._batch_run(args.floodfills, True)
            if args.nodes:      self._batch_run(args.nodes,      False)
        log.info("*** Testnet is running")
```

5.1.2 Erkenntnisse aus der Untersuchung

Die drei Implementationen weisen Gemeinsamkeiten hinsichtlich der Isolierung zwischen den I2P-Router-Prozessen auf. Alle nutzen virtuelle Netzwerke, die sich im privaten Adressraum befinden. Zudem unterstützen sämtliche Implementationen i2pd als I2P-Router-Implementation. Interessanterweise variiert die Umsetzung des Reseedings deutlich zwischen den einzelnen Implementationen.

Wie in der Arbeit von Boss und Purtschert bereits erwähnt, hat i2pd im Januar 2021 eine Prüfung implementiert, die I2P-Router im privaten Adressraum als ungültig markiert (Boss & Purtschert, 2022). Eine Inspektion der i2p.i2p I2P-Router-Implementation hat ergeben, dass i2p.i2p eine ähnliche Prüfung in Form einer Blocklist eingeführt hat, welche standardmässig privaten und reservierten IPv4- und IPv6-Adressraum blockiert (*i2p/i2p.i2p*, o. J.). Alle untersuchten I2P-Testnetzwerke sind entstanden, bevor die Prüfung von den verschiedenen I2P-Router-Implementationen eingeführt wurde. Dies ist einer der Gründe wieso keine der untersuchten I2P-Testnetzwerke mit den aktuellen Versionen der I2P-Router-Implementationen funktionieren.

5.1.3 Aufbau eines eigenen I2P-Testnetzwerkes

Basierende auf den Erkenntnissen aus den durchgeführten Untersuchungen, den identifizieren Unterschieden und der Versionsinkompatibilität mit den neusten I2P-Router-Implementationen, wurde entschieden eine eigene I2P-Testnetzwerk-Implementation zu entwickeln. Dafür wurden die Mindestanforderungen an ein I2P-Testnetzwerk definiert und ein knapper Entwurf für einen Prototyp erstellt.

Entwurf I2P-Testnetzwerk

Für den Entwurf wurden folgende Anforderungen definiert:

- **Router-Implementation:** Das I2P-Testnetzwerk soll mit i2pd oder i2p.i2p umgesetzt werden. Diese beiden Router-Implementationen sind am weitesten verbreitet und am besten unterstützt. Darüber hinaus verfügen beide über ein offizielles Container-Image.
- **Funktionsfähig:** Tunnel zwischen den I2P-Routern sollen, wie im normalen I2P-Netzwerk automatisch aufgebaut werden und die I2P-Router sollen ohne Probleme miteinander Kommunizieren sollen.
- **Eigenständig:** Das I2P-Testnetzwerk soll ohne externe I2P-Router oder Reseed-Server aufgebaut werden können. Also das keine Abhängigkeit zu Systemen ausserhalb des Testnetzwerkes besteht.
- **Reproduzierbar & Skalierbar:** Das I2P-Testnetzwerk soll reproduzierbar und skalierbar sein. Es soll ohne grossen Aufwand auf einer neuen Infrastruktur aufgesetzt oder migriert werden können.
- **Netzwerkverkehr:** Jeglicher Netzwerkverkehr im I2P-Testnetzwerk soll mit relativ wenig Aufwand oder speziellen Netzwerkgeräten aufgezeichnet werden können.

Mit Hilfe der Anforderungen, wurde dann eine erste Idee ausgearbeitet.

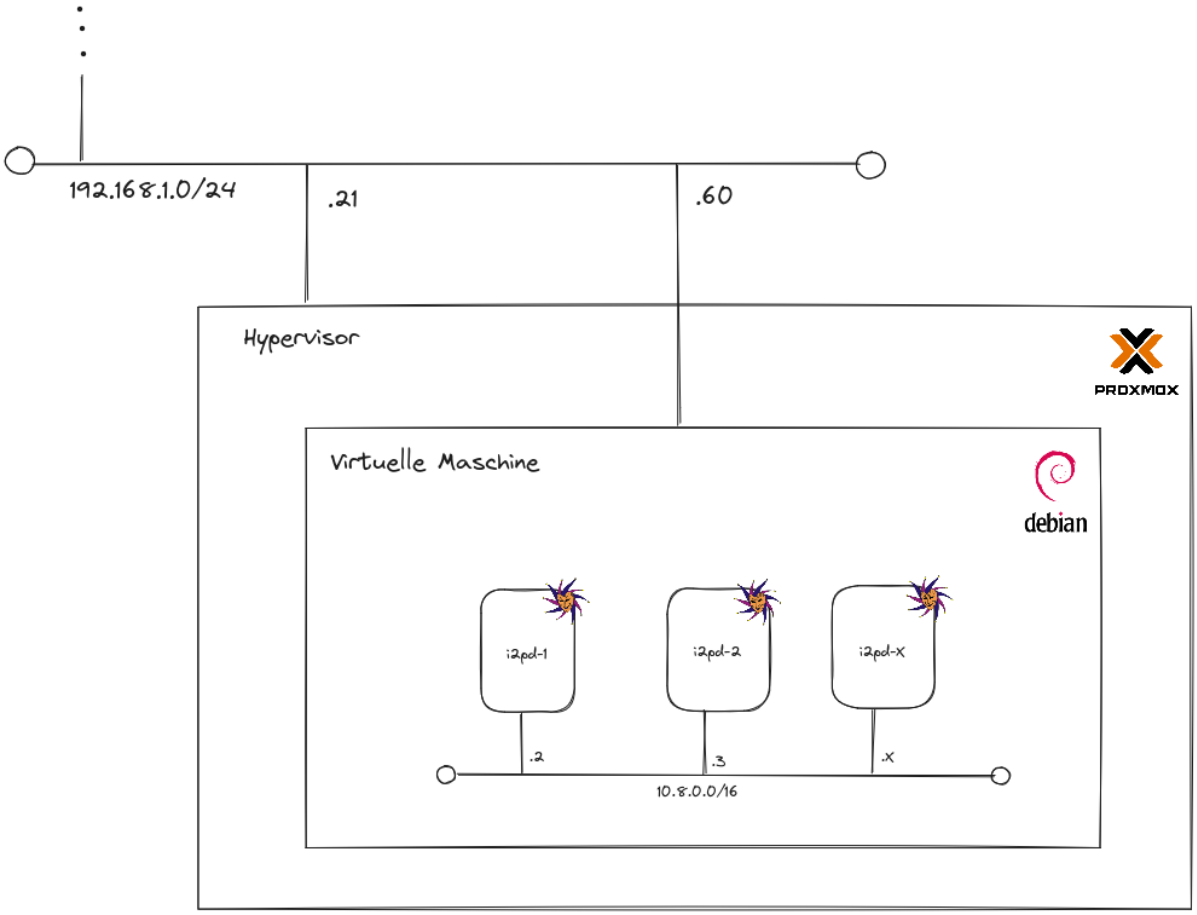

Abbildung 8: Grober Entwurf I2P-Testnetzwerk

Wie bei den untersuchten I2P-Testnetzwerken soll das Netzwerk zwischen den I2P-Routern völlig virtuell Ablaufen um das Aufzeichnen des Verkehrs sicherzustellen. Des Weiteren soll die Entwicklung auf einem bestehenden Hypervisor in Form von einer oder mehreren virtuellen Maschinen erfolgen.

5.1.3.1 Vorbereitung Für die Entwicklung wurde eine K3s Kubernetes-Cluster bereitgestellt. Die Installationsschritte dafür wurden in einem öffentlichen Github-Repository beschrieben, welches in Anhang B verlinkt ist. Zusammengefasst wurde als Vorbereitung ein Kubernetes-Cluster auf einer Debian-Plattform mithilfe von k3s eingerichtet.

5.1.3.2 Resultat Das Resultat der Entwicklung ist ein Helm-Chart, das die reibungslose Installation eines i2pd Kubernetes-Deployments ermöglicht. Dieses Helm-Chart beinhaltet eine umfassende Konfiguration, die nicht nur die Bereitstellung von i2pd selbst ermöglicht, sondern auch die Integration eines Nginx-Deployments welches über i2pd verwendet werden kann. Darüber hinaus werden verschiedene Kubernetes Service-Objekte bereitgestellt, um den Zugriff auf einzelne Dienste wie den HTTP-Proxy oder die i2pd-Konsole zu erleichtern. Um den gesamten Aufbau eines I2P-Testnetzwerks in Kubernetes durchzuführen, wird ein begleitendes Skript bereitgestellt. Dieses Skript automatisiert den Prozess der Konfiguration und Bereitstellung des i2pd Helm-Charts sowie der zugehörigen Dienste. Die Kombination aus dem Helm-Chart und dem bereitgestellten Skript ermöglicht es, in kurzer Zeit ein funktionsfähiges I2P-Testnetz aufzusetzen.

Das Helm-Chart ist wie folgt aufgebaut:

