Softwareprojekt 2016

Reviewdokument: PECTO

Rapid Layer-2 Encryption Framework

Yannic Faulwetter
Michael Fuchs
Milan Haverkock
Felix Seidel
Daniel Scheliga
Tobias Schubert
Jörn Weisensee
Nils Winkelbach

letzte Änderung: 22. April 2017

Inhaltsverzeichnis

Reviewdokument der Validierungsphase

In diesem Reviewdokument sollen die Ergebnisse des Softwareprojektes in der dritten Phase betrachtet werden. Insbesondere wird auf wichtige Testfälle eingegangen, die beim Testen der Software entwickelt wurden. Als Anhang liegt die komplette von Doxygen generierte Entwicklerdokumentation und ein Benutzerhandbuch, welches die Installationsanleitung beinhaltet, bei.

1 Ziele und Relevanz des Frameworks PECTO

PECTO (Paket EnCryption on layer TwO) ist ein Framework, dass es ermöglicht, verschiedene zu schützende, rote Netze sicher zu verbinden. Da hier verschiedenartig sturkturierte Netzwerke mit unterschiedlichen Kommunikationsprotokollen arbeiten, agiert es als ein Gateway, welches die Kommunikation zwischen roten Netzen über ein unsicheres, schwarzes Netz ermöglicht. Dabei werden die Pakete, welche zwischen roten Netzen versandt werden, verschlüsselt über ein schwarzes Netz übertragen. Im Gegensatz zu schon verbreiteten VPN-Lösungen erfolgt die Verschlüsselung jedoch auf Layer-2 des ISO/OSI-Schichtenmodells, behandelt also Ethernet-Frames statt IP-Pakete. PECTO kann zur Absicherung von lokalen Netzwerken verwendet werden, ohne Spezialhardware nutzen zu müssen.

2 Ziele im beim Entwurf PECTOs

An das zu entwickelnde Framework PECTO stellten wir verschiedene Anforderungen, insbesondere auch an die Performance und Sicherheit. Diese finden sich auch an unterschiedlichen Stellen in den Entwurfsentscheidungen wieder. Dabei konkurriert der Sicherheitsgedanke, welcher nach möglichst einfachem Code strebt, stets mit dem Wunsch nach hoher Performance. Um einen guten Ausgleich zu finden, werden alle Komponenten zunächst auf gute Verständlichkeit des Codes optimiert. Sollten sich beim Profiling Probleme zeigen, werden diese Stellen gezielt überarbeitet.

Verwendete Muster

Zur Realisierung von PECTO werden folgende Entwurfsmuster verwendet:

Inversion of Control (IoC) Um das Schichtenmodell umsetzen zu können und zyklische Abhängigkeiten zu vermeiden, wird Inversion of Control eingesetzt. Die tiefere Schicht stellt dafür jeweils Interfaces bereit, die von den höheren Schichten implementiert werden. Der sich dadurch ergebende Overhead ist in der Regel zu vernachlässigen. Insbesondere überwiegen die Vorteile durch eine bessere Entkoppelung. Wenn sich Performance-Probleme ergeben, kann davon abgewichen werden.

Strategy Pattern An verschiedenen Stellen im Entwurf können verschiedene Algorithmen verwendet werden. Damit dies konfgurierbar ist, wird das Strategy Pattern verwendet. Ein Beispiel hierfür ist die Wahl des Verschlüsselungsalgorithmus, der zu Debuggingzwecken und für Erweiterbarkeit verschiedenartig gewählt werden kann. Zur Auswahl stehen zum Beispiel AES-GCM, eine unsichere XOR-Verschlüsselung oder gar keine Verschlüsselung.

Exceptions Zur Vereinfachung des Codes werden Exceptions eingesetzt. Diese sind immer dann auszugeben, wenn ein unerwartetes Ereignis eingetreten ist. Da der Einsatz von Exceptions einen erheblichen Overhead zur Folge haben kann, wenn zu viele von ihnen geworfen werden, müssen für besonders performance-kritische Funktionen immer zwei Schnittstellen zur Verfügung stehen. Die erste dieser Schnittstellen löst die Exceptions aus, während die zweite über Rückgabewerte entscheidet ob die Funktion erfolgreich ausgeführt wurde.

