Masarykova univerzita
Fakulta informatiky

Démon poskytujúci HTTP API pre prácu s databázou ABRT

BAKALÁRSKA PRÁCA

Matej Buday

Brno, 2013

Masarykova univerzita
Fakulta informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Datum: 28. 4. 2011

Student: Matej Buday

Program: FI B-IN Informatika

Obor: FI PSK Počítačové sítě a komunikace

Vedoucí práce: Mgr. Marek Grác

Název práce: Démon poskytujúci HTTP API pre prácu s databázou ABRT

Zadání: Študent preskúma existujúce knižnice na prácu s HTTP(s) a šifrovaním obsahu na strane serveru. Hlavným cieľom projektu je navrhnúť HTTP aplikačné rozhrania pre prácu s databázou problémov zaznamenaných systémom ABRT. Komunikáciu je nutné podporovať aj pomocou zabezpečených kanálov (napr. SSL). V praktickej časti práce študent naimplentuje vybranú množinu HTTP API.

Základní literatura:

Souhlas se zadáním (podpis, datum)

student(ka)	vedoucí bakalářské práce	vedoucí odpovědného pracoviště

*KTP
KPGD
KPSK
KIT
* Hodící se označte!

Prehlásenie

Prehlasujem, že táto práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovávaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Matej Buday

Vedúci práce: Mgr. Marek Grác

Poďakovanie

Ďakujem Marekovi Grácovi, Jiřímu Moskovčákovi a Alene Ganobčíkovej.

Zhrnutie

Obsahom tejto práce je návrh rozhrania dostupného prostredníctvom HTTP pre prácu s databázou problémov programu ABRT. Ďalej implementácia démona, ktorý podmnožinu tohto API poskytuje na zvolenom sieťovom rozhraní. API dokáže odpovedať vo viacerých formátoch, medzi ktoré patria XML a HTML. Démon podporuje autentifikáciu užívateľov prostredníctvom PAM a komunikáciu cez spojenie zabezpečené pomocou SSL.

Kľúčové slová

HTTP, API, démon, server, Linux, Xml, SSL

Obsah

1 Úvod....1
2 Automatic Bug Reporting Tool....2
2.1 Fungovanie ABRT....2
2.2 Naša práca v rámci projektu ABRT....3
3 HTTP....4
3.1 Čo je to HTTP....4
3.2 Výber knižnice....6
3.3 Zhodnotenie výberu....7
4 Autentifikácia užívateľa....8
4.1 PAM....8
4.2 Prepojenie s HTTP....8
5. Zabezpečenie komunikácie....9
5.1 Princíp fungovania SSL....9
5.2 Knižnice implementujúce SSL....9
5.3 Zhodnotenie výberu....10
6. API....11
6.1 Návrhové štýly aplikačných rozhraní....11
6.2 Návrh API pre abrt....12
7 Implementácia....15
7.1 Použité knižnice a nástroje....15
7.2 Štart démona....17
7.3 Prijímanie spojenia....17
7.4 HTTP dotaz v abrt-http....17
7.5 HTTP odpoveď....18
7.6 Autentifikácia....18
7.7 API v abrt-http....19
7.8 Zaslanie odpovede....19
7.9 Diagram funkcií....20
8 Záver....21
9 Literatúra....22
A Obsah priloženého CD....23
B Inštalácia....23
C Manuálová stránka....24

1 Úvod

ABRT (Automatic Bug Reporting Tool) je projekt, ktorého cieľom je uľahčiť reportovanie chýb v prostredí operačného systému založeného na jadre Linux. Jeho hlavnou súčasťou je Linuxový démon, ktorý sleduje pády programov, zaznamenáva ich a prostredníctvom zásuvných modulov (pluginov) poskytuje možnosti na ich odosielanie. Okrem samotného démona sa v projekte nachádzajú užívateľské aplikácie umožňujúce prácu s databázou problémov[i], reportovanie problémov a konfiguráciu pluginov. Tieto aplikácie však majú určité obmedzenia – sú určené iba pre lokálnych užívateľov a nie sú priamo dostupné po sieti.