```
helm/
`-- i2pd-chart
    |-- charts
    |-- Chart.yaml
    |-- setup.sh
    |-- templates
    |   |-- configmap-i2pdconf.yaml
    |   |-- configmap-reseedzip.yaml
    |   |-- configmap-tunnels.yaml
    |   |-- deployment.yaml
    |   |-- _helpers.tpl
    |   |-- ingress.yaml
    |   |-- nginx.yaml
    |   |-- NOTES.txt
    |   |-- pvc.yaml
    |   |-- serviceaccount.yaml
    |   |-- service-console.yaml
    |   |-- service-httpproxy.yaml
    |   |-- service-i2pcontrol.yaml
    |   |-- service-sam.yaml
    |   |-- service-socksproxy.yaml
    |   `-- tests
    |       `-- test-connection.yaml
    `-- values.yaml
```

Das Skript (`setup.sh`) und Helm-Chart sind Teil eines öffentlichen Github-Repository, welches in Anhang B verlinkt ist.

5.1.4 Herausforderungen

Die Entwicklung bis zum ersten funktionsfähigen Prototypen hat beträchtliche Zeit in Anspruch genommen. Während dieses Prozesses stellte sich heraus, dass die Logs der I2P-Router (i2pd und i2p.i2p) nur begrenzte Informationen zum aktuellen Status bereitstellen, was die Fehlersuche und Optimierung erschwerte. Um dieses Problem zu überwinden, wurde zeitweise der Source-Code der i2pd-Implementation durchsucht, um tiefergehende Einblicke zu erhalten. Das Verständnis für verschiedene Verhaltensweisen konnte erst erlangt werden, nachdem der Source-Code genauer unter die Lupe genommen wurde. Der Blick in den Source-Code erwies sich somit als entscheidend, um tiefergehende Einblicke in das System zu gewinnen.

In bestimmten Phasen der Entwicklung wurden Anstrengungen unternommen, geeignete Kanäle zu finden, um Unterstützung und Hilfe zu erhalten. Hierbei wurden Plattformen wie Reddit und das I2P-Forum aufgesucht, jedoch ohne grossen Erfolg. Nach der i2pd-Dokumentation stand ein offizieller IRC-Server, für den direkten Kontakt mit den i2pd-Entwicklern zur Verfügung. Jedoch zeigte sich bei der ersten Verbindung, dass einige Räume auf dem IRC-Server nicht den professionellen Standards entsprachen und anhand ihrer Raumnamen tendenziell problematisch, geschmacklos oder sogar rechtlich fragwürdig waren. Die Kontaktaufnahme über IRC wurde aufgrund dieser Feststellung nicht verwendet.

Der Entwicklungsprozess war geprägt von mehrfachem “Trial and Error”, wobei verschiedene Ansätze ausprobiert und wiederholt überarbeitet wurden, um den bestmöglichen Lösungsweg zu finden. Diese iterative Vorgehensweise ermöglichte es, Herausforderungen zu überwinden und den Prototypen schliesslich in einen funktionsfähigen Zustand zu versetzen.

Während der Entwicklung wurden folgende grösseren Herausforderungen angetroffen.

Privater Adressraum

Beide I2P-Router-Implementationen i2pd und i2p.i2p verwenden standardmässig eine Prüfung, die es verhindert mit I2P-Routern im privaten oder reservierten Adressraum zu kommunizieren. Anders ausgedrückt akzeptieren die I2P-Router-Implementationen nur RouterInfos welche eine öffentliche IP-Adresse hinterlegt haben.

Initial wurde probiert, diese Prüfung im i2pd per Konfiguration zu deaktivieren. i2pd verfügt über eine versteckte Konfiguration `reservedrange`.

```
/* libi2pd/Config.pp */
("reservedrange", bool_switch()->default_value(true), "Check remote RI for
↪  being in blacklist of reserved IP ranges (default: enabled)")
```

Diese Konfiguration wird in der i2pd-Dokumentation nicht erwähnt und es wurde neben der Optionen-Beschreibung im Programmcode (`libi2pd/Config.pp`) keine weiteren Informationen dazu gefunden (*Configuring - i2pd documentation*, o. J.). Da alle I2P-Router im selben Netzwerk sind und zwischen den I2P-Routern kein NAT stattfindet, wurden ausserdem die Konfigurationsoptionen `nat` und `upnp.enabled` angepasst. Diese sind wie folgt im Programmcode beschrieben.

```
/* libi2pd/Config.pp */
("nat", bool_switch()->default_value(true), "Should we assume we are
↪  behind NAT? (default: enabled)")
("upnp.enabled", value<bool>()->default_value(upnp_default), "Enable or
↪  disable UPnP: automatic port forwarding")
```

Im Verlauf der Entwicklung and nach längerem Debugging über mehrere Tage wurde aber festgestellt, dass selbst mit `reservedrange` auf `false` gesetzt, das Reseeding und der Tunnel-Aufbau mit I2P-Routern in dem privaten Adressraum fehlschlägt. Dies hatte zur Folge, dass im Endeffekt das Vortäusche eines öffentlichen IP-Adressbereichs nötig war. Konkret wurde jedem I2P-Router eine öffentliche IP-Adresse im virtuellen Netzwerk zugewiesen und folgende i2pd-Konfiguration verwendet.

Bootstrapping des I2P-Testnetzwerkes (Reseeding)

Für das Bootstrapping bzw. das Reseeding der I2P-Router in i2pd musste immer eine Zip-Datei mit den aktuellen Routern im Netzwerk erstellt und beim Start der Router übergeben werden. Dies führte dazu, dass die Startsequenz des I2P-Testnetzwerkes mehrere Schritte umfasst:

1. Alle I2P-Router-Container werden gestartet.
2. Alle RouterInfos werden gesammelt.
3. Die RouterInfos werden in eine Zip-Datei verpackt.
4. Alle I2P-Router-Container werden gestoppt.
5. Die I2P-Router-Container werden erneut gestartet, und die Zip-Datei wird beim Start eingehängt.
6. Nachdem die Zip-Datei erfolgreich eingehängt wurde, erfolgt das Reseeding durch die I2P-Router. Dabei werden die in der Zip-Datei enthaltene RouterInfos verwendet um die NetDB zu initialisieren.

Im Verlauf dieser Sequenz mussten einige Dinge beachtet werden für den erfolgreichen Aufbau.

- Beim initialen Start der I2P-Router-Container musste sichergestellt werden, dass alle wichtigen Dateien wie RouterInfo oder Schlüssel in einen Pfad geschrieben werden, welcher über das Leben des Containers hinweg persisiert. Ansonsten würden beim nächsten Start des Containers die Dateien neu generiert werden und sich die Router-Identität mit RouterInfo ändern und das Reseeding fehlschlagen.
- Für den späteren Tunnel-Aufbau und End-to-End-Verbindungen musste eine gewisse Anzahl von I2P-Routern aktiv sein. In dem Projekt hat eine Netzwerkgrösse von mindestens 16 I2P-Routern sehr gut funktioniert.
- Beim erneuten Starten der I2P-Router-Container musste sichergestellt sein, dass der Container dieselbe IP-Adresse erhält wie zuvor. Die IP-Adresse, mit welcher ein I2P-Router erreichbar ist, ist Teil der RouterInfo (*Common structures Specification - I2P*, o. J.) und sollte für ein erfolgreiches Reseeding übereinstimmen.

Letzteres hat in der Umsetzung für längere Zeit zu Problemen geführt. Standardmässig werden beim Start der Container in Docker oder Kubernetes dynamische IP-Adressen verteilt und statische IP-Adressen müssen explizit konfiguriert werden. Interessanterweise wurde in der docker-testnet Implementation von LNS dieses Problem, wie es scheint nicht angetroffen, da beim Start des I2P-Containers, welcher für das Reseeding verwendet wird, keine statische IP-Adresse konfiguriert wird. Es konnte nicht verifiziert werden aber es wird angenommen, das bei der Vergabe der IP-Adressen in Docker die Reihenfolge eine Rolle spielt. Also, wenn die Container in derselben Reihenfolge gestartet werden, diese auch die gleichen IP-Adressen erhalten. Im Gegensatz dazu wurden in dem entwickelten I2P-Testnetzwerk in Kubernetes die IP-Adressen anfangs “willkürlich” vergeben und eine explizite Konfiguration wurde benötigt um dieses Problem zu lösen

5.2 Datenerhebung im I2P-Testnetzwerk

Zur Überwachung und Analyse der I2P-Aktivitäten sowie Angriffen wurden verschiedene Ansätze diskutiert, um umfassende Datenerhebung zu ermöglichen. Konkret wurden folgende Daten gesammelt:

1. **Logs:**

Um das Verhalten der einzelnen I2P-Router besser zu verstehen wurden alle möglichen Logs gesammelt. Dazu wurde die i2pd-Konfiguration angepasst.

