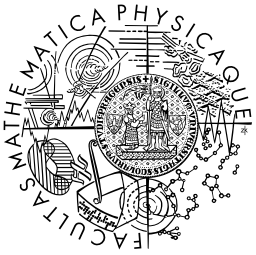


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Peter Lakatoš

Analyzátor USB paketů

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D.
Studijní program: Informatika
Studijní obor: Programování a softwarové systémy

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Analyzátor USB paketů

Autor: Peter Lakatoš

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: USB zbernica je dnes jedným z najrozšírenejších spôsobov pripojenia periférií k počítaču. Cieľom práce bolo vytvoriť software, ktorý analyzuje zachytenú komunikáciu medzi zariadnením pripojeným na danú zbernicu a počítačom.

Aplikácia následne rozumným spôsobom vizuálne zobrazuje zanalyzované dáta. Počiatočná verzia sa špecificky zameriava na HID triedu zariadení a ponúka aj sémantický význam jej úzkej podmnožiny do ktorej patria myš, klávesnica a joystick. Pri vizuálnej reprezentácii dát sa práca inšpiruje rôznymi dostupnými softwarmi, pričom rozlične kombinuje resp. dopĺňa ich vlastnosti a implementuje z nich tie, ktoré vníma ako najlepšie riešenie v danej situácii.

Ďalšie vlastnosti aplikácie sú napríklad parsovanie HID Report Descriptoru vďaka ktorému je jednoduchšie pridať sémantickú analýzu rôznym ďalším HID zariadeniam. Celkový návrh aplikácie by mal ponúknuť možnosť budúcej implementácie ďalších USB tried pre prípadné rozšírenie.

Klíčová slova: USB HID

Title: USB Packet Analyzer

Author: Peter Lakatoš

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: key words

Obsah

1 Úvod **3**
- 1.1 Základné pojmy . . . 3
- 1.2 Existujúce aplikácie . . . 8
- 1.3 Požadované funkcie . . . 13
- 1.4 Ciele práce . . . 15

2 USB a Windows **16**
- 2.1 USB zbernica . . . 16
- 2.2 Device object a device stack . . . 16
  - 2.2.1 Drivery . . . 16
- 2.3 Komunikacia s USB zariadenim . . . 16
- 2.4 USB descriptory . . . 16
  - 2.4.1 Rozlozenie USB zariadenia z hladiska descriptorov . . . 16
- 2.5 HID zariadenia . . . 17
  - 2.5.1 Reporty . . . 17
  - 2.5.2 Report Descriptor . . . 17

3 Analýza **18**
- 3.1 Získanie USB packetov . . . 18
  - 3.1.1 Windows exclusive mód . . . 18
  - 3.1.2 Známe knižnice . . . 19
  - 3.1.3 Driver . . . 20
  - 3.1.4 Third-party aplikácie . . . 20
- 3.2 Spracovávanie pcap súborov . . . 20
- 3.3 Sémantická reprezentácia dát . . . 20
- 3.4 Voľba frameworku . . . 21
- 3.5 Zobrazenie základných informácií . . . 21
- 3.6 Zobrazenie sémantického významu dát . . . 21
- 3.7 Hexdump . . . 21

4 Vývojová dokumentácia **22**
- 4.1 Architekrúra aplikácie . . . 22
- 4.2 Jadro aplikácie . . . 22
  - 4.2.1 USB_Packet_Analyzer . . . 22
  - 4.2.2 Item Manager . . . 22
  - 4.2.3 DataViewer . . . 22
  - 4.2.4 TreeItem . . . 22
- 4.3 Modely . . . 22
  - 4.3.1 AdditionaldataModel . . . 22
  - 4.3.2 ColorMapModel . . . 22
  - 4.3.3 DataViewerModel . . . 22
  - 4.3.4 TreeItemBaseModel . . . 22
  - 4.3.5 USBPcapHeaderModel . . . 22
- 4.4 Interpretery . . . 23
  - 4.4.1 BaseInterpreter . . . 23

4.4.2 Interpreter factory . . . 23
4.4.3 Interpretery descriptorov . . . 23
4.4.4 Interrupt transfer interpretery . . . 23
4.5 Delegáti . . . 23
4.6 HID . . . 23
4.6.1 HIDDevices . . . 23
4.7 Práca so súbormi . . . 23
4.7.1 FileReader . . . 23
4.8 Globálne dáta . . . 24
4.8.1 ConstDataHolder . . . 24
4.8.2 PacketExternStructs . . . 24

5 Možnosti rozšírenia 25
5.1 Ukladanie výstupu do súboru . . . 25
5.2 Iná vizuálna reprezentácia dát . . . 25
5.3 Pridávanie nových interpreterov pre descriptory . . . 25
5.4 Pridanie intepreteru na interrupt tranfser . . . 25
5.4.1 Pridanie nových HID zariední . . . 25
5.5 Pridanie analýzy pre isochronous a bulk transfer . . . 25
5.6 ?Možnosť rozšírenia na iné platformy? . . . 25

6 Užívateľská dokumentácia 26
6.1 Inštalácia . . . 26
6.2 Orientácia v GUI aplikácie . . . 26
6.3 Používanie aplikácie . . . 26

7 Záver 27
7.1 Zhrnutie . . . 27
7.2 Budúce plány . . . 27

Zoznam použitej literatúry 28

Zoznam obrázkov 30

Zoznam tabuliek 31

Seznam použitých zkratek 32

Prílohy 33
.1 První příloha . . . 34

1. Úvod

USB je najrozšírenejšia zbernica na pripojenie rôznych periférií k počítaču. Vznikla v druhej polovici 90. rokov 20. storočia, kedy boli rôzne zariadenia a ich porty veľmi úzko mapované. Na pripojenie základných zariadení ako myš alebo klávesnica slúžil napríklad sériový port PS/2 [1]. K pripojeniu tlačiarne sa často používal paralelný port Centronics [2]. Ešte pred PS/2 portom sa myš pripájala cez veľmi známy sériový port RS-232 [3]. Všetky tieto porty boli typu *point-to-point* – na jeden port je možné pripojiť len jedno zariadenie. Toto sa paralelným portom dalo čiastočné obísť tým, že niektoré zariadenia podporovali tzv. *daisy chain* – do pripojeného zariadenia sa pripojí ďalšie zariadenie, do toho sa pripojí ďalšie, atď. (napríklad typická tlačiareň toto nepodporovala, takže musela byť pripojená na konci *daisy chainu*). Existovala takisto paralelná *SCSI* zbernica [4], ktorej návrh bol prispôsobený aby podporoval *daisy chain*. Tá fungovala dobre na zariadeniach ako externé HDD a skenery, ale bola nepraktická pre zariadenia ako myš a klávesnica.

USB vzniklo za účelom nahradiť a zjednotiť tieto spôsoby pripojenia bežných periférií k počítaču. Návrh zbernice je založený na hviezdicovej topológii, ktorá umožňuje cez jeden port pripojiť až 127 zariadení súčasne. Z toho vyplýva, že USB interface v sebe zahŕňa obrovskú množinu protokolov, ktoré sú hierarchicky usporiadané. K analýze paketov ktoré sa pohybujú na danej zbernici nám slúžia tzv. USB paket analyzátory. Tie môžu mať podobu hardwarového zariadenia, alebo softwarovej aplikácie. Môžu slúžiť napríklad ako učebná pomôcka pre účel lepšieho pochopenia jednotlivých protokolov. Takisto sa často využívajú pri implementácii vlastného USB zariadenia na ladenie komunikácie medzi daným zariadením a driverom. Využitie ale majú aj v opačnom prípade, keď implementujeme vlastný driver a potrebujeme sledovať jeho komunikáciu s konkrétnym zariadením. Cieľom tejto práce bude naprogramovať funkčný softwarový USB analyzátor, ktorý by mal presnejšie slúžiť práve ako učebná pomôcka pre lepšie pochopenie určitej množiny protokolov. Cieľová platforma aplikácie bude Windows.