Code Contracts Jede Methode, die zu einer öffentlichen Schnittstelle gehört, validiert ihre Parameter und Ausgaben. Unerwartete Eingaben, beispielsweise Nullzeiger, müssen erkannt werden und zu einer Exception führen. Ebenfalls sollen alle Ausgaben einer einfachen Plausibilitätsprüfung unterzogen werden. Auch hier liegt besonderes Augenmerk auf Nullzeigern. So wird sichergestellt, dass keine unerwarteten Rückgabewerte zu Fehlern an anderer Stelle führen können. Ausnahmen bilden einige performancekritische Funktionen, die in auch eine Variante bereitstellen, die keine Prüfung durchführt.

3 Testergebnisse

Um die Funktionalität und Korrektheit unserer Software zu überprüfen, wurden einige Unit Tests und manuelle Tests durchgeführt.

3.1 Überprüfung des korrekten Verhaltens wichtiger Komponenten

Die Unit Tests aller getesteten Kompontenen verlaufen erfolgreich. Für die verschiednenen Module wurden folgende Eigenschaften geprüft:

PESpinlock Der gegenseitige Ausschluss wird getestet, indem zwei Threads gleichzeitig die gleiche Variable hochzählen.

PEMemoryBuffer Das Anfügen und Löschen von Daten am Anfang und Ende wird getestet. Zudem wird überprüft, ob Puffer nach ihrer Verwendung wieder korrekt freigegeben werden.

PEForwardingTable Die korrekte Übersetzung von IP-Addressen in MAC-Addressen der zuständigen Instanz wird überprüft.

PECryptoAlgorithm Die verschiedenen Verschlüsselungsmethoden werden auf ihre Performance und Funktionalität getestet.

Weitere Komponenten wurden nicht durch Unit Tests überprüft, weil dazu umfangreiche Fakes und Mocks nötig wären.

3.2 Nachweis funktionaler und nichtfunktionaler Eigenschaften durch manuelle Tests

Verschiedene manuelle Tests wurden genutzt, um die funktionalen und nichtfunktionalen Eigenschaften nachzuweisen. Die Tests fanden in einer realen und einer virtualisierten Umgebung statt.
Grundsätzlich wurden zwei Arten von Tests vorgenommen: Lasttests und Funktionstests. Die Lasttests fanden stets in der nichtvirtualisierten Umgebung statt, um aussagekräfigte Ergebnisse zu liefern. Funktionstests wurden in beiden Testumgebungen ausgeführt.
Folgende Funktionstests wurden ausgeführt:

Ping-Test Beim Ping-Test wurden Pakete von verschiedenen roten Teilnetzen aus in jeweils andere Teilnetze versendet. Dadurch konnte die Funktionalität von ARP- und IP-Weiterleitung getestet werden. Das Gesamtnetz hat dabei die Netzadresse 10.0.0.0/16, während die einzelnen roten Teilnetze die Form 10.0.x.0/24 besaßen. In jedem Teilnetz gab es genau einen Teilnehmer, welcher die IP-Adresse 10.0.x.1 verwendete.

Die Anzahl der Teilnetze betrug in der realen Testumgebung zwei ($x \in \{1, 2\}$) und in der virtualisierten Umgebung drei ($x \in \{1, 2, 3\}$).

TCP-Übertragungstest Mithilfe des Linux-Tools `iperf` konnte die Funktionalität von TCP-Verbindungen und die Übertragung von großen Paketen getestet werden. Dabei konnten auch Probleme entdeckt werden, welche Folge einer zu klein gewählten MTU waren.

Die Netze wurden wie beim Ping-Test gewählt.