```
log = stdout
loglevel = debug
```

Einerseits wurden die Logs der I2P-Router an stdout umgeleitet. Logs in Containern werden häufig der Einfachheit und Kompatibilität mit Containerisierungsplattformen wegen an stdout gesendet. In Kubernetes erlaubt das Senden an stdout, dass die I2P-Router-Logs über kubectl abgerufen werden können. Ausserdem wurde das tiefste Log-Level debug konfiguriert um so viel Informationen wie möglich zu sammeln

2. **Netzwerkverkehr:**

Für das Aufzeichnen des Netzwerkverkehrs wurde tcpdump verwendet. Die Aufzeichnung des Netzwerkverkehrs mittels tcpdump trug dazu bei, detaillierte Einblicke in die Kommunikation innerhalb des I2P-Netzwerks zu erhalten. Da der gesamte Verkehr virtualisiert war und der Kubernetes-Cluster sich nur über ein einzelnes System erstreckte, war eine zentralisierte Überwachung und Aufzeichnung möglich. Durch die gezielte Anwendung von tcpdump mit einem passenden Filter auf das I2P-Netzwerk konnte jeglicher Verkehr im I2P-Testnetzwerk gesammelt werden.

```
sudo tcpdump -nnni any net "123.8.0.0/16" -w traffic.pcap
```

In diesem Fall entspricht `123.8.0.0/16` dem IPv4-Adressraum, welcher für die I2P-Router-Container verwendet wurde. Wie bereits erwähnt war das Vortäuschen eines öffentlichen IP-Adressbereichs nötig. Der aufgezeichnete Traffic wurde dann mit Wireshark genauer untersucht.

3. **API-Daten:**

Neben den Logs und des Netzwerkverkehrs wurde eine Möglichkeit gesucht, um Statusinformationen der I2P-Router zu erhalten. Dafür wurde die von den I2P-Routern zur Verfügung gestellte i2pcontrol Schnittstelle verwendet. Diese musste im Fall von i2pd zuerst über die Konfiguration aktiviert werden.

```
    [i2pcontrol]
    enabled = true
    address = 0.0.0.0
    port = 7650
```

Die Schnittstelle i2pcontrol verwendet JSON-RPC und ermöglicht nach einer initialen Authentifizierung die Abfrage von Informationen. Dafür wurde ein kleines Python-Skript geschrieben. Für die Abfrage von grundlegende Informationen wie Betriebszeit, Version oder die Anzahl aktiver Knoten wurde beispielsweise die i2pcontrol-Methode mit dem Namen *RouterInfo* verwendet. Die Funktion für dieses Beispiel wurde wie folgt implementiert.

```
def getRouterInfo(url, token):
    payload = {
        "method": "RouterInfo",
        "params": {
            "i2p.router.uptime": None,
            "i2p.router.net.status": None,
            "i2p.router.netdb.knownpeers": None,
            "Token": token,
        },
        "jsonrpc": "2.0",
        "id": 0,
    }
    return req(url, payload)
```

Leider stellte sich heraus, dass sich die i2pcontrol-Schnittstellen stark zwischen den I2P-Clients unterscheiden und die Schnittstelle in i2pd sehr wenig Informationen liefert. Somit konnten keine zusätzlichen Informationen gewonnen werden, welche nicht bereits über die Logs oder den Netzwerkverkehr bekannt waren. Aus diesem Grund wurde dieser Ansatz nicht weiter verfolgt.

5.3 Angriffs-Simulation

Nach dem Aufbau der Laborumgebung und der Fertigstellung des I2P-Testnetzwerkes, wurde mit den Angriffsideen weiter gefahren. Dafür wurde Netzwerkverkehr in Form von Verkehr zu I2P-Webseiten generiert. Das I2P-Testnetzwerk wurde so konzipiert, dass jeder I2P-Router einen eigenen Webserver hat. Dieser Webserver ist über I2P von jedem I2P-Router erreichbar. Zusätzlich bestand die Möglichkeit, dass jeder I2P-Router als HTTP-Proxy fungieren und somit als Eintrittspunkt in das I2P-Testnetzwerk verwendet werden konnte. Dadurch bestand die Möglichkeit Verkehr zwischen allen I2P-Routern im Testnetzwerk zu generieren.

Mit dem Tool curl konnte so beispielsweise Verkehr in einem Skript simuliert werden.

```
http_proxy="http://192.168.1.60:30223" curl
↪  "http://gnxag3t6iuuzmb5wlmpkgb44sixkrwnox3opk3djdlko5cfwq2xa.b32.i2p"
```

Bei der HTTP-Proxy-Adresse `192.168.1.60` in diesem Beispiel handelte es sich um die IP-Adresse der virtuellen Maschine, die den Kubernetes-Cluster betrieben hat. Der Port `30223` war ein Port-Forwarding, welches durch ein Kubernetes-Service-Object erstellt wurde.

```
kubectl -n i2pd get svc | grep i2pd-4
i2pd-4-nginx                    ClusterIP      10.43.229.131   <none>
↪  8080/TCP          12d
i2pd-4-i2pd-chart-console       LoadBalancer   10.43.18.247    <pending>
↪  7070:31610/TCP    12d
i2pd-4-i2pd-chart-httpproxy     LoadBalancer   10.43.4.118     <pending>
↪  4444:30223/TCP    12d
```

So wurde automatisch jeglicher Verkehr auf Port `30223` an den HTTP-Proxy Port `4444` von `i2pd-4` geleitet. Mit diesem Ansatz wurde ein Skript erstellt, welches durchgehend Verkehr generiert.

Mit der Laborumgebung und dem simuliertem Netzwerkverkehr wurde dann konkret an den Ideen Tunnelkontrolle und Mustererkennung Datenverkehr gearbeitet.

Tunnelkontrolle

Die gesammelten Daten (Logs, Netzwerkverkehr und API-Daten) aus dem simulierten Netzwerkverkehr waren nicht ausreichend um die beschriebene Zuordnung in der Tunnelkontrolle durchzuführen. Unter anderem fehlten Informationen zu Tunnel-Aufbau und Versand von Tunnel-Nachrichten. Darunter zählen beispielsweise die I2NP-Message Header-Informationen die den nächsten Hop in einem Tunnel definieren.

Damit diese Informationen von den einzelnen I2P-Routern hätte gesammelt werden können, wären Änderungen an der i2pd-Implementation vonnöten gewesen. Aufgrund der fortgeschrittenen Zeit und

geschätzen Aufwand welcher für die Anpassung an dem i2pd-Source-Code vonnöten gewesen wäre, wurde die Idee zur Tunnelkontrolle nicht weiter geführt.

Das bedeutet das diese Idee grundsätzlich immer noch als möglich betrachtet wird, aber das ein Angreifer eine "manipulierte" bzw. erweiterte I2P-Router-Implementation benötigt, um die Zuordnung durchzuführen.

Mustererkennung Datenverkehr

Die zweite Angriffsidee hat sich mit der Mustererkennung des Datenverkehrs beschäftigt. Diese Idee bezieht sich nur auf den reinen Datenverkehr. Dazu werden keine I2P-Router-Logs oder Router interne Informationen benötigt. Jedoch muss der Verkehr zwischen den I2P-Routern mitgeschnitten werden können.

Für diesen Angriff wurde ein sehr kleines I2P-Netzwerk mit 16 I2P-Routern gestartet und es automatisch Verkehr zwischen den Knoten generiert. Ausserdem wurde der ganze Verkehr für 20 Sekunden zwischen den Knoten aufgezeichnet. In diesem Fall kontrollierte der Angreifer einen der I2P-Router (`i2pd-13` mit IP-Adresse `123.8.0.14`) im Netzwerk und verwendete diesen um eine HTTP-Request an einen I2P-Dienst zu schicken. Genauer gesagt wurde eine kleine GET-Request an die Adresse `gnxag3t6iuuzmb5wlmpkgb44sixkrwnox3opk3djdlko5cfwq2xa.b32.i2p` gesendet.

Die HTTP-Request beinhaltete folgende Daten:

```
GET http://gnxag3t6iuuzmb5wlmpkgb44[...]ko5cfwq2xa.b32.i2p/ HTTP/1.1
Host: gnxag3t6iuuzmb5wlmpkgb44sixkrwnox3opk3djdlko5cfwq2xa.b32.i2p
User-Agent: curl/8.0.1
Accept: */*
Proxy-Connection: Keep-Alive
```

Auf diese Anfrage hat der Angreifer folgende Antwort bzw. HTTP-Response erhalten.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 17
Connection: close
Proxy-Connection: close

Hello from i2pd-3
```

Der Webserver hinter der I2P-Adresse hat also mit der Nachricht "`Hello from i2pd-3`" geantwortet. In einem echten Netzwerk würde ein I2P-Webseiten-Betreiber natürlich nie den Router preisgeben, welche diese I2P-Webseite zur Verfügung stellt. Dies würde die Zuordnung des LeaseSet auf die RouterInfo ermöglichen und den Betreiber deanonymisieren. Für dieses Szenario ist der Inhalt der HTTP-Request

und HTTP-Response nicht relevant. Es wird aber davon ausgegangen, dass der Angreifer für diese HTTP-Request die genaue Zeit und Dauer festgehalten hat.

Somit sind folgende Informationen dem Angreifer bekannt:

1. Der I2P-Router, welcher die HTTP-Request versendet hat: `i2pd-13` mit IP-Adresse `123.8.0.14`
2. Wann die HTTP-Request gesendet und eine Antwort zurückgekommen ist:

```
HTTP-Request:  frame.time="2023-11-22 16:00:40.517477+0100"
HTTP-Response: frame.time="2023-11-22 16:00:40.530469+0100"
```

3. Somit auch wie lange das gedauert hat: `13 ms` (gerundet)

Diese Informationen wurden dazu verwendet, den Netzwerkverkehr in Wireshark zu analysieren. Dazu wurde der aufgezeichnete Verkehr in Wireshark geöffnet und folgender filter angewendet.

```
(frame.time >= "2023-11-22 16:00:40.517477+0100"
and frame.time <= "2023-11-22 16:00:40.530469+0100")
&& !(tcp.analysis.retransmission or
tcp.analysis.fast_retransmission or
tcp.analysis.duplicate_ack)
```

Dieser Filter verursacht einerseits, dass nur Pakete zwischen der HTTP-Request und der HTTP-Response angezeigt werden. Andererseits, dass wiederholte oder duplizierte TCP-Pakete ignoriert werden.

Im Stand der Technik wurde erklärt, dass Tunnel für den Transport für Übertragungen nicht immer neu erstellt werden und einfach aus einem Tunnel Pool gewählt werden. Aus diesem Grund kann für die Verfolgung der einzelnen Hops im Tunnel nicht auf den TCP- oder UDP-Verbindungsaufbau geschaut werden, da die Tunnel zu diesem Zeitpunkt bereits existieren. Es muss aber Verkehr fliessen, welcher die HTTP-Request und HTTP-Response beinhaltet. Zudem muss der Verkehr in einer gewissen Reihenfolge fliessen. Anders gesagt, wenn z.B. ein Outbound-Tunnel mit den Knoten `A-D` besteht und Verkehr von `A` nach `D` fliesst, muss logischerweise nach dem Verkehr zwischen `A` und `B`, zuerst Verkehr zwischen `B` und `C` fliessen, bevor Verkehr von `C` nach `D` fliessen kann. Also es können nicht einfach Knoten übersprungen werden.

Aus diesem Grund wurde ein Flow-Graph mit Wireshark generiert, um das Fliessen der Daten zu verfolgen.

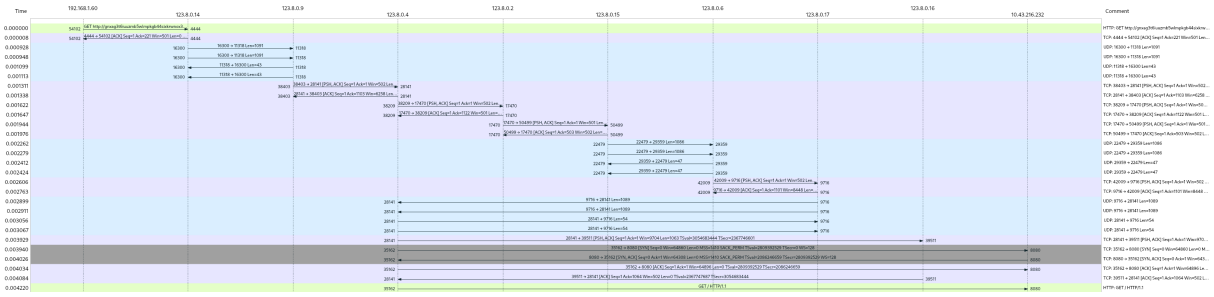

Abbildung 9: Wireshark Flow-Graph (originales Bild ist in Anhang B verlinkt)

Über diesen Flow-Graph lassen sich, ein einem Netzwerk mit sehr wenig Verkehr, die Knoten in den Tunneln relativ einfach bestimmen. In diesem Flow-Chart sind auch die eigentlichen HTTP-Pakete zwischen dem Angreifer dem von dem Angreifer kontrollieren I2P-Router und zwischen dem Ziel-I2P-Router und dem Ziel-Webserver. In einem realen Umfeld würde natürlich der Angreifer den Verkehr zwischen Ziel I2P-Router und Ziel-Webserver nicht sehen.

Aus dem Flow-Graph lässt sich nun folgender Fluss ableiten:

1. Angreifer: `123.8.0.14`
2. Outbound-Tunnel: `123.8.0.9 -> 123.8.0.4 -> 123.8.0.2`
3. Inbound-Tunnel: `123.8.0.15 -> 123.8.0.6 -> 123.8.0.17`
4. Ziel: `123.8.0.4`

Dies lässt sich in diesem Szenario relativ sicher sagen, da die Reihenfolge und Anzahl Hops andere Varianten ausschliesst. Konkret wäre das Ziel in diesem Szenario nun deanonymisiert. Da es sich hier aber um ein Szenario handelt, welches relativ weit weg von einem realen Umfeld ist, wurde ein zweites Szenario durchgespielt.

Im zweiten Szenario wurden wieder 16 I2P-Router aufgesetzt. Dieses Mal wurde aber die Verkehrsgeneration angepasst. Unter anderem wurden sehr viel mehr Verbindungen aufgebaut zudem wurden vor allem Verbindungen gleichzeitig aufgebaut. Es wurden also die `curl`-Befehle im Skript nicht nur nacheinander abgesetzt, sondern auch gleichzeitig und in verschiedenen Intervallen.

Dies hatte zur Folge, dass innerhalb von 20 Sekunden nicht nur viel mehr Verkehr generiert wurde, sondern, dass der Flow-Graph nicht mehr so klar zu lesen ist. Zur Veranschaulichung wurde ein Graph aus den IP-Paketen erstellt (Skript dazu ist im Anhang B verlinkt).