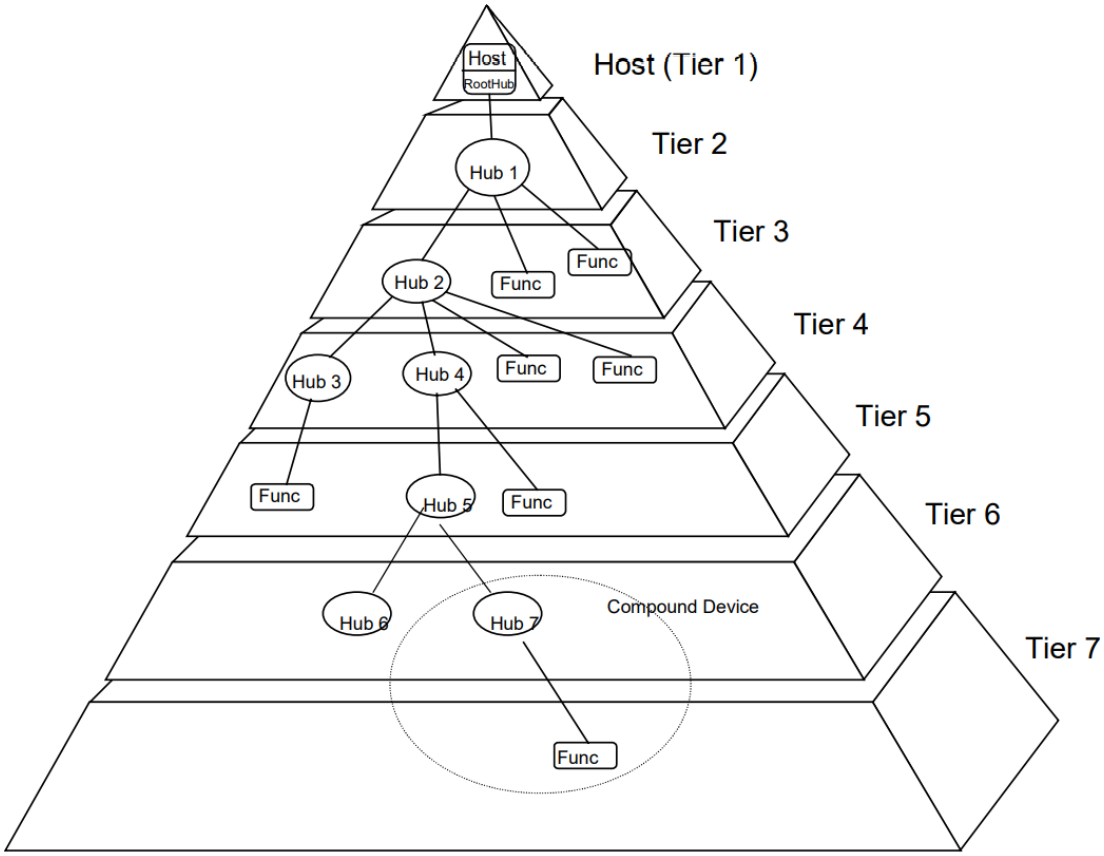
1.1 Základné pojmy

V tejto sekcii si vysvetlíme niektoré základné pojmy ktoré budeme neskôr v texte používať.

Ako sme už vyššie spomínali a ako ilustruje obrázok 1.1, USB zbernica je založená na vrstevnatej hviezdicovej topológii. Na vrchu všetkého sa nachádza **USB Host** [6] čo je systém do ktorého sa pripájajú ostatné USB zariadenia (v našom prípade je *USB Host* počítač). **USB zariadenie** [7] je buď:

- **Hub** [8] – poskytuje dodatočné pripojenia k USB zbernici.
- **Funkcia** [9] – poskytuje novú funkcionalitu systému (napríklad joystick, reproduktory, myš a pod.)

V každom USB systéme sa nachádza práve jeden *USB Host*. Ten má integrovaný tzv. **Root Hub** [6], ktorý poskytuje možné body pripojenia pre ďalšie zariadenia. Interface medzi hostom a USB sa nazýva **Host Controller** [6]. Vzhľadom



Obr. 1.1: USB topológia. Obrázok prevzatý z USB 2.0 špecifikácie [5].

na niektoré časové obmedzenia USB je maximálny počet vrstiev 7 (vrátane *USB Host* vrstvy). Každý káblový segment je *point-to-point* [10] spojenie medzi:

- *host* $\longleftrightarrow$ *hub/funkcia*
- *hub* $\longleftrightarrow$ *hub/funkcia*

USB zariadenia využívajú tzv. **descriptory** na predávanie informácií o sebe samých. **Descriptor** [11] je dátová štruktúra s predom definovaným formátom. Existuje viacero typov *USB descriptorov* (device, endpoint, interface atď.), ktorých význam si vysvetlíme neskôr.

Vzhľadom na hierarchickú štruktúru USB protokolov sa *USB zariadenia* delia na rôzne triedy [12]. USB trieda je zoskupenie zariadení (alebo interfacov) ktoré majú spoločné vlastnosti alebo funkcionalitu. Tieto triedy umožňujú USB hostovi identifikovať dané zariadenie a jeho funkcionalitu. Každá trieda má svoju vlastnú *Class Specification* – definuje správanie zariadení v jednotlivých triedach a opisuje ich komunikačný protokol, ktorý sa naprieč triedami líši. Takisto definuje rôzne descriptory, ktoré sú špecifické pre danú triedu. Príklady USB tried a jednotlivých zariadení ktoré do nich patria sú:

- Mass Storage (napr. SD karta a flash disk)
- Audio (napr. reproduktory a slúchadlá)
- HID – Human Interface Device (napr. myš, klávesnica alebo joystick)

Paket [13] je súbor dát zoskupený na prenos po zbernici. Typicky sa skladá z troch častí:

- základné informácie o danom pakete (napríklad zdroj, cieľ, dĺžka) – takisto nazývané hlavička paketu
- samotné dáta
- detekcia chýb, opravné bity

Komunikácia na zbernici medzi *USB hostom* a *zariadením* prebieha práve pomocou prenosu *USB paketov*. Existujú 4 typy takýchto prenosov [14]:

- **Control Transfer** – používa sa na nakonfigurovanie USB zariadenia v momente keď sa pripojí na zbernicu.
- **Bulk Data Transfer** – typicky pozostáva z väčšieho množstva dát ktoré sú posielané nárazovo (využívajú ho najmä tlačiarne alebo skener). Vďaka detekcii chýb na hardwarovej úrovni je zaistená správnosť prenesených dát.
- **Interrupt Data Transfer** – spoľahlivý prenos ktorý sa využíva hlavne na odovzdávanie aktuálnych informácií (ako napríklad pohyb myšou). Tieto informácie musia byť doručené USB zbernicou za čas kratší ako má špecifikované dané zariadenie.
- **Isochronous Data Transfer** – takisto nazývaný ako streaming v reálnom čase. Typický príklad je prenos zvuku.

Našu aplikáciu by sme chceli zamerať na Windows a tak si vysvetlíme ešte zopár špecifických pojmov, ktoré sa viažu na túto kokrétnu platformu.

Podľa MSDN dokumentácie [15] je **USB client driver** software nainštalovaný na počítači, ktorý komunikuje s USB zariadením aby spojazdnil jeho funkcionalitu. Žiaden *USB client driver* ale nemôže priamo komunikovať so svojím zariadením. Namiesto toho vytvorí požiadavku, súčasťou ktorej je dátová štruktúra nazývaná **URB** [16] (USB Request Block). Tá opisuje detaily požiadavku, takisto ako aj status o jeho vykonaní.

Na záver si ešte zadefinujeme rozdiel medzi *USB paket analyzátorom* a *USB paket snifferom*. Pod pojmom **USB paket sniffer** budeme rozumieť aplikáciu, ktorá monitoruje dianie na USB zbernici a je schopná ho rozumným spôsobom ukladať v predom definovanom formáte. Ako **USB paket analyzátor** budeme brať aplikáciu ktorá je schopná rozanalyzovať USB pakety (istým spôsobom ich vyobraziť alebo ukázať ich sémantický význam) uložené v predom definovanom formáte. Bežne sa tieto pojmy označujú za jednu a tú istú vec, aj keď ich funkcionalita spolu nijako priamočiaro nesúvisí a existujú nástroje, ktoré vedia len jedno alebo druhé. Preto dáva zmysel ich od seba explicitne oddeliť.

Momentálne by sme mali chápať všetky základné pojmy týkajúce sa USB, a tak si poďme trochu bližšie objasniť zameranie našej aplikácie. Našu aplikáciu zameriavame výukovým smerom pre programátorov, ktorí chcú lepšie pochopiť komunikáciu na USB zbernici. Z toho dôvodu by sme v nej určite chceli zahrnúť analýzu základných USB descriptorov, ktoré sú bližšie definované v špecifikácii USB 2.0 [17] v kapitole 9.6. Keďže chceme bližšie priblížiť komunikáciu na danej zbernici, potrebujeme konkrétne zariadenia, s ktorými ju budeme analyzovať.

Dáva dobrý zmysel si zvoliť zariadenia, ktoré každý z nás dobre pozná, má ich k dispozícii a bežne ich využíva. Zároveň by ale mali mať dostatočne jednoduchý komunikačný protokol. Práve preto sa s našou aplikáciou zameriame na užšiu podmnožinu HID zariadení, konkrétne myš, klávesnica a joystick. Vzhľadom na zameranie našej aplikácie výukovým smerom prikladáme najväčšiu prioritu samotnej analýze dát. Z dôvodu celkovej univerzality USB je z didaktického hľadiska ťažké nasimulovať jednotný príklad u každého študenta zvlášť. Už len obyčajná myš, aj keď je to jedno zariadenie, má od rôznych výrobcov inak nadefinované správanie a posiela dáta v rozličných formátoch. Preto je pre nás dôležité vedieť analyzovať pakety, ktoré si učiteľ predpripraví, skontroluje ich didaktickú správnosť a uloží do súboru. Podpora živého zachytávania paketov a ich analýzy je tak v našom programe najmenej dôležitá.

Keďže sa v našej práci budeme venovať hlavne analýze HID zariadení, tak si túto USB triedu rozoberieme trochu detailnejšie.