Neben den Funktionstests wurden in der realen Umgebeung auch Performancetests ausgeführt. Hierbei gab es zwei Arten von Tests. Einerseits wurde ein Netz, wie es schon beim Ping-Test beschrieben ist, verwendet, andererseits wurden noch einfachere Netze genutzt, um die Performance der Kryptokompontenten im Gesamtsysten zu testen. Das reine Performancetestnetzwerk besteht aus zwei Clients mit den IP's 1.1.1.1 und 2.2.2.2. Beide senden und empfangen mithilfe eines PECTO-basierenden Tools kleine IP-Pakete. So kann die Performance des Systems bei der Verarbeitung von 64 Byte großen Paketen untersucht werden. Die einzelnen Tests lieferten Folgende Ergebnisse:

Lasttest mit kleinen Paketen Das Generator-Tool produziert 1,43 Mio. Pakete pro Sekunde, wovon wieder 1,01 Mio. Pakete auf der Gegenseite empfangen werden. Die entstehenden rund 30% Verlust kommen dadurch zustande, dass der Generator die Gigabit-Leitung voll auslastet, die verschlüsselten Pakete jedoch größer sind und damit die vorhandene Bandbreite überstiegen wird.

Latenzbestimmung Der Aufbau für die Latenzbestimmung entspricht dem Ping-Test im realen Netz. Es werden etwa 1000 Ping-Pakete pro Sekunde versendet und die Round-Trip-Time gemessen. Durch eine Verlgeichsmessung ohne zwischengeschaltene PECTO-Instanzen kann die Latenz der Ver- und Entschlüsselung abgeschätzt werden.

Die Auswertung dieses Tests ergab, dass die Latenz für eine Verschlüsselung und anschließende Entschlüsselung kleiner Pakete etwa 65-70µs beträgt. Diese Verzögerungen können jedoch nur eingehalten werden, wenn das System nicht ausgelastet ist.

Durchsatzbestimmung für TCP-Verbindungen Die Durchsatzbestimmung erfolgte mit demselben Versuchsaufbau wie der TCP-Übertragungstest in der realen Umgebung. Es ergab sich ein Durchsatz von 913 MBit, wobei der limitierende Faktor wiederum die fehlende Bandbreite im schwarzen Netz war.

4 Bugreview

Im folgenden werden die schwerwiegendsten Probleme beschrieben, welche während der Erstellung dieser Software aufgetreten sind. Außerdem wird dargestellt wie diese gelöst werden konnten.

4.1 Receive Side Scaling (RSS) im schwarzen Netz

In diesem Abschnitt wird das Receive Side Scaling beschrieben. Dabei handelt es sich um die verwendete Technologie, um Pakete zwischen verschiedenen Warteschlangen zu verteilen.

Receive Side Scaling Receive Side Scaling ist eine Technologie, bei der die Netzwerkkarte empfangene Pakete auf verschiedene Warteschlangen verteilt. Diese wird von PECTO benutzt, um das Load Balancing zwischen verschiedenen Threads umzusetzen.

Problembeschreibung RSS nutzt verschiedene Informationen aus höheren Schichten, z.B. IP-Adressen, um die Pakete sinnvoll an verschiedende Warteschlangen zu verteilen. Wenn jedoch ein PECTO-Paket aus dem schwarzen Netz empfangen wird, kann auf diese Informationen zwangsläufig noch nicht zugegriffen werden. Damit kann auch kein Balancing stattfinden.

Workaround Einige teurere Netzwerkkarten können RSS auch anhand des gesamten Paketes ausführen, jedoch ist dies keine Lösungsmöglichkeit, welche für die Realisierung dieses Systems in Frage kommt. Ein anderer Workaround ist das Setzen von benutzerdefinierten Filtern, sogenannten Flex Filters, auf dem Netzwerkinterface, die verschiedene Pakete explizit den verschiedenen Queues zuordnen.

Problemlösung Es wurde ein Feld zum manuellen Round-Robin-Loadbalancing in die PECTO-Header eingefügt. Mithilfe meherer Flex-Filter werden die Pakete dann automatisiert von der Netzwerkkarte auf mehrere Threads verteilt. Somit wird kein RSS mehr verwendet, sondern eine gleichweritge Alternative.