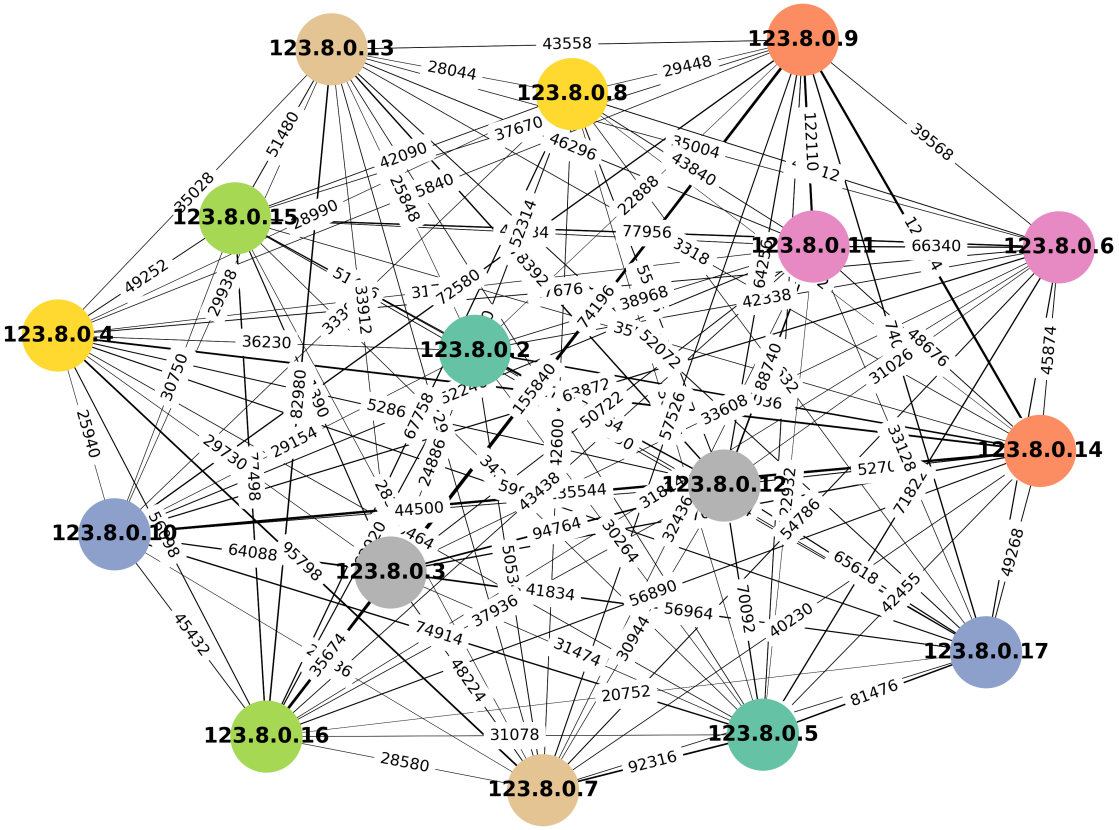

Abbildung 10: Verbindungen zwischen I2P-Routern (20 Sekunden)

Für diesen Graphen wurden nur 6 HTTP-Requests von 3 verschiedenen Quellen an 3 verschiedene Ziele gesendet. Trotzdem ist zu sehen, wie eigentliche alle 16 I2P-Knoten miteinander kommuniziert haben. Die farbigen Kreise identifizieren jeweils einen I2P-Router und die Kanten eine Verbindung. Die Zahl in der Mitte der Kanten beschreibt die Grösse der IP-Pakete in Bytes und die Kantendicke den relativen Unterschied zu den anderen Kanten (dickere Kante = mehr Verkehr).

Mit diesen Erkenntnissen wurde versucht, Ideen zu entwickeln, die Verbindungen zwischen den I2P-Routern für eine spezifische Verbindung ausschliessen lassen. Unter anderem wurde überprüft, ob eine Kombination von **Brute-Force-Angriff** und **Verkehrsanalyse** (vor allem mit dem Spielen von verschiedenen Übertragungsgrössen) Verbindungen für einen Tunnel ausgeschlossen werden können. Aus zeitlichen Gründen konnten, aber keine weiteren Schlüsse daraus gezogen und auch nicht verifiziert werden.

6 Evaluation und Validation

6.1 I2P-Testnetzwerk

Die Validation des I2P-Testnetzwerkes bestand grundsätzlich darin die Funktionalität und Verhalten zu prüfen. Der I2P-Client i2pd bietet dafür bereits ein paar Anhaltspunkte.

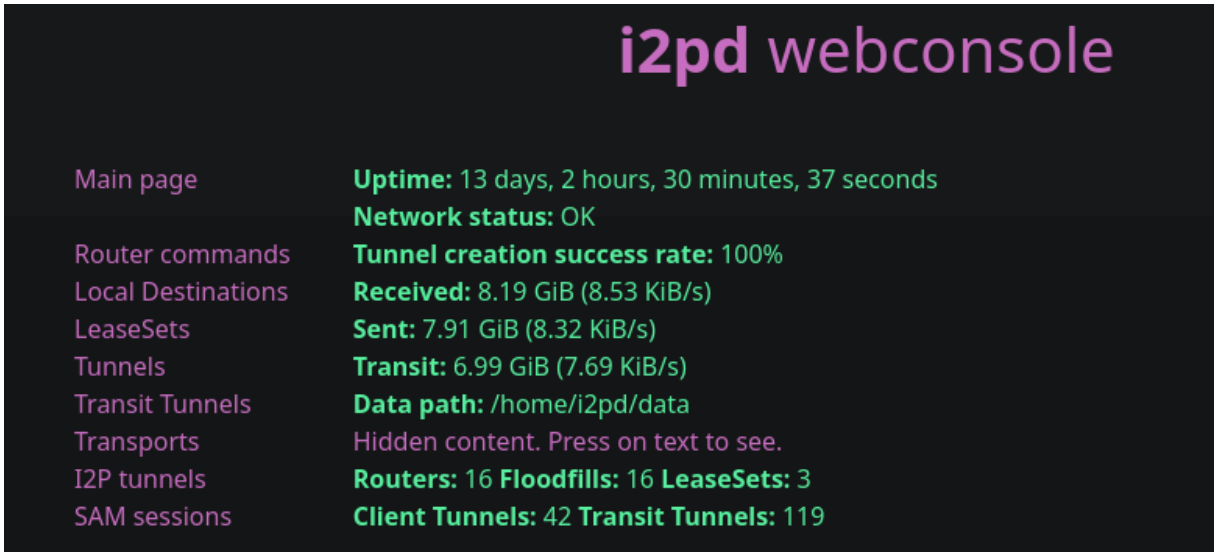

Abbildung 11: i2pd Router-Konsole (Testnetzwerk)

Über die Router-Konsole lassen sich unter anderem Informationen zu "`Network status`", "`Routers`" und "`Tunnel creation success rate`" erlangen.

Der `Network status` ist ein Wert welcher von i2pd sich aus mehreren Faktoren zusammensetzt. Dazu zählen z.B. Verbindungstests mit anderen I2P-Routern und Erkennung ob der I2P-Router hinter einem NAT bzw. Firewall ist. Der Status `OK` bedeutet, dass keine Probleme festgestellt wurden (PurpleI2P, o. J.).

Die Anzahl `Routers` gibt Rückschluss darauf wie viele aktive I2P-Router in der NetDB vorhanden sind. Falls Router nicht erreichbar sind werden diese automatisch entfernt. Der Wert `16` bedeutet, dass der Router alle Knoten im Netzwerk kennt und grundsätzlich erreichen kann. (PurpleI2P, o. J.).

Die `Tunnel creation success rate`, wie der Name schon sagt, meldet die Erfolgsquote bei der Erstellung von Tunnel. Eine Erfolgsquote von `100%` bedeutet, dass in letzter Zeit keine Erstellungen fehlgeschlagen sind, welche auf Probleme im Netzwerk hindeuten können. Im realen Netz über das Internet wird selten eine Quote von `100%` erreicht.

Neben den Informationen, die von den verschiedenen I2P-Routern zurückgegeben werden, wurden natürlich noch End-to-End-Tests durchgeführt. Unter anderem wurden in der Realisierung diverse

Verbindungen zwischen verschiedenen Routern erstellt. Dabei wurden verschiedene I2P-Router als HTTP-Proxy verwendet und verschiedene I2P-Adressen aufgerufen. Dabei konnten keine Probleme festgestellt werden.

Schlussendlich konnten mit der Analyse von Traffic in der Angriffs-Simulation auch kein falsches Verhalten festgestellt werden.

6.2 Angriffsideen

Die praktische Durchführung und Validation der Angriffsideen war Teil der Realisierung und wird dort genauer beschrieben. Zusammengefasst konnten die Ideen **Tunnelkontrolle** und **Mustererkennung Datenverkehr** für eine reale Umgebung nicht validiert oder wiederlegt werden. Zwar wurden in der Durchführung der **Mustererkennung Datenverkehr** Knoten auf einen Tunnel zugeordnet, dies war aber nur möglich, da es sich um ein kleines Netzwerk mit wenig “Lärm” handelte.