HID

Podľa dodatku k USB špecifikácii [18] je **HID** (z anglického „Human Interface Device“) USB trieda pozostávajúca prevažne zo zariadení, ktoré sú využívané človekom na riadenie určitých systémovových aplikácií. Medzi najpoužívanejšie príklady patrí myš, klávesnica alebo joystick.

Ako sme už spomínali vyššie, jednotlivé USB triedy majú definované vlastné descriptory špecifické pre danú triedu. Jedným z takýchto descriptorov je aj *Report Descripor*. Ten popisuje dáta, ktoré generuje konkrétne zariadenie. Analýzou *Report Descriporu* sme schopní určiť veľkosť a kompozíciu dát posielaných zariadením. Z toho vyplýva, že komunikácia HID zariadením s USB hostom sa môže líšiť nie len vo veľkosti posielaných dát, ale takisto aj v ich význame.

Lepšie to uvidíme na konkrétnom príklade. K dispozícii máme 2 rozdielne myši – Genius DX-120 [19] a Logitech G502 Proteus Spectrum [20]

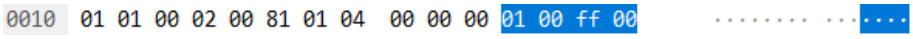
(a) Fotka genius myši prevzatá z oficiálnej genius stránky [21]

(b) Fotka logitech myši prevzatá zo stránky obchodu [22]

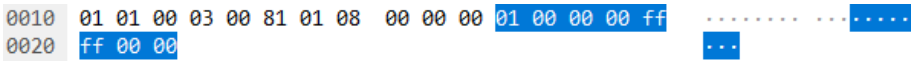
Obr. 1.2: Ukážka myší, ktorých input budeme porovnávať

Teraz si ukážeme ako sa líši ich input. Dáta budeme vyobrazovať pomocou obyčajného hexdumpu (zvýraznená časť v hexdumpe reprezentuje input zariadenia). Na oboch myšiach stlačíme ľavé tlačidlo a mierne ich posunieme smerom

hore. Dáta, ktoré poslala genius myš sú vyobrazené na obrázku 1.3 a dáta poslané logitech myšou môžeme vidieť na obrázku 1.4.



Obr. 1.3: Ukážka hexdumpu so zvýrazneným inputom genius myši.



Obr. 1.4: Ukážka hexdumpu so zvýrazneným inputom logitech myši.

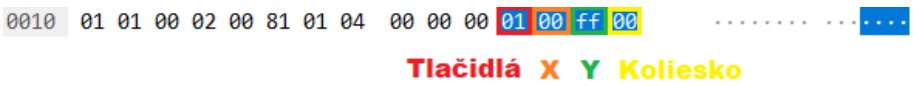
Aj napriek tomu, že sa jedná o zariadenia z tej istej USB triedy a dokonca o rovnaké zariadenie – myš, má ich komunikácia rozličný tvar definovaný priamo výrobcom zariadenia. Analýzou *Report Descriporu* (o ktorej si viac povieme neskôr v práci) sme zistili, že dáta, ktoré poslala genius myš majú nasledujúci význam:

- Byte 0: bity 0–2 reprezentujú stlačenie jednotlivých tlačidiel, bity 3–7 tvoria len dodatočnú výplň bytu
- Byte 1: reprezentuje súradnicu X
- Byte 2: reprezentuje súradnicu Y
- Byte 3: reprezentuje koliesko myši

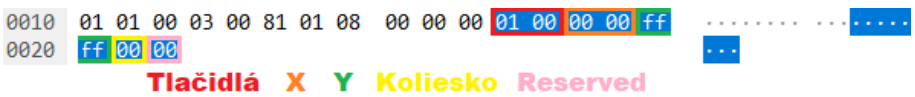
Pre porovnanie, význam dát poslaných logitech myšou je nasledovný:

- Byte 0–1: reprezentujú stlačenie jednotlivých tlačidiel
- Byte 2–3: reprezentuje súradnicu X
- Byte 4–5: reprezentuje súradnicu Y
- Byte 6: reprezentuje koliesko myši
- Byte 7: je rezervovaný výrobcom myši

Vizuálne zobrazený význam dát genius myši je ukázaný na obrázku 1.5 a logitech myši na obrázku 1.6.



Obr. 1.5: Ukážka hexdumpu so zvýrazneným inputom genius myši s významom.



Obr. 1.6: Ukážka hexdumpu so zvýrazneným inputom logitech myši s významom.

Z toho vyplýva, že aby sme boli schopní vykonať sémantickú analýzu HID zariadení, bude pre nás kľúčové vedieť rozparsovať *Report Descripor* a na základe toho interpretovať input zariadení.

1.2 Existujúce aplikácie

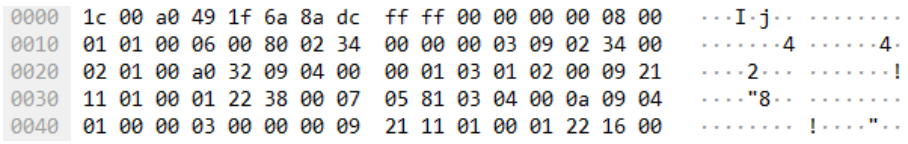
Momentálne existuje niekoľko známych aplikácií ktoré slúžia na analýzu USB paketov. Ich predbežným skúmaním a používaním sme ale zistili, že úplne nevyhovujú našim konkrétnym požiadavkám. Avšak mnohé ich funkcie nám prídu užitočné a môžu poslúžiť ako inšpirácia v implementovaní našej aplikácie. V tejto kapitole si ukážeme výhody a nevýhody zopár aplikácií, ktoré sme si zvolili ako príklady v oblasti paket analyzátorov. Ich výber spočíval v tom, že sú veľmi rozšírené medzi verejnosťou a sú najbližšie k tomu čo by sme chceli od našej aplikácie.

Je nutné upozorniť, že väčšina dnešných analyzátorov sú platené aplikácie, prípadne majú odomknuté len základné vlastnosti s možnosťou dokúpenia si plnej verzie. Práve preto sme nemali možnosť si pri všetkých vyskúšať ich celú funkcionalitu a na niektoré platené funkcie máme tak len ilustračný pohľad.

Wireshark

Aplikácia, ktorá na prvý pohľad nesúvisí s USB zbernicou. Wireshark je pravdepodobne najznámejší analyzátor a sniffer sieťových paketov. Jeho funkcionalita je veľmi rozsiahla, a vzhľadom na to, že sa jedná o open-source projekt, neustále rastie. Vďaka jeho obecnému návrhu podporuje spoluprácu s rôznymi inými sniffermi (LANalyzer, NetXRay a pod.). Jeden z takýchto snifferov je *USBPcap*, ktorý zachytáva USB komunikáciu a tým pádom je Wireshark schopný analyzovať pakety aj nad touto zbernicou.

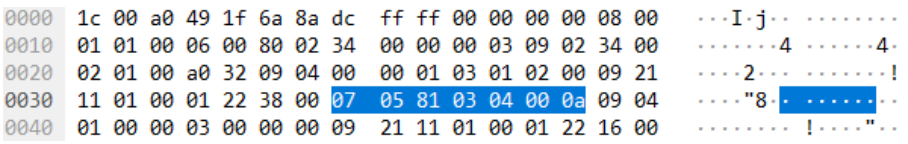
Pre priblíženie niektorých funkcií Wiresharku si ukážeme analýzu komunikácie s USB myšou (Genius DX-120 [19]). Medzi tie úplne základné funkcie určite patrí hexdump dát nad ktorými prebieha analýza, ktorý je vyobrazený na obrázku 1.7.



Obr. 1.7: Ukážka hexdumpu vo Wiresharku.

Tento hexdump je tvorený dátami z jedného control prenosu, kde zariadenie posiela informácie o sebe samom v podobe rôznych descriptorov.

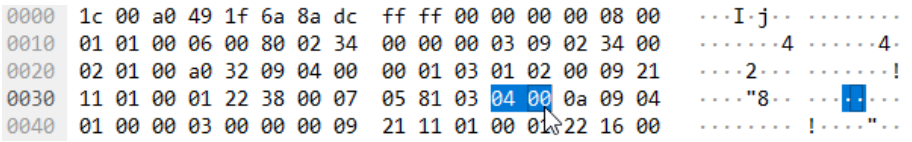
V hexdumpe si takisto vieme pomocou kliknutia a ťahania myšou označiť ľubovoľné dáta, ktoré chceme. Zvýraznené byty na obrázku 1.8 reprezentujú jeden *endpoint descriptor*.



Obr. 1.8: Ukážka hexdumpu so zvýrazneným endpoint descriptorom.

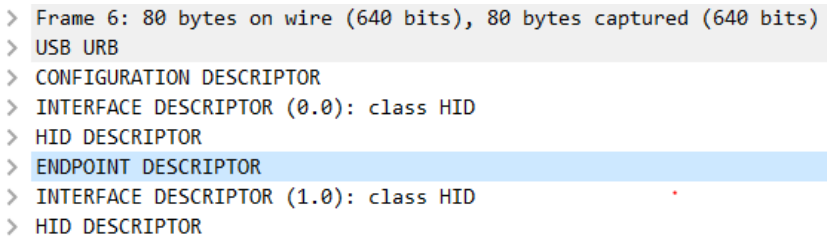
Pri pohybe myšou nad daným hexdumpom ponúka Wireshark interaktívnu odozvu, pričom farebne oddeľuje jednotlivé byty podľa ich významu. Na obrázku 1.9 vidíme konkrétny príklad – ak podržíme myš nad hexa časťou bytu

00, automaticky nám to označí aj byte 04 pred ním, pretože spoločne reprezentujú jednotnú informáciu – položku *wMaxPacketSize* v *endpoint descriptore*.