5 Bewertung des Projektes

In Hinblick auf die Bewertung des Projektes wird im folgenden insbesondere Erfolg und Vorgehen kritisch betrachtet.

5.1 Kritische Bewertung des Projekterfolgs

Im Verlauf dieses Softwareprojektes konnten die einzelnen Schritte, der Entstehung einer Softwarekomponente praktisch abgearbeitet und miterlebt werden. Insbesondere stellte die Erarbeitung der verschiedenen Dokumente eine Herausforderung dar. Nachdem die Einarbeitung und eine Arbeitseinteilung für die einzelnen Dokumentationsbereiche erfolgt waren, wurde sich gut in den Prozess der Erstellung solcher Dokumente eingefunden.
Die einzelnen Ziele, welche vom Team erstellt und angestrebt wurden, konnten immer gut und termingetreu abgearbeitet werden. Sämtliche Anforderungen welche nach reichlichen Überlegungen innerhalb des Pflichtenheftes vereinbart wurden, konnten mit Abschluss des Projektes umgesetzt werden. Insbesondere wurden auch alle Wunschkriterien erfolgreich implementiert. Dies gelang nicht jedoch ohne weitere Schwierigkeiten. Die Einarbeitung in das zugrunde liegende DPDK-Framework erwies sich unerwartet deutlich komplexer, als zu Beginn der Projektplanung angenommen wurde. Da nur bestimmte Komponenten aus dem Framework benötigt wurden mussten diese zunächst identifiziert werden und erst anschließend konnte mit der Einarbeitung in die jeweiligen Teilgebiete begonnen werden.
Wegen der hohen Funktionalität, welche das DPDK mit sich bringt und dessen teilweise sehr schwer verständlichen Dokumentation, konnte die Einarbeitung wegen Zeitmangels nur parallel zur eigentlichen Entwicklung der eigenen Software geschehen. Durch derartige Verzögerungen musste die Planung der Entwicklung so strukturiert werden, dass die Cryptokomponente lediglich zum Ende der zweiten Iteration implementiert und getestet werden konnte.
Des Weiteren stellten die Strukturierung der Schnittstellen und das Testen der Skalierbarkeit neue Herausforderungen dar. Hierbei spielte die Entscheidung über die Verwendung der Datentypen zur Speicherung von Tabelleneinträgen und anderen Objekten eine tragende Rolle. Zum Beispiel sollten anfangs Einträge für das Forwarding mit einer Map in jedem Thread festgehalten werden. Dies wurde aus flexibilitätsgründen wieder verworfen, da eine Tabelle für alle Threads zur Verfügung stehen muss. Die Umgestaltung der Wahl von Datenstrukturen und ihrer Realisierung wurde somit im Laufe des Projektes mehrfach abgewandelt um die Optimierung bzw. Einfachheit der Implementierung zukünftiger Elemente zu gewährleisten.
Größere Bugs welche den Prozess der Arbeit eingedämmt haben, manifestierten sich aufgrund eines guten Gesamtüberblicks über die Software nur in Form des Receive-Side-Scalings. Nach intensiven Überlegungen und Kommunikation innerhalb des Teams, als auch mit dem Betreuer, konnte schließlich ein gleichwertiger Lösungsansatz entwickelt und umgesetzt werden.
Das Testen der Software wurde zunächst für einzelnen Komponenten durchgeführt. Im weiteren Verlauf des Projektes konnten die verschiedenen Teile jedoch immer mehr zusammengefügt werden und die entstandenen komplexeren Teilsysteme konnten getestet werden. Mit frühen Tests an den einzelnen Komponenten, konnte im Fehlerfall schnell und präzise reagiert und entstandene Probleme bereits im Ursprung ausgemerzt werden. Bei Tests mit mehreren Komponenten konnte das Zusammenspiel des gesamten Teilsystems getestet und hierbei auf

Fehler in den Schnittstellen und der Kommunikation geachtet werden.
Im gesamten Verlauf des Projektes konnten durch die Arbeit im Team verschiedene und komplexe Probleme gelöst werden, welche viele neue Perspektiven aufgezeigt haben. Jedes der Teammitglieder konnte sich diverse neue Fähigkeiten angeeignen, die voraussichtlich auch im weiteren Verlauf des Studiums, als auch der zukünftigen Arbeitspraxis, von Bedeutung sind. Mit dem Ende der Projektzeit konnte schließlich der gesamte Entwicklungsprozess einer Software erlebt werden, welche PECTO als Endprodukt mit sich bringt.