7 Reflexion

Generell ist das Projekt auf die Sicht der Implementation eines I2P-Testnetzwerkes gut verlaufen. Ein funktionsfähiges I2P-Testnetzwerk hat aber sehr viel mehr Zeit in anspruch genommen als initial Erwartet. Dadurch standen sehr viel weniger Zeit für die eigentlichen Angriffe zur Verfügung.

Das I2P-Projekt und sein dazugehöriges Ökosystem haben von Anfang an fasziniert. Die Idee, ein anonymes Overlay-Netzwerk zu schaffen, das es ermöglicht, sicher und privat im Internet zu kommunizieren, ist äusserst spannend. Die Vielseitigkeit der Anwendungsfälle, von sicheren Kommunikationskanälen bis hin zu anonymem Surfen, macht das I2P-Projekt zu einem bedeutenden Bestandteil der Privatsphäre im digitalen Zeitalter. Es ist bemerkenswert, dass trotz der begrenzten Grösse der Community, eine Fülle von Informationen verfügbar ist. Die Wissensdatenbank “geti2p.net” erwies sich als äusserst hilfreich und umfangreich. Sie bot einen perfekten Einstieg in das komplexe Thema von I2P. Durch die strukturierte Aufbereitung von Informationen auf der Webseite konnte ich mich schnell orientieren und einen Überblick über die verschiedenen Aspekte des Projekts gewinnen.

Es wäre sehr interessant die Verkehrsanalyse im I2P-Testnetzwerk weiterzuverfolgen und auf “realere” Begebenheiten anzupassen. Darüber hinaus könnte das Helm-Chart für das I2P-Testnetzwerk noch aufgeräumt und erweitert werden. Durch eine verbesserte Strukturierung und Erweiterung könnten verschiedene Konfigurationen noch einfacher verwendet werden, was die Test- und Entwicklungsprozesse effizienter gestalten würde. Dies könnte dazu beitragen, die Nutzung des I2P-Testnetzwerks zu erleichtern und die Entwicklung von Anwendungen im I2P-Ökosystem zu fördern.

Anhang

A. Aufgabenstellung Complesis

Typ der Arbeit

- Bachelorarbeit (BAA)

Titel

Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte („DIVA.EXCHANGE“)

Ausgangslage und Problemstellung

Das freie Software- und Netzwerkprojekt DIVA.EXCHANGE (https://diva.exchange) entwickelt den Softwareprototypen DIVA.

Technisch besteht diese freie und quelloffene Software aus einer Anonymisierungsschicht („I2P“), einer auf einer Blockchain basierenden Datenhaltung („Divachain“) und der darauf aufbauenden Handels- und Verwaltungssoftware („DIVA Frontend“).

DIVA hat den Zweck aufzuzeigen, wie die Aufbewahrung, der Handel und der Zahlungsverkehr mit digitalen Werten ganz ohne zentrale Dienstleister funktioniert – sicher und mit kompromisslosem Schutz der Privatsphäre.

Es handelt sich um ein langfristiges Forschungsprojekt.

Aus welchen Komponenten die Gesamtlösung besteht, kann aktuell wie folgt dargestellt werden: siehe Themenbild

Problemstellung:

Die Software „DIVA“ basiert auf dem Anonymisierungsnetzwerk „I2P“. Es soll aufgezeigt werden, wie die Teilnehmer des Netzwerkes de-anonymisiert werden können. De-Anonymisierung ist definiert als „Offenlegung der IP Adresse“ eines am DIVA.EXCHANGE Netzwerk teilnehmenden Knotens. Ziel der Arbeit und erwartete Resultate Das übergeordnete Ziel der Arbeit ist der Aufbau eines I2P Testnetzwerkes und die Durchführung von De-Anonymisierungs-Attacken innerhalb dieses Testnetzwerkes. aufzuzeigen, mit welchen Massnahmen und unter welchen Bedingungen Teilnehmer im I2P Netzwerk, spezifischer im DIVA.EXCHANGE I2P Netzwerk, de-anonymisiert werden können. Dazu soll auf bestehenden theoretischen Arbeiten der HSLU und DIVA.EXCHANGE aufgebaut werden.

Das Arbeitsresultat ist Source Code und ein Dokument:

- Source Code: automatisierter Aufbau Testnetzwerk und die automatisierte der Attacken zwecks Gewinnung von Daten zur Auswertung
- Dokument: Bachelor Arbeit

Gewünschte Methoden, Vorgehen

Das Projekt hat einen wissenschaftlichen Charakter basierend auf einer konkret vorliegenden Problemstellung. Das Vorgehen kann so gegliedert werden:

- Verständnis und Analyse vom I2P Netzwerk im Zusammenhang mit dem Projekt DIVA.EXCHANGE
- Spezifikation von Kriterien und Zielen
- Festlegung von geeigneten Methoden
- Programmierung und Durchführung der festgelegten Methoden
- Auswertung und Beschreibung der Resultate

Kreativität, Varianten, Innovation

Im Rahmen des Projektes sind alle Ideen und Varianten willkommen. Da das Projekt ein wissenschaftliches Forschungsprojekt ist, hat es nichts mit dem Tagesgeschäft zu tun und bietet viel Freiraum für Kreativität.

Gemäss Wissensstand der Auftraggeber existiert per April 2023 kein „Vollständig verteiltes und nichtdiskriminierendes Handelssystem für digitale Werte“ mit einer hinreichend Privatsphäre-schützenden Architektur. DIVA.EXCHANGE ist Innovation in Reinform.

Sonstige Bemerkungen (Anforderungen, Vorkenntnisse,. . .)

Das gesamte DIVA Projekt ist öffentlich. Der Quellcode ist hier: https://github.com/diva-exchange

Die freundlichen Entwickler freuen sich über Deine Kontaktaufnahme: https://t.me/diva_exchange_chat_de

Ein aktiver Austausch während der gesamten Arbeit ist vorteilhaft für alle Beteiligten und ist gewünscht.

Schlagwörter: Verteiltes System, dezentrales Netzwerk, Blockchain, digitale Werte, Handelssystem, I2P, Privatsphäre, Anonymität

Organisation Verein DIVA.EXCHANGE

Ansprechperson Carolyn Bächler-Schenk und Konrad Bächler

Funktion Gründungsmitglieder und Vorstand

Strasse Schochenmühlestrasse 4

PLZ/Ort 6340 Baar

Telefon 079 423 25 48

Email carolyn@diva.exchange, konrad@diva.exchange

Website https://diva.exchange

Themenerfasser:in Arnold Dieter dieter.arnold@hslu.ch

Betreuer:in Arnold Dieter dieter.arnold@hslu.ch

Projektarten:

- Innovationsprojekte (Projekte mit Erkenntnisgewinn, Forschungsprojekte)

Schwerpunkte:

- ICT Infrastrukturen
- Security/Privacy
- Software-Erstellung

Weitere Schwerpunkte:

- Blockchain
- I2P Netzwerk

B. Konfigurationen des I2P-Testnetzwerks

Die Skripte und das Helm-Chart für den Aufbau des I2P-Testnetzwerkes wurden in einem öffentlichen Github-Repository bereitgestellt.

- i2pd-testnet-kubernetes (Commit Hash: `96130a1f25e4787e9953fd542868b5fe9156f031`)

```
/i2pd-testnet-kubernetes/
|-- calico
|   `-- test-pool.yaml
|-- helm
|   `-- i2pd-chart
|       |-- charts
|       |-- Chart.yaml
|       |-- setup.sh
|       |-- templates
|       |   |-- configmap-i2pdconf.yaml
|       |   |-- configmap-reseedzip.yaml
|       |   |-- configmap-tunnels.yaml
|       |   |-- deployment.yaml
|       |   |-- _helpers.tpl
|       |   |-- ingress.yaml
|       |   |-- nginx.yaml
|       |   |-- NOTES.txt
|       |   |-- pvc.yaml
|       |   |-- serviceaccount.yaml
|       |   |-- service-console.yaml
|       |   |-- service-httpproxy.yaml
|       |   |-- service-i2pcontrol.yaml
|       |   |-- service-sam.yaml
|       |   |-- service-socksproxy.yaml
|       |   `-- tests
|       |       `-- test-connection.yaml
|       `-- values.yaml
|-- LICENSE
|-- README.md
|-- scripts
|   `-- traffic.py
|-- t1_flow_short.png
`-- traffic.png

8 directories, 25 files
```

Glossar

Container Ein Container ist eine Art Softwarepaket, das den Code und alle seine Abhängigkeiten verpackt, damit die Anwendung schnell und zuverlässig in einer anderen Computerumgebung ausgeführt werden kann (*What is a Container? | Docker*, o. J.).