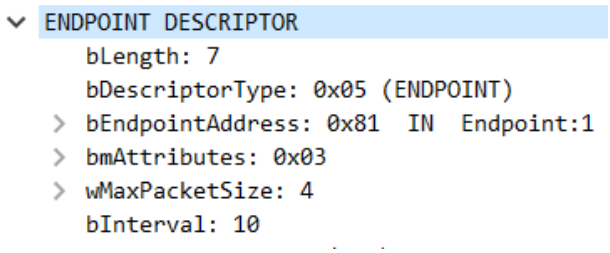
Obr. 1.9: Ukážka hexdumpu s farebným oddelením na základe významu.

Ďalšia užitočná vlastnosť je, že pri označení hexa znakov v hexdumpe, sa samé označia aj im odpovedajúce tlačiteľné znaky (obdobne to funguje aj opačným smerom). To, že vyššie označených 7 bytov na obrázku 1.8 reprezentujú *endpoint descriptor* sme zistili vďaka špecifikácii jednotlivých descriptorov a vlastnou analýzou bytov v hexdumpe. Wireshark ale ponúka rozličné zobrazenie tých istých dát, a to napríklad aj pomocou stromovej štruktúry, ktorá už jednotlivým bytom pridáva ich sémantický význam v slovnom tvare ako je ukázané na obrázku 1.10 nižšie.



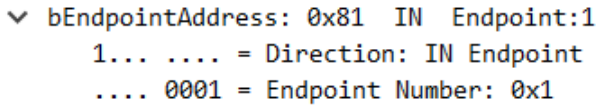
Obr. 1.10: Ukážka reprezentácie dát pomocou stromovej štruktúry.

Jednotlivé položky si môžeme bližšie rozbaliť. Napríklad vyššie zvýraznených 7 bytov reprezentujú konkrétny *endpoint descriptor*, ktorý je ukázaný na obrázku 1.11. Na tom istom obrázku si takisto môžeme všimnúť, že položka *wMaxPacketSize* má hodnotu 4, čo je presne hodnota bytov 04 00, ktoré sme spomínali vyššie na obrázku 1.9



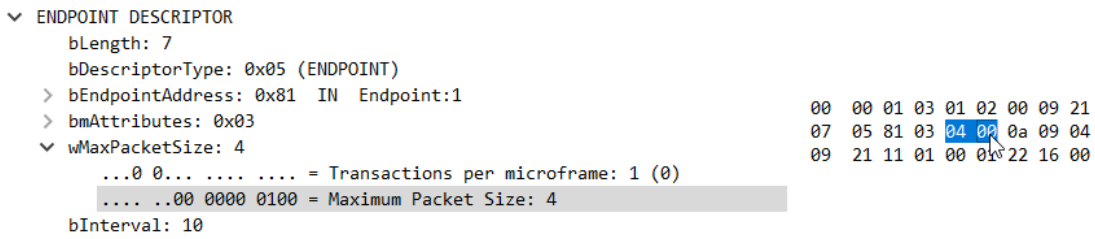
Obr. 1.11: Endpoint descriptor reprezentovaný dátami zvýraznenými na obrázku 1.7 vyššie.

Medzi viac špecifické funkcie patrí detailnejšie vyobrazenie jednotlivých bytov a ich význam, ako je možné vidieť nižšie na obrázku 1.12. Na tomto obrázku vidíme rozbalenú položku *bEndpointAddress*, ktorej hodnota je 0x81. Siedmy bit tejto hodnoty reprezentuje smer endpointu (IN – slúži na prenos dát device $\longrightarrow$ host, OUT opačne) a dolné 4 bity označujú číslo endpointu. Túto vlastnosť aj napriek jej využitiu mnohé konkurenčné aplikácie postrádajú.



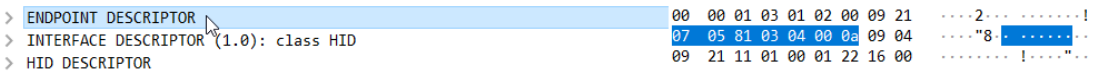
Obr. 1.12: Ukážka vyobrazenia jednotlivých bytov.

Wireshark ponúka interaktívne užívateľské rozhranie. V prípade kliknutia na konkrétny byte v hexdumpe sa nám označí jemu odpovedajúca položka v stromovej štruktúre. Príklad je ukázaný na obrázku 1.13.



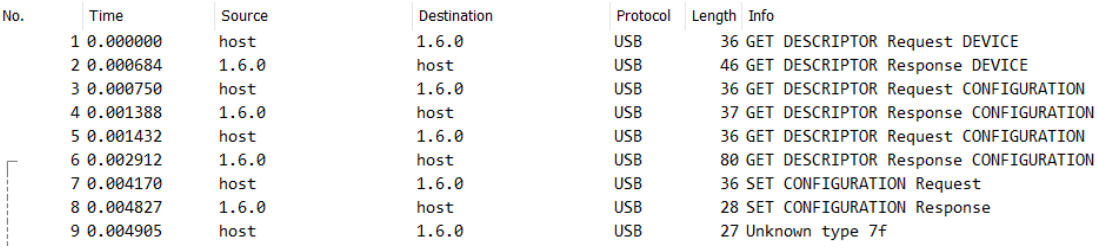
Obr. 1.13: Ukážka kliknutia na položku v hexdumpe.

Podobne to funguje aj opačne, takže ak klikneme na položku v stromovej štruktúre, označí sa jej odpovedajúca časť v hexdumpe. Príklad kliknutia na *endpoint descriptor* v stromovej štruktúre a označenia jemu odpovedajúcej časti hexumpu je vidieť na obrázku 1.14.



Obr. 1.14: Ukážka kliknutia na položku *endpoint descriptoru* v stromovej štruktúre.

Obecné vyobrazenie pohybu paketov na zbernici bez hlbšej analýzy je ukázané na obrázku 1.15 nižšie.



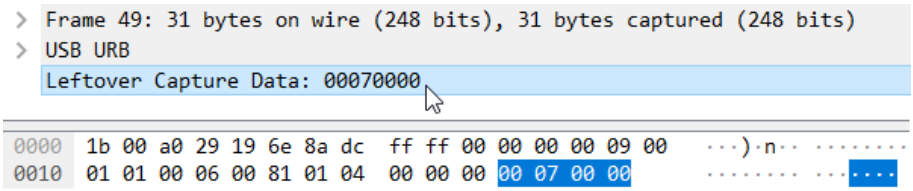
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	host	1.6.0	USB	36	GET DESCRIPTOR Request DEVICE
2	0.000684	1.6.0	host	USB	46	GET DESCRIPTOR Response DEVICE
3	0.000750	host	1.6.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
4	0.001388	1.6.0	host	USB	37	GET DESCRIPTOR Response CONFIGURATION
5	0.001432	host	1.6.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
6	0.002912	1.6.0	host	USB	80	GET DESCRIPTOR Response CONFIGURATION
7	0.004170	host	1.6.0	USB	36	SET CONFIGURATION Request
8	0.004827	1.6.0	host	USB	28	SET CONFIGURATION Response
9	0.004905	host	1.6.0	USB	27	Unknown type 7f

Obr. 1.15: Príklad obecného vyobrazenia jednotlivých paketov vo Wiresharku.

Výhoda Wiresharku je hlavne v tom, že podporuje širokú škálu descriptorov a plná verzia programu je dostupná úplne zadarmo. Z pohľadu užívateľa je až prekvapivé, že aj napriek rozsiahlosti programu je aplikácia veľmi user-friendly orientovaná a dopĺňa ju intuitívne užívateľské rozhranie.

Naopak, jeho nevýhodou je sčasti neprehľadný hexdump. Ako môžeme vidieť na obrázku 1.7, jedná sa o obyčajný hexdump, ktorý nijakým spôsobom neoddeľuje význam dát bez interakcie užívateľa. Preto v momente ak by sme nemali

stromovú štruktúru k odpovedajúcemu hexdumpu, museli by sme sa riadiť špecifikáciou a vlastnou analýzou. V prípade rozsiahlejšieho hexdumpu môže byť veľmi obtiažné sa v ňom potom zorientovať. Ďalšia vec ktorá nám nevyhovuje, je chýbajúca sémantická analýza inputu rôznych zariadení. Ten je vyobrazený len pomocou hexdumpu a popisu „Leftover Capture Data“ ako je ukázané na obrázku 1.16. Zo sekcie 1.1 nám je teda jasné, že vôbec netušíme čo jednotlivé dáta znamenajú, pretože ich význam je definovaný v *Report Descriptore*.