Pri správe väčšieho počtu pracovných staníc či serverov vzniká potreba spravovať problémy jednoducho a efektívne. To zväčša znamená mať možnosť pracovať s nimi z jedného miesta. Toto sa dá dosiahnuť aj pomocou existujúcich nástrojov, avšak nejedná sa o systémové riešenie. Cieľom nasledujúcej práce bude navrhnúť a z časti implementovať rozhranie, ktoré umožní vzdialenú prácu s ABRT.

Hlavným zámerom je poskytnúť rozhranie, ktoré sa dá použiť v iných aplikáciách. Takto bude možné integrovať agregovanú správu vzdialených systémov do správcovských nástrojov. Okrem tohto rozhrania pre programovanie aplikácií[ii] bude možnosť pracovať s týmto rozhraním aj priamo, pomocou webového prehliadača.

Výsledný program, ďalej nazývaný abrt-http, bude fungovať ako Linuxový démon počúvajúci na zvolenom sieťovom rozhraní. Na komunikáciu bude využívať protokol HTTP a bude podporovať šifrovanie spojenia. Databáza problémov bude dostupná užívateľom podľa prístupových práv, s ktorými boli jednotlivé záznamy vytvorené. Na overenie identity užívateľa sa využijú možnosti, ktoré poskytujú HTTP a PAM.

---

i V našej práci sa bude pod pojmom problém myslieť záznam pádu aplikácie, ktorý vygeneroval ABRT démon.

ii Ďalej označované ako API – skratka z anglického Application Programming Interface – programovateľné aplikačné rozhranie.

2 Automatic Bug Reporting Tool

ABRT, pôvodne nazývaný CrashCatcher, je otvorený softvér vyvíjaný spoločnosťou Red Hat. Primárne je vývoj zameraný na systémy Fedora, Red Hat Enterprise Linux a ich deriváty (CentOS, Scientific Linux), avšak existujú aj pokusy s jeho použitím na iných systémoch založených na jadre Linux (napríklad OpenSUSE a Mandriva). Je licencovaný pomocou GNU General Public License (GPL) vo verzii 2. Podľa licenčných podmienok túto licenciu prevezme aj kód vytvorený v rámci tejto práce.

V tejto práci budeme používať ABRT verzie 2, pričom sa budeme držať aktuálnej vývojárskej verzie.

2.1 Fungovanie ABRT

Nevyhnutnou súčasťou projektu ABRT je takzvaný „hook“, ktorý sleduje nevhodné ukončenia aplikácií. Momentálne sú podporované nasledovné typy aplikácií.

2.1.2 Natívne aplikácie v C a C++

V prípade C a C++ programov (respektíve knižníc) ide o zaregistrovanie funkcie v jadre systému. Táto funkcia je volaná vždy, keď systém zaznamená pád aplikácie, a jadro jej predá základné informácie o danom páde. Zo všetkých možných ukončovacích signálov sa ABRT zaujíma o nasledovné: SIGILL, SIGFPE, SIGSEV, SIGBUS, SIGABRT a SIGTRAP. Ak je zaznamenaný jeden z týchto signálov, tak sa uloží obraz pamäte (ktorý taktiež vygenerovalo jadro systému) a spolu s ostatnými informáciami sa odovzdá démonu ABRT zavolaním funkcie.

2.1.3 Aplikácie v jazyku Python

Pre aplikácie v jazyku Python sa „hook“ zavádza do interpreta CPython, kde zachytáva výnimky, ktoré by sa inak dostali až k interpretu. Pri zaznamenaní výnimky sa vytvorí spojenie s démonom ABRT a prostredníctvom neho sa odovzdajú požadované informácie.

Démon ABRT, spúšťaný pri štarte systému programom abrtd, získané informácie následne analyzuje a vytvorí záznam v databáze problémov. Nie každý problém sa tam dostane, niektoré nie sú z rôznych dôvodov[i] pre ABRT zaujímavé.

Takéto oddelenie zachytávania problémov a ich následného spracovávania umožňuje pridávanie ďalších „hookov“ pre iné typy aplikácií.

i Môže sa jednať o bežné správanie aplikácie, prípadne nebolo získané dostatočné množstvo informácií.