Clearnet bezieht sich in der Regel auf den regulären, nicht anonymen und öffentlich zugänglichen Teil des Internets.

Curl Curl ist ein Cli-Tool für die Übertragung von Daten über verschiedene Netzwerkprotokolle (z.B. HTTP).

Debian Debian ist ein weit verbreitetes und frei verfügbares Betriebssystem, das zu den sogenannten Linux-Distributionen gehört.

Docker Docker ist eine Plattform für die Entwicklung, den Transport und die Ausführung von Anwendungen in leichtgewichtigen, portablen Containern, die den Anwendungscode und die Abhängigkeiten kapseln (*What is a Container? | Docker*, o. J.).

Docker-Image / Container-Image Ein Container-Image ist ein leichtgewichtiges, eigenständiges und ausführbares Softwarepaket, das alles enthält, was zur Ausführung einer Software benötigt wird, einschliesslich Code, Laufzeit, Bibliotheken und Systemtools.

Fork In der Softwareentwicklung bezieht sich der Begriff “Forks” auf die Erstellung einer unabhängigen Kopie eines Projekts oder einer Codebasis. Ein Fork entsteht, wenn ein Entwickler oder eine Gruppe von Entwicklern beschliesst, eine Abspaltung (Fork) eines bestehenden Projekts vorzunehmen, um daran unabhängig weiterzuarbeiten.

Freenet Freenet ist ein dezentralisiertes und Zensur resistentes Peer-to-Peer-Netzwerk, das es den Nutzern ermöglicht, Informationen auszutauschen und über ein verteiltes Datenspeicher- und -abrufsystem anonym auf Inhalte zuzugreifen.

Geoblocking Geoblocking, kurz für geografisches Blockieren, bezeichnet die Praxis, den Zugang zu Inhalten oder Diensten auf der Grundlage des geografischen Standorts des Nutzers zu beschränken.

Git Git ist ein verteiltes Open-Source Versionskontrollsystem, das entwickelt wurde, um alles von kleinen bis zu sehr grossen Projekten schnell und effizient zu bearbeiten (*Git*, o. J.).

Github GitHub eine cloud-basierter Git-Repository Hosting-Service von Microsoft.

Helm Helm ist ein Paketmanager für Kubernetes-Anwendungen. Er vereinfacht die Bereitstellung und Verwaltung von Anwendungen auf Kubernetes, indem er eine Möglichkeit bietet, selbst die komplexesten Kubernetes-Anwendungen zu definieren, zu installieren und zu aktualisieren (*Helm*, o. J.).

Helm-Chart Ein Helm Chart ist eine Sammlung von vordefinierten Kubernetes-Ressourcen und Konfigurationen, die es ermöglichen, Anwendungen einfach und wiederholbar in Kubernetes-Clustern zu installieren und zu verwalten.

I2P-Client Der Begriff I2P-Client bezieht sich auf eine spezifische Implementation von einem I2P-Router

(z.B. i2pd oder i2p.i2p).

I2pcontrol I2PControl ist eine I2P-Router-Schnittstelle um sich mit einem laufenden I2P-Router zu verbinden.

I2P-Router / Knoten Ein I2P-Router ist ein Gerät oder Computer, auf dem die I2P-Software installiert ist und aktiv läuft.

IRC IRC steht für "Internet Relay Chat". Es handelt sich dabei um ein textbasiertes Kommunikationsprotokoll, das für die Echtzeitkommunikation über das Internet verwendet wird.

JSON JSON kurz für *JavaScript Object Notation* ist ein offenes Dateiformat welches initial als Teil von JavaScript entwickelt wurde.

JSON-RPC JSON-RPC (JavaScript Object Notation Remote Procedure Call) ist ein in JSON kodiertes Protokoll für *Remote Procedure Call* (RPC). Es ermöglicht die Kommunikation zwischen einem Client und einem Server über ein Netzwerk, in der Regel über HTTP oder andere Transportprotokolle.

K3s K3s ist eine zertifizierte Kubernetes-Distribution für IoT und Edge-Computing.

Kubernetes Kubernetes ist eine Open-Source-Platform für die Verwaltung und Orchestrierung von Container-Workloads, die sowohl die Automatisierung als auch eine deklarative Konfiguration ermöglicht (*Overview | Kubernetes*, o. J.).

Kubectl Kubectl ist ein Cli-Tool für die Interaktion mit Kubernetes-Clustern.

Layered Encryption Layered Encryption bezieht sich auf eine Sicherheitstechnik, bei der mehrere Verschlüsselungsschichten zum Schutz von Daten eingesetzt werden. Jede Schicht bietet eine zusätzliche Sicherheitsstufe, die es für Unbefugte schwieriger macht, auf die ursprünglichen Informationen zuzugreifen.

LeaseSet Ein LeaseSet speichert Informationen über ein bestimmtes Ziel (z.B. I2P-Website) (siehe Kapitel Network Database) (*I2P*, o. J.).

Link Layer Der Link Layer ist die erste Schicht im TCP/IP-Modell. Diese Schicht befasst sich mit der physischen Verbindung zum Netz und der lokalen Übermittlung von Rahmen zwischen Geräten im selben Netz. Sie umfasst Protokolle für Ethernet, Wi-Fi und andere. Die Adressierung auf dieser Schicht erfolgt in der Regel in Form von MAC-Adressen (Media Access Control).

Linux Namespace Ein Linux Namespace ist ein Kernel-Feature, das dazu dient, Ressourcen zu isolieren und um Prozesse in separaten Umgebungen auszuführen. Dies ermöglicht die Schaffung von unabhängigen Instanzen bestimmter Systemressourcen für verschiedene Prozesse oder Gruppen von Prozessen. Jeder Namespace bietet eine isolierte Sicht auf eine bestimmte Ressource, was bedeutet, dass Prozesse in einem Namespace nur diejenigen Ressourcen sehen und beeinflussen können, die diesem Namespace zugeordnet sind (*namespaces(7) - Linux manual page*, o. J.).

Nginx Nginx ist ein Open-Source-Applikation für die Bereitstellung von Webserver, Reverse-Proxy, Load-Balancer, etc.

NTCP NTCP kurz für *NIO-based TCP* ist ein Protokoll im *I2P Transport Layer* welches auf TCP aufbaut.

Noise Protocol Das Noise-Protokoll-Framework ist ein kryptografisches Protokoll-Framework, das

für die Erstellung sicherer und effizienter Kommunikationsprotokolle entwickelt wurde. Es wird meist mit der modernen VPN-Applikation *WireGuard* in Verbindung gebracht (*The Noise Protocol Framework*, o. J.).

Onion Routing Onion Routing st eine Technik zum Aufbau von Pfaden oder Tunneln durch eine Reihe von Knoten und zur anschliessenden Nutzung dieser Tunnel (siehe Tunnel Encryption).

Orchestrator Englisch für Koordinator. Wird verwendet für die Komponente, welche verantwortlich für die Orchestrierung ist. Unter Orchestrierung versteht man die automatische Ausführung von mehrerer verketteter Workflows und Prozesse, die über verschiedene Systeme und Services hinweg koordiniert werden (Rauch, 2022).

Peer-to-Peer Peer-to-Peer (P2P) bezieht sich auf eine dezentrale Netzwerkarchitektur, bei der die Teilnehmer, die sogenannten Peers, direkt miteinander interagieren und Ressourcen oder Informationen gemeinsam nutzen, ohne dass ein zentraler Server oder eine Behörde erforderlich ist.

Proxmox Proxmox oder genauer Proxmox Virtual Environment (Proxmox VE) ist eine Open-Source-Virtualisierungsplattform, die zwei Virtualisierungstechnologien kombiniert: KVM (Kernel-based Virtual Machine) für virtuelle Maschinen (VMs) und LXC (Linux Containers) für eine leichtgewichtige containerbasierte Virtualisierung. Proxmox VE ist als vollständige Virtualisierungslösung konzipiert, die eine webbasierte Verwaltungsoberfläche, Speicherunterstützung und Netzwerkfunktionen umfasst.

RouterInfo Eine RouterInfo ist sozusagen die Router-Identität und speichert Informationen über einen bestimmten I2P-Router und wie man ihn kontaktieren kann (*I2P*, o. J.).

SSU SSU kurz für *Secure Semireliable UDP* ist ein Protokoll im *I2P Transport Layer* welches auf UDP aufbaut.