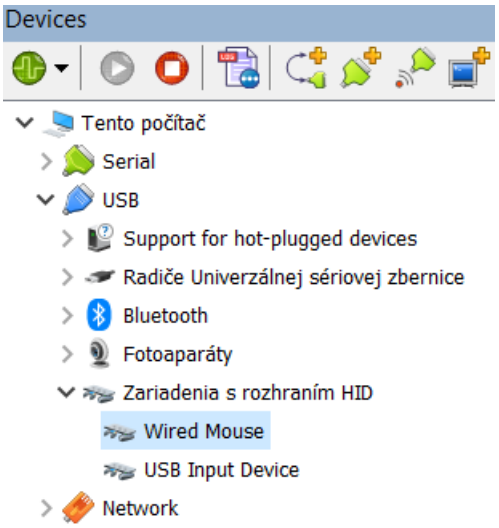


Obr. 1.16: Príklad inputu myši vo Wiresharku.

Device Monitoring Studio

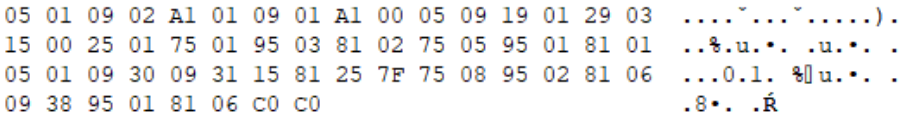
Aplikácia ponúka analýzu sieťových a USB paketov, tak ako aj analýzu komunikácie prebiehajúcej cez sériový port. Zároveň slúži aj ako sniffer na všetkých týchto portoch.

Ako prvé na aplikácii zaujme spôsob zvolenia si zariadenia s ktorým bude sledovaná komunikácia. Je implementovaný štýlom stromovej štruktúry ako je ukázané na obrázku 1.17 nižšie, kde máme konkrétne označenú rovnakú myš s ktorou komunikáciu sme sledovali predchádzajúcim programom.



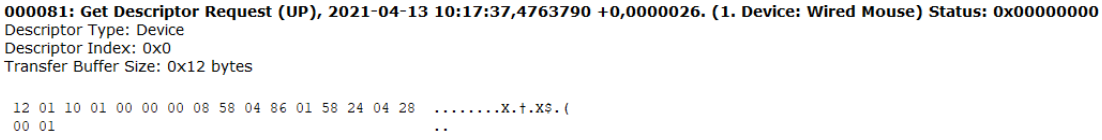
Obr. 1.17: Ukážka stromovej štruktúry na zvolenie si zariadenia, s ktorým bude zachytávaná komunikácia.

Základná verzia programu ponúka vizuálne zobrazenie *URB*, tak ako aj analýzu jednotlivých paketov. Pod analýzou si tu môžeme predstaviť ale len obyčajný hexdump, ktorý neposkytuje žiadne významové oddelenie dát a tým pádom je obtiažnejšie sa v ňom zorientovať. Príklad môžeme vidieť na obrázku 1.18.



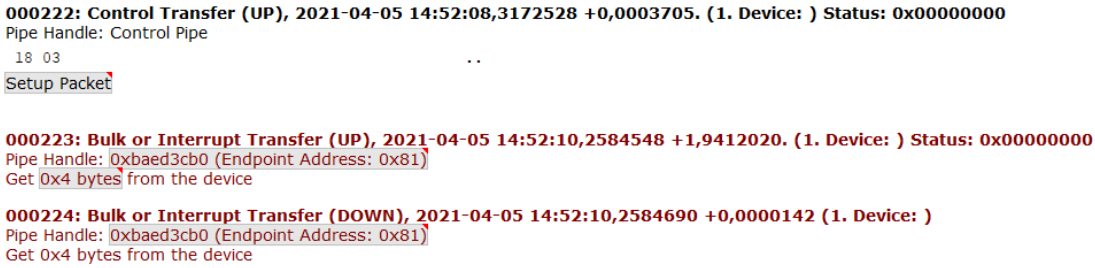
Obr. 1.18: Príklad hexdumpu v Device Monitoring Studio.

Takisto tu nemáme kompletné sémantické vysvetlenie čo dané dáta znamenajú (napríklad pomocou stromovej štruktúry ako to rieši konkurencia). K dispozícii máme len veľmi obmedzený popis jednotlivých paketov (číslo paketu, device request, a pod.), pričom ani nie je veľmi jasné odkiaľ sa tieto informácie vzali. Príklad takéhoto popisu aj s hexdumpom je ukázaný na obrázku 1.19 nižšie.



Obr. 1.19: Príklad analýzy paketov.

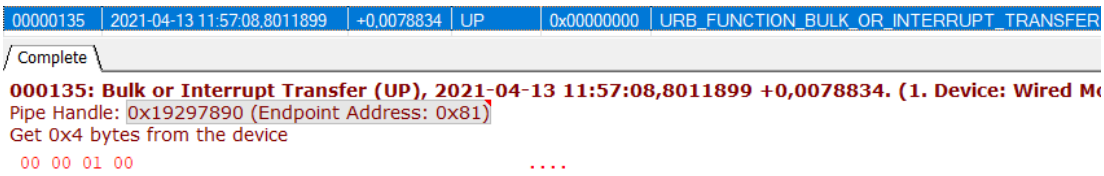
Vyobrazenie *URB* (obrázok 1.20) tak ponúka súhrn týchto popisov jednotlivých paketov, ktoré sú postupne zachytené počas komunikácie na zbernici.



Obr. 1.20: Ukážka vyobrazenia URB.

Pričom pri dvojkliku na šedé časti textu (napríklad *Setup Packet* alebo *Endpoint Address*) sa užívateľovi rozbalí okno s detailnejším popisom.

Analýza inputu myši, ktorú môžeme vidieť na obrázku 1.21, je riešená podobným spôsobom ako pri analýze descriptorov.



Obr. 1.21: Príklad inputu myši v Device Monitoring Studio.

Obecné vyobrazenie jednotlivých paketov bez bližšej analýzy je riešené podobne ako vo Wiresharku, pričom pakety sú farebne oddelené podľa ich smeru pohybu na zbernici (posielané smerom host $\longrightarrow$ zariadenie/smerom zariadenie $\longrightarrow$ host). Toto je veľmi pekná funkcionalita, ktorá celkovo sprehľadňuje komunikáciu zariadenia s hostom. Príklad je ukázaný na obrázku 1.22.

00000073	2021-04-13 10:17:35,1931283	+25,9758328	UP	0xc0000011	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER
00000074	2021-04-13 10:17:35,1932268	+0,0000985	UP	0xc0010000	URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER
00000075	2021-04-13 10:17:35,2130691	+0,0198423	UP		PnP: Device Surprise Removal
00000076	2021-04-13 10:17:35,2249868	+0,0119177	DOWN	0xffffd184	URB_FUNCTION_ABORT_PIPE
00000077	2021-04-13 10:17:35,2250067	+0,0000199	UP	0x80000300	URB_FUNCTION_ABORT_PIPE
00000078	2021-04-13 10:17:35,2250219	+0,0000152	UP		PnP: Device Disconnected
00000079	2021-04-13 10:17:37,4763688	+2,2513469	UP		PnP: Device Connected
00000080	2021-04-13 10:17:37,4763764	+0,0000076	DOWN	0x00000000	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE
00000081	2021-04-13 10:17:37,4763790	+0,0000026	UP	0x00000000	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE
00000082	2021-04-13 10:17:37,4763850	+0,0000060	DOWN	0x00000000	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE
00000083	2021-04-13 10:17:37,4763874	+0,0000024	UP	0x00000000	URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE

Obr. 1.22: Príklad obecného vyobrazenia jednotlivých paketov v Device Monitoring Studio.

Zaujímavá funkcionalita, ktorú ale program ponúka len v platenej verzii, je umožnenie užívateľovi priamo komunikovať so zvoleným zariadením. Môžeme mu tak posielať rôzne požiadavky (niektoré z nich sú spomenuté v USB 2.0 špecfikácii[17] v kapitole 9.4) ako napríklad *GET_REPORT* kde špecifikujeme *Report ID* a prípadné ďalšie parametre, a zariadenie nám patrične odpovie.

Užívateľské rozhranie vyobrazené nižšie pomocou obrázku 1.23 ,pozostáva z pomerne veľa ikoniek a celkovo sa javí ako trochu neprehľadné. Pri prvotnej interakcii s programom chvíľu trvá, kým človek nájde čo i len základné informácie ako napríklad hlavičky ku jednotlivým paketom. Nepoteší ani fakt, že verzia zadarmo nedovoľuje monitorovanie dlhšie ako 10 minút a maximálny počet monitorovaní za jeden deň je taktiež 10.

Obr. 1.23: Užívateľské rozhranie Device Monitoring Studio.

1.3 Požadované funkcie

Ako prvé by sme si mali zadefinovať platformu na ktorú budeme cieliť s našou aplikáciou:

P1 Cieľová platforma našej aplikácie by mala byť Windows.