5.2 Kritische Bewertung des Vorgehens

Im Anfangsstadium des Softwareprojektes, stellte sich zunächst die Frage welches Vorgehen das Team für die Entwicklung der Softwarekomponente anstreben sollte. Dabei stellte sich heraus, dass das agile Vorgehensmodell aufgrund seiner Struktur am besten geeignet war um eine Cryptosystem zu implementieren, da stetige Anpassungen und Weiterentwicklungen eine hohe Priorität in der Auswahl unseres Vorgehens bildeten. Um zunächst eine Teamstruktur aufzustellen, wurden Verantwortlichkeitsbereiche festgelegt und aufgeteilt. Dies erleichterte ungemein die folgenden Arbeitsaufteilungen.
Als einzige Hürden erwiesen sich die unterschiedlichen Programmierniveaus und die verschiedenen Vorlesungszeiten der einzelnen Teammitglieder, wodurch die Planung der Treffen der Programmiergruppen beeinflusst wurde. Diese Programmierteams bestanden vorwiegend aus Zweier- bzw. Dreierteams um möglichst effektiv Pair-Programming zu betreiben. Bei der Teambildung wurde im Laufe der Projektes verstärkt versucht darauf zu achten, dass die verschiedenen Stärken und Schwächen der individuellen Mitglieder durch den jeweils anderen kompensiert werden konnten. Trotzdem konnten einzelne Teams schneller mit der eigentlichen Implementierung beginnen, während andere mehr Zeit für die Einarbeitung in die Programmiersprache, das Framework oder andere Verständnisprobleme benötigten. Daher war insbesondere gegen Ende des Projektes die Unterstützung der Kleingruppen untereinander unabdingbar, um die vereinbarten Meilensteine der jewailigen Iteration zu komplettieren.
Aufgrund der individuellen Treffen innerhalb der Gruppe, fand die Kommunikation größtenteils während der Gruppenmeetings und persönlich statt. Die digitalen Kommunikationsmöglichkeiten auf die sich innerhalb der Gruppe geeinigt wurde, wurden dann primär zur Aktualisierung des derzeitigen Entwicklungsstandes und dem Austausch der fertig gestellten Dokumente genutzt.
Währed der ersten Iteration wurde mit dem Entwurf der Software und der Implementierung einiger grundlegenden Kernfunktionalitäten begonnen. So musste anschließend nur noch eine Verfeinerung der vorhandenen Grobstruktur stattfinden und das Team konnte sich mehr auf die Erweiterung weiterer essentieller Programmteile konzentrieren. Das System wurde also Stück für Stück iterativ mit zusätzlicher Funktionalität ausgestatet. Vorteil hierbei war unter anderem, dass pro Iteration nur der Teil, welcher hinzukommen sollte, genauer geplant und umgesetzt werden musste. Diese Art der Umsetzung wäre in den alternativen Vorgehensmodellen kaum oder nur sehr schwer realisierbar gewesen. Die Wahl des agilen Vorgehensmodells für dieses Projekt wurde auch durch den positiven Ausgang und Ablauf des Projektes bestätigt.

6 Glossar/ Abkürzungen

AES (Advanced Encryption Standard) ist ein deterministisches Verschlüsselungsverfahren, bei dem durch einen Schlüssel ein Text fester Länge in ein Chiffre fester Länge transformiert wird.

AES-NI (Advanced Encryption Standard New Instructions) ist eine Erweiterung zur x86-Befehlssatzarchitektur für Mikroprozessoren von Intel und AMD. Man kann hiermit eine Verbesserung der Geschwindigkeit von Anwendungen, welche AES-Ver- und Entschlüsselungen nutzen, erzielen.