Stdout In Unix(-ähnlichen) Betriebssystemen, wie z.B. GNU/Linux, steht “stdout” für Standardausgabe. Es ist einer der drei Standardströme, die ein Unix-Prozess für die Ein- und Ausgabe verwenden kann, neben Standardeingabe (stdin) und Standardfehler (stderr).

Tcpdump Tcpdump ist ein weit verbreitetes Open-Source-Tool für die Paketanalyse im Bereich der Netzwerkkommunikation. Es ermöglicht Benutzern, den Datenverkehr auf einem Netzwerksegment aufzuzeichnen und zu analysieren.

Tor Tor ist eine Open-Source-Software, die anonyme Kommunikation ermöglicht, indem sie den Internetverkehr durch ein freiwilliges Overlay-Netzwerk leitet, das den Standort und die Nutzung eines Benutzers vor Netzwerküberwachung oder Verkehrsanalyse verbirgt.

Wireshark Wireshark ist ein Open-Source-Tool, für das erfassen und Analysieren von Netzwerkverkehr.

Abkürzungsverzeichnis

API Application Programming Interface
HTTP Hypertext Transfer Protocol
I2P Invisible Internet Project
IP Internet Protocol
NTCP Nio-based TCP (I2P)
SSU Secure Semireliable UDP (I2P)

Abbildungsverzeichnis

Literaturverzeichnis

A Gentle Introduction to How I2P Works - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/en/docs/how/intro

Alternative I2P clients - I2P. (o. J.). Abgerufen 1. Januar 2024, von https://geti2p.net/en/about/alternative-clients

Boss, Brian, & Purtschert, Marco. (2022). *Deanonymisierung von Teilnehmern des I2P-basierten Handelsnetzwerk für digitale Werte („DIVA.EXCHANGE")*. HSLU.

Common structures Specification - I2P. (o. J.). Abgerufen 1. Januar 2024, von https://geti2p.net/spec/common-structures#routerinfo

Configuration File Specification - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/spec/configuration

Configuring - i2pd documentation. (o. J.). Abgerufen 29. Dezember 2023, von https://i2pd.readthedocs.io/en/latest/user-guide/configuration/

DevX Glossray: I2P. (o. J.). In *DevX*. Abgerufen 27. Dezember 2023, von https://www.devx.com/terms/i2p/

DIVA.EXCHANGE. (o. J.). Abgerufen 21. September 2023, von https://www.diva.exchange/de/#body_top

Douceur, John R. (2002). The sybil attack. *International workshop on peer-to-peer systems*, 251–260.

Egger, Christoph, Schlumberger, Johannes, Kruegel, Christopher, & Vigna, Giovanni. (2013). Practical attacks against the I2P network. *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings 16*, 432–451.

Freie Banking-Technologie für alle : DIVA.EXCHANGE. (2020). https://www.diva.exchange/de/verein-diva-exchange/freie-banking-technologie-fuer-alle/

Garlic Routing - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/en/docs/how/garlic-routing

Git. (o. J.). Abgerufen 30. Dezember 2023, von https://git-scm.com/

Helm. (o. J.). Abgerufen 1. Januar 2024, von https://helm.sh/

Herrmann, Michael, & Grothoff, Christian. (2011). Privacy-implications of performance-based peer selection by onion-routers: a real-world case study using I2P. *Privacy Enhancing Technologies: 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings 11*, 155–174.

I2CP - I2P. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/docs/protocol/i2cp

I2P Anonymous Network. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/

I2P Network Protocol (I2NP) - I2P. (o. J.). Abgerufen 28. Dezember 2023, von https://geti2p.net/en/docs/protocol/i2np

I2P's Threat Model - I2P. (o. J.). Abgerufen 1. Januar 2024, von https://geti2p.net/en/docs/how/threat-model

I2P: A scalable framework for anonymous communication - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/en/docs/how/tech-intro

i2p/i2p.i2p: I2P is an anonymizing network, offering a simple layer that identity-sensitive applications can use to securely communicate. All data is wrapped with several layers of encryption, and the network is both distributed and dynamic, with no trusted parties. (o. J.). Abgerufen 31. Dezember 2023, von https://github.com/i2p/i2p.i2p/tree/master

LNS. (o. J.-a). *l-n-s/docker-testnet: Run and manage your own local I2P network with docker*. Abgerufen 31. Dezember 2023, von https://github.com/l-n-s/docker-testnet

LNS. (o. J.-b). *l-n-s/testnet.py: I2P test network management tool for Linux. Runs both I2P and i2pd with minimum bloat.* Abgerufen 30. Dezember 2023, von https://github.com/l-n-s/testnet.py

Low-level Cryptography Specification - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/spec/cryptography

Mikal, Villa. (o. J.-a). *How to run 128 (testnet) I2P routers in multiple subnets on a single Linux system. - 0xcc.re*. Abgerufen 31. Dezember 2023, von https://0xcc.re/2018/10/16/howto-run-128-i2p-routers-in-multiple-subnets-on-a-single-linux-system.html

Mikal, Villa. (o. J.-b). *mikalv/i2p-testnet*. Abgerufen 31. Dezember 2023, von https://github.com/mikalv/i2p-testnet/tree/master

namespaces(7) - Linux manual page. (o. J.). Abgerufen 31. Dezember 2023, von https://www.man7.org/linux/man-pages/man7/namespaces.7.html

Networking overview | Docker Docs. (o. J.). Abgerufen 31. Dezember 2023, von https://docs.docker.com/network/

NTCP 2 - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/spec/ntcp2

NTCP (NIO-based TCP) - I2P. (o. J.). Abgerufen 28. Dezember 2023, von https://geti2p.net/en/docs/transport/ntcp

Overview | Kubernetes. (o. J.). Abgerufen 1. Januar 2024, von https://kubernetes.io/docs/concepts/overview/

Peer Profiling and Selection - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/en/docs/how/peer-selection

Protocol Stack - I2P. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/docs/protocol

PurpleI2P. (o. J.). *PurpleI2P/i2pd: I2P: End-to-End encrypted and anonymous Internet*. Abgerufen 2. Januar 2024, von https://github.com/PurpleI2P/i2pd

Rauch, Gedeon. (2022). *Was bedeutet Orchestrierung?* https://www.dev-insider.de/was-bedeutet-orchestrierung-a-678fccf49678d4063bcb1727cc00cae6/

sadie. (o. J.). *20 Years of Privacy: A Brief History of I2P - Blog - I2P*. Abgerufen 1. Januar 2024, von https://geti2p.net/en/blog/post/2021/08/28/History-of-I2P

Secure Semireliable UDP (SSU) - I2P. (o. J.). Abgerufen 28. Dezember 2023, von https://geti2p.net/en/docs/transport/ssu

SSU2 - I2P. (o. J.). Abgerufen 28. Dezember 2023, von https://geti2p.net/spec/ssu2

TCP/IP Model - GeeksforGeeks. (o. J.). Abgerufen 28. Dezember 2023, von https://www.geeksforgeeks.org/tcp-ip-model/

The Network Database - I2P. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/docs/how/network-database

The Noise Protocol Framework. (o. J.). Abgerufen 28. Dezember 2023, von https://noiseprotocol.org/noise.html#overview

Tor Documentation for Whonix Users. (2023). In *Whonix*. https://www.whonix.org/wiki/Tor

Tunnel Implementation - I2P. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/docs/tunnels/implementation

Tunnel Message Specification - I2P. (o. J.). Abgerufen 28. Dezember 2023, von https://geti2p.net/spec/tunnel-message

Tunnel Routing - I2P. (o. J.). Abgerufen 29. Dezember 2023, von https://geti2p.net/en/docs/how/tunnel-routing

Wang, Qiyan, Mittal, Prateek, & Borisov, Nikita. (2010). In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. *Proceedings of the 17th ACM conference on Computer and communications security*, 308–318.

Web Browser Configuration - I2P. (o. J.). Abgerufen 21. September 2023, von https://geti2p.net/en/about/browser-config

What is a Container? | Docker. (o. J.). Abgerufen 1. Januar 2024, von https://www.docker.com/resources/what-container/

Wilde, Thomas, & Hess, Thomas. (2007). Forschungsmethoden der Wirtschaftsinformatik. *Wirtschaftsinformatik*, *4*(49), 280–287.

Wilson, Maxim, & Bazli, Behnam. (2016). Forensic analysis of I2P activities. *2016 22nd International Conference on Automation and Computing (ICAC)*, 529–534.