Keďže má naša aplikácia mať výukový charakter, tak sa pozrieme na typický výukový scénár jej používania. Učiteľ si dopredu do súboru zachytí komunikáciu s určitým zariadením na ktorej overí, že je didakticky dobrá a ilustruje to čo má. Následne daný súbor posunie študentom aby si mohli zobraziť analýzu konkrétnych paketov. Užitočná je ale aj analýza priamej interakcie užívateľa s jeho konkrétnym zariadením, preto by sme zároveň chceli podporovať aby si študenti mohli pripojiť vlastné zariadenie a skúmať s ním komunikáciu v reálnom čase. Z toho nám vyplývajú naledujúce požiadavky:

P2 Mala by byť schopná analyzovať USB pakety zachytené do súboru v rozumnom formáte pomocou predom definovaného snifferu.

P3 Mala by byť schopná analýzy paketov v reálnom čase. To znamená, že bude podporovať čítanie súboru súvisle s tým ako do neho bude zapisovať iný software (za predpokladu, že to daný software povoľuje).

Ako sme mohli vidieť aj na predchádzajúcich príkladoch, hexdump je jednou zo základných funkcií na analýzu paketov. Zároveň sa nám ale nepáčilo, že väčšina hexdumpov je neprehľadná a ťažko sa v nich orientuje. Preto si zadefinujeme nasledujúce požiadavky:

P4 Mala by pomocou hexdumpu vedieť zobraziť dáta, ktoré daný sniffer zachytí a uloží.

P5 Mala by mať prehľadnejší hexdump a užívateľovi uľahčiť orientáciu v ňom. Jednotlivé znaky by mali byť farebne označené na základe ich významu (hlavička paketu, rôzne typy descriptorov a pod.).

K sémantickej analýze sa nám môže hodiť vedieť zobraziť dáta a ich význam pomocou stromovej štruktúry. Pretože sa s našou aplikáciou budeme snažiť vysvetliť základy komunikácie na USB zbernici, mali by sme podporovať sémantickú analýzu všetkých základných USB descriptorov a takisto inputu určitej podmnožiny HID zariadení. Ako posledné by sa nám zišlo vedieť pomocou stromovej štruktúry vyobraziť hlavičku jednotlivých paketov. Z toho celého dostávame nasledovné:

P6 Mala by podporovať sémantickú analýzu (vyobrazenie pomocou stromovej štruktúry) pre všetky základné USB descriptory spomenuté v USB 2.0 špecifikácii[17] v kapitole 9.6 (ako napríklad *Device descriptor*, *Interface descriptor*, *Endpoint descriptor*, atď.).

P7 Mala by byť schopná pomocou stromovej štruktúry zobraziť sémantický význam dát posielaných danou podmnožinou HID zariadení, do ktorej patrí myš, klávesnica a joystick.

P8 Mala by byť schopná pomocou stromovej štruktúry zobraziť sémantický význam jednotlivých hlavičiek paketov.

Vyššie v texte sme označili funkciu Wiresharku vyobraziť sémantický význam dát na bitovej úrovni (obrázok 1.12) za zaujímavú. Preto by sme ju chceli implementovať aj v našej aplikácii, z čoho vyplýva:

P9 V miestach kde to dáva zmysel, by aplikácia mala byť schopná zobrazovať význam dát až na úrovni jednotlivých bitov.

Nechceme užívateľov hneď zaplaviť všetkými detailnými informáciami o paketoch. Preto by sme mali vedieť zobraziť zopár obecných vecí ku každému paketu a vyobraziť tak pohyb na zbernici, a až v prípade interakcie užívateľa s aplikáciou zobraziť podrobný popis jednotlivých paketov. Zároveň sa nám páčila funkcia Device Monitoring Studia, kde boli jednotlivé pakety farebne rozlišiteľné, čo zvyšovalo celkový prehľad pohybu paketov na zbernici. Z toho dostávame nasledujúce požiadavky:

P10 Mala by na prvý pohľad jasne zobraziť základné informácie o každom analyzovanom pakete (ako napr. dĺžka paketu, typ prenosu a pod.) a pri bližšom skúmaní jednotlivých paketov detailnejšie zobraziť celú jeho hlavičku. Tieto základné informácie by mali byť farebne rozlišiteľné na základe smeru paketu po zbernici.

P11 Detailnejšie informácie o pakete budú zobrazované na základe interakcie užívateľa s aplikáciou.

Aby sme boli schopní sémantickej analýzy dát myši, klávesnice alebo joysticku podľa osobného výberu užívateľa, musíme si získať informácie o ich inpute z *HID Report Descriptoru*, takže naša ďalšia požiadavka je:

P12 Mala by byť schopná rozparsovať *HID Report Descriptor* takým štýlom, aby bolo neskôr možné sématnicky reprezentovať input nami zvolených HID zariadení – myš, klávesnica a joystick.

1.4 Ciele práce

Celkové ciele tejto práce sú následovné :

C1 Naprogramovať funkčný analyzátor, ktorý spĺňa všetky požadované funkcie **P1**-**P12**

C2 Návrh programu musí byť dostatočne obecný aby splňoval nasledujúce:

- Jednoduché rozšírenie o analýzu ďalších typov USB prenosov.
- Jednoduché pridanie sémantickej analýzy pre ďalšie HID zariadenia.

2. USB a Windows

vysvetlenie zakladnych pojmov spojenych USB: historia, usb port/conector, plug and play(https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/introductio to-plug-and-play), low/full/high speed zariadenia

2.1 USB zbernica

Plug and Play device tree(sposob akym si windows udrziava strom zariadeni na zbernici)(https://docs.microsoft.com/sk-sk/windows-hardware/drivers/gettingstarted/devic nodes-and-device-stacks)

2.2 Device object a device stack

PDO,FDO, Device object(https://docs.microsoft.com/en-us/windows-hardware/drivers/ker to-device-objects) https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/creatin a-device-object

2.2.1 Drivery

Opisat ako teda bezne analyzatory/sniffery funguju windows driver model(WDM) : https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/types-of-wdm- drivers bus driver(https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/bus- drivers), function driver(https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/f drivers) a filter driver(https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/filte drivers)

2.3 Komunikacia s USB zariadenim

sposob komunikacie operacneho systemu so zariadenim pripojenym na USB zbernicu : IRP(https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/i- o-request-packets) , URB (https://docs.microsoft.com/en-us/windows-hardware/drivers/usbco with-a-usb-device) a pod. https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/ irps

2.4 USB descriptory

opis zakladnych USB descriporov, hlavne tych ktore neskor aj vyuzivam v program(Device, Interface, Endpoint, Configuration, String, Setup) : https://docs.microsoft.com us/windows-hardware/drivers/usbcon/usb-descriptors https://docs.microsoft.com/en- us/windows-hardware/drivers/usbcon/usb-control-transfer

2.4.1 Rozlozenie USB zariadenia z hladiska descriptorov

https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/usb-device- layout

2.5 HID zariadenia

hid zariadenie obecne, priklady https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/

2.5.1 Reporty

Input/Output/Feature reporty.

2.5.2 Report Descriptor

Opis report descriptoru, k comu sluzi, pripadne ako z neho vycitat zaujimave data (neskor vyuzite v programe pri parsovani HID Report Descriptoru na naslednu semanticku analyzu dat ktore posiela zariadenie)

3. Analýza

V tejto kapitole sa pozrieme na zopár problémov a ich riešení, na ktoré sme narazili počas vytvárania nášho analyzátoru. Ako prvé sme sa museli rozhodnúť v akom jazyku budeme náš projekt implementovať. Na to si autor zvolil obecne rozšírený jazyk C++, pretože ho pozná najlepšie a má s ním najviac skúseností. Táto voľba nás zároveň v ničom zásadne neobmedzila a nemala negatívny dopad na celkový výsledok aplikácie. Teraz sa pozrieme na niekoľko konkrétnych problémov a spôsoby akými sme ich vyriešili.

3.1 Získanie USB packetov

Na získavanie USB paketov nám bude obecne slúžiť paket sniffer. Väčšina paket analyzátorov má implementované vlastné sniffery a preto sme sa o to pokúsili tiež. Narazili sme ale na niekoľko zásadných problémov, ktoré sa úzko viažu s platformou na ktorú cielime s našou aplikáciou – Windows.

Microsoft dokumentácia podrobnejšie opisuje komunikáciu medzi HID zariadením a kernel/user-mode aplikáciou [23]. Pri tejto komunikácii sme schopní zachytiť USB pakety posielané zariadením a neskôr ich analyzovať. Keďže naša aplikácia beží v user-mode, prejdeme si práve tento spôsob komunikácie:

1. Aplikácia nájde a identifikuje HID zariadenie.
2. Aplikácia pomocou metódy *CreateFile* otvorí spojenie s HID zariadením.
3. Aplikácia pomocou *HID API* [24] metód *HidD_Xxx* získa *Preparsed Data* a informácie ohľadom HID zariadenia.
4. **Aplikácia použije metódu *ReadFile* resp. *WriteFile* na získanie inputu zariadenia resp. poslanie reportu zariadeniu.**
5. Aplikácia pomocou *HID API* [24] metód *HidP_Xxx* interpretuje HID reporty.

Podstatný je práve bod 4 v ktorom vidíme, že pomocou metódy *ReadFile* sme schopní od daného zariadenia získať USB pakety, ktoré reprezentujú jeho input. Tie by sme následne mohli pomocou nášho analyzátoru spracovať. Tu narážame na prvý problém, ktorý sa priamo viaže na platformu Windows a ktorý si detailnejšie opíšeme v nasledujúcej sekcii.