2.2 Naša práca v rámci projektu ABRT

Program abrt-http bude v súčasnom stave projektu ABRT jeho nedeliteľnou súčasťou. Dôvodom je, že sa aj pre úplne základnú funkcionalitu spolieha na funkcie z knižnice abrtlib. Aj keď je databáza problémov uložená na disku a je nám voľne prístupná (so správnymi užívateľskými právami), tak by priama práca s ňou mohla priniesť mnoho implementačných aj funkčných problémov. Okrem toho je týmto prístupom program abrt-http oddelený od fungovania abrtd a prípadné zmeny, napríklad v spôsobe ukladania problémov, sa nás nedotknú (ak nedôjde aj k zmenám v používanej abrt knižnici).

Zdrojové kódy výslednej aplikácie budú súčasťou stromu ABRT a budú sa aj spolu prekladať, zostavovať a baliť.

3 HTTP

3.1 Čo je to HTTP

HTTP, z anglického Hypertext Transfer Protocol, je bezstavový sieťový protokol pracujúci na aplikačnej vrstve. Jeho presná špecifikácia a požiadavky na jeho implementáciu sa nachádzajú v dokumente RFC číslo 2616[4] a v ňom uvedených aktualizáciách. V tomto dokumente sa nachádzajú aj všetky nasledujúce informácie o jednotlivých častiach HTTP správ. Na ovládanie komunikácie sa v prípade dotazov používajú metódy (Request Methods), v prípade odpovedí kódy (Response Codes). Ako v dotazoch, tak aj v odpovediach sa používajú hlavičky (Header Fields).

3.1.1 Formát správ

Telá správ sú založené na nasledujúcich schémach. Na konci každého riadku sa nachádzajú znaky CR LF. Telo správy (Message Body) nie je povinné vo všetkých prípadoch.

Schéma dotazu:

```
Method Request-URI HTTP-Version
Request Header Fields

Message Body
```

Schéma odpovede:

```
HTTP-Version Status-Code Reason-Phrase
Response Header Fields

Message Body
```

3.1.2 Request Methods

Metóda žiadosti patrí medzi povinné súčasti HTTP dotazu. Celkovo ich existuje deväť, pre našu prácu sú relevantné nasledujúce:

- GET

  Metóda GET slúži na získanie zdroja (dokument, súbor) a nemala by ovplyvniť jeho stav. To znamená, že viac rovnakých GET dotazov by vždy malo vrátiť rovnakú odpoveď.
- HEAD

  Je zhodná s metódou GET, jediným rozdielom je, že server nemusí vracať telo odpovede.

- OPTIONS

  Odpoveď na túto metódu má obsahovať hlavičku so zoznamom povolených metód pre danú URI. S metódou OPTIONS je namiesto URI možné použiť znak „*“, ktorý špecifikuje, že sa nezaujímame o konkrétnu URI.
- DELETE

  Dotaz s touto metódou žiada o zmazanie uvedeného zdroja.

Ďalšími bežne používanými metódami sú POST a PUT, ku ktorým sa pripája aj telo správy. V našej implementácii však nebudú použité.

3.1.3 Response Codes

Kód uvedený v odpovedi, nazývaný tiež „Status Code“, určuje ako skončil dotaz, prípadne v akom stave sa nachádza požadovaný zdroj. Každý kód je asociovaný s textovou správou, ktorá je ako nezáväzná definovaná v špecifikácii HTTP. Kódy sa delia podľa významu do skupín rozdelených po stovkách. Kódy v rozmedzí 100-199 sú informačné správy, 200-299 oznamujú úspech, 300-399 značia presmerovanie[i], 400-499 sú odpoveďou na chybný dotaz a 500-599 informujú o chybe na strane servera. Zadefinovaná je iba časť zo všetkých kódov. Medzi hlavné, a pre našu prácu potrebné, patria:

- 200 OK

  Odpoveď na úspešný dotaz. V prípade použitia metódy GET bude telo správy obsahovať požadovaný zdroj.
- 400 Bad Request

  Dotaz mal chybnú syntax.
- 401 Unauthorized

  Klient nemá právo na získanie daného zdroja a musí sa autentizovať. Odpoveď s týmto kódom musí obsahovať hlavičku WWW-Authenticate.
- 404 Not Found

  Požadovaný zdroj sa nepodarilo nájsť. Z bezpečnostných dôvodov takto server môže odpovedať aj na dotaz na existujúci zdroj, na ktorý však vykonávateľ dotazu nemá právo.
- 405 Method Not Allowed

  Použitá metóda nie je dovolená nad daným zdrojom.
- 500 Internal Server Error

  Všeobecná chybová správa. Vhodná napríklad pri chybe súborového systému.
- 501 Not Implemented

  Požadovaný dotaz nie je možné uskutočniť z dôvodu, že server nepodporuje danú funkcionalitu.