ARP (Address Resolution Protocol) ist ein Protokoll, mit dem Netzwerkadressen auf Hardwareadressen abgebildet werden können, damit eine Kommunikation auf dem Network Layer stattfinden kann.

Chiffre ist ein Geheimtext, der unter Verwendung eines Schlüssels mit kryptographischen Verfahren derart verändert wurde, dass es nicht mehr möglich ist, dessen Inhalt zu verstehen.

CPH (Control Paket Hub) regelt die Verarbeitung von Schlüsselpaketen innerhalb PECTOs.

cxxtests ist ein Framework, welches zur Erstellung von Unit-Tests verwendet wird.

Dispatch-Komponente steuert die Einteilung der Pakete (verschlüsselt/unverschlüsselt) für das System.

DPDK (Data Plane Development Kit) ist eine Sammlung von Bibliotheken und Netzwerkkontrolltreibern, die zur schnellen Paketverarbeitung genutzt werden kann.

EAL (Environment Abstraction Layer) ist eine Hardwareabstraktionsschicht, die erzeugt wird, um direkte Anfragen an die Hardware leichter zu stellen und die allgemeine Nutzung zu vereinfachen.

Effizienz ist das Ausmaß der Sparsamkeit des Systems bezüglich seiner Ressourcen. Ziel sind insbesondere ein geringer Speicherverbrauch, eine geringe CPU-Last und eine hohe Paketrate.

IV-Space ist der separiete Zahlenraum, welcher jeder Instanz des Systems individuell zugeordnet wird, um unterschiedliche Initialisierungsvektoren zu erstellen.

Layer-2-Switch ist ein einfaches Kopplungsgerät, das lokale Netzwerksegmente miteinander verbindet und eine Weiterleitfunktion der Datenpakete, auf dem Data Link Layer, übernimmt. Sie haben insbesondere keine Vermittlungs- und Routingfunktionen.

Logging ist das automatische Speichern von Datenänderungen, welche in Logdateien hinterlegt werden.

Mock ist ein Objekt, welches das Verhalten eines realen Objektes nachbildet, und für Unit-Tests verwendet wird.

Network Abstraction-Komponente abstrahiert die Verwendung des DPDK und bildet die Schnittstelle zum übrigen System.

Paketdurchsatz ist die Anzahl der Pakete, die in einer bestimmten Zeit gesendet werden können.

Passphrase ist eine Zeichenfolge, über die der Zugriff auf ein Netzwerk gesteuert wird.

Portabilität ist die Möglichkeit das System auf einem anderen Betriebssystem einzusetzen.

Robustheit ist die Fähigkeit, auch unter ungünstigen Bedingungen zuverlässig zu funktionieren. Sie dürfen zu keinerlei Problemen führen.

Sicherheit ist die Fähigkeit, dass Systemfunktionen nicht von einer dritten Person abgehört oder manipuliert werden können.

Skalierbarkeit ist die Fähigkeit eines Systems, die Leistung durch das Hinzufügen von Ressourcen zu steigern.

Unit-Test ist ein Test, der verwendet wird, um Einzelteile von Computerprogrammen auf korrekte Funktionalität zu testen.

Zuverlässigkeit ist die Fähigkeit, dass ein Programm während einer gewissen Betriebsdauer nur begrenzt viele Fehlerfälle aufweisen darf.

Polling bezeichnet in der Informatik die Methode, den Status eines Geräts aus Hard- oder Software oder das Ereignis einer Wertänderung mittels zyklischem Abfragen zu ermitteln.

GCM ist ein Betriebsmodus, in der Blockchiffren für eine symmetrische Verschlüsselungsanwendung betrieben werden können

Bug bezeichnet im Allgemeinen ein Fehlverhalten von Computerprogrammen.

VPN steht für Virtual Private Network. Dient dazu, Teilnehmer des bestehenden Kommunikationsnetzes an ein anderes Netz zu binden.

Memory Pool ist ein dynamischer Speicher mit festen Blockgrößen.