3.1.1 Windows exclusive mód

Windows má definovaný tzv. *Access Mode*, ktorý určuje restrikciu prístupu *HID Clienta* k HID zariadeniu. Ten môže byť buď *Shared* alebo *Exclusive*. *Exclusive Mode* zabraňuje ostatným *HID Clientom* v zachytávaní alebo získavaní inputu HID zariadenia, pokiaľ nie sú hlavným príjemcom daného inputu. Preto z bezpečnostných dôvodov otvára *RIM (Raw Input Manager)* niektoré zariadenia v *Exclusive Mode*.

Ak je zariadenie otvorené v *Exclusive Mode*, aplikácia má stále prístup k niektorým jeho údajom pomocou *HID API* [24] metód *HidD_**Get**Xxx*. Tieto metódy nám obecne umožnia získať niektoré descriptory zariadenia, tak ako aj jeho *Preparsed Data*. Nie je nám ale umožnené volať metódu *ReadFile*, takže nemáme akým spôsobom zachytávať komunikáciu HID zariadenia s clientom.

Tabuľka zariadení [25] (obrázok 3.1), ktoré *RIM* otvára v *Exclusive Mode* obsahuje aj tie, ktoré sme si v úvode zvolili ako podmnožinu HID zariadení na analýzu – myš a klávesnica.

Windows supports the following top-level collections:

Usage Page	Usage	Windows 7	Windows 8	Windows 10	Notes	Access Mode
0x0001	0x0001 - 0x0002	Yes	Yes	Yes	Mouse class driver and mapper driver	Exclusive
0x0001	0x0004 - 0x0005	Yes	Yes	Yes	Game Controllers	Shared
0x0001	0x0006 - 0x0007	Yes	Yes	Yes	Keyboard / Keypad class driver and mapper driver	Exclusive

Obr. 3.1: Tabuľka zariadenía ich *Access Mode*. Zariadenia postupne po riadkoch – myš, joystick a klávesnica

3.1.2 Známe knižnice

Keďže komunikáciu nemôžeme zachytávať priamo pomocou *HID API* [24] metód, rozhodli sme sa skúsiť použiť niektoré známe knižnice na sledovanie USB zbernice.

Pravdepodobne najznámejšou z nich je *libusb* [26] – knižnica napísaná v jazyku C, ktorá slúži na poskytnutie prístupu k USB zariadeniu. Je cielená na programátorov aby im uľahčila vývoj aplikácií, ktoré komunikujú s USB hardwarom. Podporuje viaceré platformy, medzi ktorými sa nachádza aj Windows. Zároveň beží v user-mode, takže aplikácia ktorá ju využíva nepotrebuje žiadne špeciálne privilégiá na používanie libusb API. Ďalšou výhodou je, že podporuje všetky verzie USB od 1.0 až po 3.1. Je schopná zachytiť všetky typy prenosov (control,bulk,interrrupt,isochronus), a teda by sme pomocou nej mohli sledovať celú komunikáciu so zariadením vrátane počiatočného nakonfigurovania. Bohužiaľ, ani pomocou tejto knižnice sme neboli schopní získať prístup k zariadeniam, ktoré Windows otvára v exclusive móde.

Keďže sa zameriavame na HID zariadenia, skúsili sme hľadať niečo viac špecifické práve pre túto triedu USB zariadení, a to nás zaviedlo k *HIDAPI* [27]. *HIDAPI* je ďalšia veľmi rozšírená knižnica, ktorá umožňuje komunikáciu s USB zariadeniami z HID triedy a takisto beží na viacerých patformách vrátane Windowsu. Pretrváva tu ale rovnaký problém ako s minulou knižnicou – nie je možné získať prístup k zariadeniam, ktoré sú otvorené vo Windows exclusive móde.

Vyzerá to tak, že jediné riešenie nášho problému je priamo pracovať s filter driverom, ktorý sa nachádza na driver stacku daného zariadenia.

3.1.3 Driver

Windows ponúka niekoľko vstavaných driverov pre rôzne typy zariadení. Stačí ak nájdeme ten, ktorý nám umožní komunikáciu s konkrétnymi HID zariadeniami, s ktorými bol problém kvôli exclusive módu – klávesnica a myš. Pre tieto dve konkrétne zariadenia momentálne existuje *Moufiltr* [28] (upper-level filter driver pre myš) a *Kbfiltr* [29] (upper-level filter driver pre klávesnicu). Tie ale podporujú len legacy zariadenia – non-USB, non-Bluetooth a non-I2C zariadenia.

Riešením by teda bolo naprogramovanie vlastného filter driveru pre myš a klávesnicu. Toto je ale už nad rámec našej práce a preto na získavanie USB paketov použijeme inú third-party aplikáciu.

3.1.4 Third-party aplikácie

Na zachytávanie USB paketov sme sa rozhodli použiť *USBPcap* [30], s ktorým sme sa už stretli v kapitole 1.2 keď sme opisovali Wireshark a rôzne sniffery, s ktorými je schopný spolupracovať. USBPcap je veľmi rozšírený Windows sniffer USB paketov a jeho hlavnou výhodou je jednoduchá inštalácia. Takisto nám vyhovuje, že je s ním schopný spolupracovať Wireshark, čo nám častokrát počas vývoja poskytlo spôsob akým sme si mohli overiť správnosť nami vykonanej analýzy niektorých paketov. Daný sniffer zachytáva pakety do súborov v obecne známom formáte – pcap [31].

Miernou nevýhodou je, že nepodporuje čítanie zo súboru do ktorého práve zapisuje – analýzu paketov v reálnom čase tak budeme musieť vykonať trochu iným spôsobom, a to za pomoci Wiresharku. Ako sme už spomínali, Wireshark je schopný spolupráce s USBPcapom a zároveň podporuje ukladať výstup analýzy do súboru, ktorý je takisto formátu pcap. Wireshark už ale podporuje čítanie z daného súboru počas toho, ako do neho zapisuje. To znamená, že ak budeme chcieť vykonať analýzu v reálnom čase, urobíme to pomocou Wiresharku (konkrétnejší postup si ukážeme neskôr v užívateľskej dokumentácii v kapitole 6).

3.2 Spracovávanie pcap súborov

moznosti ako citat pcap subory : bud pouzit uz existujucu kniznicu : na linuxe Libpcap, windows NPcap(deprecated WinPcap), alebo citat subory manualne : std::istream alebo QFile

3.3 Sémantická reprezentácia dát

ako si z dat vytiahnut udaje ktore su potom pouzite na semanticku analyzu implementovanych HID zariadeni : HID Report parser, InputValues a Endpoint-Device struct. Nasledne sparovanie - ako vybrat spravny report pre konkretny input

3.4 Voľba frameworku

obecne co by som od toho GUI priblizne chcel, potom opisat preco som si vybral prave Qt a v nasledujucich kapitolach opisat rozhodnutia uz v Qt dovod preco som si zvolil qt namiesto inych c++ GUI frameworkov(napriklad sfml)

3.5 Zobrazenie základných informácií

ako zobrazovat zakladne info o packete : pouzit QListWidget alebo QTable-Widget (pripadne nieco ine ako nejaky abstract viewmodel), narok na zakladne funkcionality : lahka rozsiritenlnost o dalsie "stlpceky" , moznost jednoduchej interakcie(doubleClick na polozku). Mat vsetky info na jednom okne / mat pop-up okna.

3.6 Zobrazenie sémantického významu dát

ako vyzobrazit semanticky vyznam roznych dat - descriptory, usb header, vyznam input dat roznych HID zariadeni

3.7 Hexdump

ako v qt urobit hexdump - do coho zobrazovat data(vytvorit si vlastny viewer dedeny od QAbstractScrollArea, pripadne niecoho ineho) vs najst nieco co uz v qt je a upravit to aby to sedelo poziadavkam. Vziat do uvahy bezne funkcie hexdumpu : selection mody(oznacit naraz hexa a im odpovedajuce printable), logicke oddelenie dat(napriklad farbami)

4. Vývojová dokumentácia

4.1 Architekrúra aplikácie

4.2 Jadro aplikácie

4.2.1 USB_Packet_Analyzer

riadi celkovy beh programu, reaguje na input od uzivatela

4.2.2 Item Manager

spracovanie samostatneho packetu a ulozenie dat o nom

4.2.3 DataViewer

trieda ktora ma na starosti vyskakovacie okno po dvojkliku a item a nasledne reaguje na input od uzivatela v okne

4.2.4 TreeItem

reprezentuje jednotlive nody v stromovej strukture ktora sa potom vyuziva na zobrazenie dat v QTreeView

4.3 Modely

4.3.1 AdditionaldataModel

model na spravovanie zvysnych dat(data ktore nie su sucastou hlavicky packetu)

4.3.2 ColorMapModel

vyobrazenie pomocnej mapy na lepsie sa zorientovanie v zvyraznemom hexdumpe

4.3.3 DataViewerModel

model na hexdump - prenasa hex/printable a zaroven o co vlastne ide(konkretny descriptor, interrupt data, ...)

4.3.4 TreeItemBaseModel

model na QTreeView ktorz vyuziva TreeItem

4.3.5 USBPcapHeaderModel

model na QTreeView ale specialne pre USBPcap hlavicku packetu

4.4 Interpretery

4.4.1 BaseInterpreter

abstractna trieda od ktorej dedia vsetkz interpretery

4.4.2 Interpreter factory

facory trieda na pridelenie konkretneho interpreteru za runtimu kvoli jednoduchosti na lepsie rozsirenie programu do buducnosti

4.4.3 Interpretery descriptorov

Config,Device,Setup,String,...

4.4.4 Interrupt transfer interpretery

obecne interrupt transfer interpreter - sluzi skor ako factory na rozne doteraz implementovane HID zariadenia

Joystick interpreter

Mouse interpreter

Keyboard interpreter

4.5 Delegáti

DataViewerDelegate

Qt delegat - stara sa o highlight hexdumpu

4.6 HID

4.6.1 HIDDevices

staticka trieda, drzi vsetky rozpoznane HID zariadenia a obsahuje funkcie specificke nich - parsovanie HID Report descriptoru

4.7 Práca so súbormi

4.7.1 FileReader

praca zo suborom a predavanie precitanych dat, offline/online capture, QFile vs std::istream

4.8 Globálne dáta

4.8.1 ConstDataHolder

staticka trieda na drzanie si konstant ktore su potrebne napriec celym programom. Mapovanie z enumu do jeho stringovej reprezentacie

4.8.2 PacketExternStructs

obsahuje definiciu vsetkych dolezitych USBPcap structov, pcap structov, enumov a vsetkych structov ktore pouzivam v aplikacii

5. Možnosti rozšírenia

Rozobrať čo všetko sa dá urobiť s tými dátami, ktoré už mám uložené v pamati, ale momentálne sa s nimi nič nedeje

5.1 Ukladanie výstupu do súboru

výstup analýzy do súboru(textového)

5.2 Iná vizuálna reprezentácia dát

Momentálne vyzobrazujem dáta prevažne v QTreeView alebo QTableView, ale vdaka tomu ako ich mám uložené + to že nad nimi operuje nejaký model ktorý vie vrátiť dáta na základe indexu, by nemuselo byť taká zložité pridať inú vizualizáciu dát(napríklad obrázkovú ako tu : https://www.usbmadesimple.co.uk/ums_5.htm)

5.3 Pridávanie nových interpreterov pre descriptory

pridanie nových druhov descriptorov - pridať nový interpreter do factory

5.4 Pridanie intepreteru na interrupt tranfser

pridanie analyzy interrupt transferu aj pre ine ako hid zariadenia

5.4.1 Pridanie nových HID zariední

nove HID zariadenie - pridanie do interrupt "factory"

5.5 Pridanie analýzy pre isochronous a bulk transfer

semanticka analyza aj inych ako interrupt alebo control transferov - momentalne su rozpoznavane len v hexdumpe

5.6 ?Možnosť rozšírenia na iné platformy?

uprava aplikacie aby bola prenositelna aj na ine platformy, co vsetko by tam bolo treba upravit(pravdepodobne nie vela, kedze qt je prenosne, a prakticky jedine co pouzivam spojene s windowsom su jeho structy na rozne descriptory)

6. Užívateľská dokumentácia

6.1 Inštalácia

nastavenie celkovej aplikácie, ale aj nainstalovanie USBPcap + wireshark a ich kombinácia pre live capture

6.2 Orientácia v GUI aplikácie

popis k jednotlivým tlačidlám gui

6.3 Používanie aplikácie

ako spustit live/offline capture, a celkovo ako pracovať s aplikáciou(popis funkcií - doubleClick na item => zobrazi sa pop-up okno s bližšou analýzou)

7. Záver

7.1 Zhrnutie

celkove zhrnutie prace, ?praca s Qt?

7.2 Budúce plány

Zoznam použitej literatúry

[1] PS/2 port. https://en.wikipedia.org/wiki/PS/2_port.

[2] Paralelný port. https://en.wikipedia.org/wiki/Parallel_port.

[3] RS-232 port. https://en.wikipedia.org/wiki/RS-232.

[4] Paralelná SCSI zbernica. https://en.wikipedia.org/wiki/Parallel_SCSI.

[5] USB 2.0 Specification. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, Figure 4-1].

[6] USB 2.0 Specification – definícia USB Host a s ním súvisiace pojmy . https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, kap. 4.1.1.1].

[7] USB 2.0 Specification – USB zariadenie definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, kap. 4.1.1.2].

[8] USB 2.0 Specification – USB Hub definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, str. 6].

[9] USB 2.0 Specification – USB Function definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, str. 6].

[10] USB 2.0 Specification – USB topológia zbernice. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, kap. 4.1.1].

[11] USB 2.0 Specification – USB descriptor definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, kap. 9.5].

[12] Zoznam definovaných USB tried. https://www.usb.org/defined-class-codes.

[13] USB 2.0 Specification – USB paket definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, str. 7].

[14] USB 2.0 Specification – USB typy prenosov definícia. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf, kap. 4.7].

[15] USB Client Driver. https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/usb-driver-development-guide. [sekcia „Where applicable“].

[16] USB Request Block. https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/communicating-with-a-usb-device.

[17] USB 2.0 Specification. https://www.usb.org/document-library/usb-20-specification. [súbor usb_20.pdf].

[18] USB Human Interface Device Class. https://www.usb.org/document-library/device-class-definition-hid-111. [str. 9].

[19] Genius myš použitá v úvode pri porovnaní existujúcich analyzátorov. https://us.geniusnet.com/product/dx-120/.

[20] Logitech myš požitá na ukážku konkrétneho inputu zariadenia. https://www.amazon.com/Logitech-Spectrum-Shifting-Personalized-Programmable/dp/B019OB663A.

[21] Fotka genius myši prevzatá z oficiálnej genius stránky. https://us.geniusnet.com/wp-content/uploads/sites/2/2020/01/DX-110_P18_980x600.jpg.

[22] Fotka logitech myši prevzatá zo stránky obchodu. https://images-na.ssl-images-amazon.com/images/I/31qPw4sF6uL._AC_.jpg.

[23] Komunikácia HID Clienta s HID Class driverom. https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/opening-hid-collections.

[24] HID Application Programming Interface (API). https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/introduction-to-hid-concepts.

[25] Tabuľka HID zariadení a ich Access Mode. https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/hid-architecture. [sekcia „HID Clients Supported in Windows“].

[26] libusb knižnica. https://libusb.info/.

[27] HIDAPI knižnica. https://github.com/signal11/hidapi.

[28] Moufiltr – filter driver. https://docs.microsoft.com/en-us/samples/microsoft/windows-driver-samples/mouse-input-wdf-filter-driver-moufiltr/.

[29] Kbfiltr – filter driver. https://docs.microsoft.com/en-us/samples/microsoft/windows-driver-samples/keyboard-input-wdf-filter-driver-kbfiltr/.

[30] USBPcap sniffer. https://desowin.org/usbpcap/.

[31] Pcap file format. https://wiki.wireshark.org/Development/LibpcapFileFormat.

Zoznam obrázkov

1.1 USB topológia. Obrázok prevzatý z USB 2.0 špecifikácie [5]. . . . 4
1.2 Ukážka myší, ktorých input budeme porovnávať 6
1.3 Ukážka hexdumpu so zvýrazneným inputom genius myši. 7
1.4 Ukážka hexdumpu so zvýrazneným inputom logitech myši. 7
1.5 Ukážka hexdumpu so zvýrazneným inputom genius myši s významom. 7
1.6 Ukážka hexdumpu so zvýrazneným inputom logitech myši s významom. 7
1.7 Ukážka hexdumpu vo Wiresharku. 8
1.8 Ukážka hexdumpu so zvýrazneným endpoint descriptorom. 8
1.9 Ukážka hexdumpu s farebným oddelením na základe významu. . . 9
1.10 Ukážka reprezentácie dát pomocou stromovej štruktúry. 9
1.11 Endpoint descriptor reprezentovaný dátami zvýraznenými na obrázku 1.7 vyššie. 9
1.12 Ukážka vyobrazenia jednotlivých bytov. 10
1.13 Ukážka kliknutia na položku v hexdumpe. 10
1.14 Ukážka kliknutia na položku *endpoint descriptoru* v stromovej štruktúre. 10
1.15 Príklad obecného vyobrazenia jednotlivých paketov vo Wiresharku. 10
1.16 Príklad inputu myši vo Wiresharku. 11
1.17 Ukážka stromovej štruktúry na zvolenie si zariadenia, s ktorým bude zachytávaná komunikácia. 11
1.18 Príklad hexdumpu v Device Monitoring Studio. 12
1.19 Príklad analýzy paketov. 12
1.20 Ukážka vyobrazenia URB. 12
1.21 Príklad inputu myši v Device Monitoring Studio. 12
1.22 Príklad obecného vyobrazenia jednotlivých paketov v Device Monitoring Studio. 13
1.23 Užívateľské rozhranie Device Monitoring Studio. 13

3.1 Tabuľka zariadenía ich *Access Mode*. Zariadenia postupne po riadkoch – myš, joystick a klávesnica 19

Zoznam tabuliek

Seznam použitých zkratek

Prílohy

.1 První příloha