---

i Pre úspešné dokončenie požiadavku musí klient vykonať dodatočný dotaz.

3.1.4 Header Fields

Hlavičky slúžia na dodatočné riadenie komunikácie. Niektoré z nich sú povinné, väčšina však nie. Niektoré sú určené iba pre dotazy, niektoré pre odpovede a niektoré môžu byť použité v oboch prípadoch. Špecifikácia dovoľuje definíciu a používanie vlastných HTTP hlavičiek. Hlavička má nasledujúci formát:

```
Meno: Hodnota
```

Na konci každého riadku je opäť dvojica znakov CR, LF. V prípade, že je prvý znak riadku „biely znak“[i], jedná sa o pokračovanie `hodnoty` predchádzajúcej hlavičky. Ak sa `Meno` opakuje, tak sa jedná o pokračovanie predchádzajúceho výskytu `Mena`. Na poradí jednotlivých hlavičiek nezáleží. Za poslednou hlavičkou sa nachádza prázdny riadok (CR, LF) a za ním nasleduje prípadné telo HTTP správy.

Medzi hlavičky využité v tejto práci patria:

- Content-Length
  Hodnota tejto hlavičky obsahuje číslo zodpovedajúce veľkosti tela správy v bajtoch. Používa sa pri dotazoch aj pri odpovediach v prípade, že správa obsahuje telo.
- Accept
  Požadovaný formát odpovede. Je možné uviesť viac podporovaných typov, pri ktorých môže byť uvedená priorita. V prípade zhodných priorít použitý formát vyberie server.
- Content-Type
  Formát odpovede. Mal by zodpovedať jednej z hodnôt uvedených v dotaze pod hlavičkou Accept.
- WWW-Authenticate
  Hodnota tejto hlavičky obsahuje požadovaný spôsob autentifikácie, napríklad Basic[ii]. Posiela sa v odpovedi s kódom 401.
- Connection
  Udáva typ spojenia, prípadne uzatvára spojenie.
- Authorization
  Obsahuje typ autentifikácie a údaje potrebné na jej vykonanie.
- Host
  Doménové meno servera. V HTTP/1.1 ide o povinnú hlavičku, ktorá má byť prítomná v každom dotaze. Server sa pomocou nej môže rozhodnúť ako odpovedať v prípade, že je na jednej adrese a porte dostupných viac domén.

3.2 Výber knižnice

Pri výbere knižnice na prácu s HTTP sme brali do úvahy viacero kritérií. Hľadali sme knižnicu, ktorá existuje ako balík v repozitároch distribúcie Fedora. Taktiež bolo potrebné, aby knižnica umožňovala prácu s jednotlivými hlavičkami HTTP dotazu aj odpovede. Po zhodnotení týchto kritérií sa ukázalo, že najvhodnejšie by bolo použiť knižnicu, ktorá poskytuje

---

i Whitespace – medzera alebo tabulátor.
ii Viď kapitola 3.2.

rozhranie na analýzu dotazu a tvorbu odpovede. Či už je to jej jediná funkcionalita, alebo je súčasťou väčšieho spektra vlastností. Z dostupných možností vzišli nasledujúci kandidáti.

3.2.1 LibHTTPD

Táto knižnica slúži na vytvorenie HTTP serveru v rámci aplikácie. Bola vyvinutá spoločnosťou Hughes Technologies a v čase tvorby tejto práce sa nachádzala vo verzii 1.4, ktorá bola vydaná v roku 2005. Implementuje iba podmnožinu špecifikácie HTTP. Dokáže poskytovať ako statický, tak aj dynamický obsah. Dynamický obsah je bez ďalšej medzi vrstvy možné generovať iba v jazyku C. Podporuje prácu s hlavičkami, má vstavané funkcie na spracovanie formulárov a má aj podporu pre prácu s cookies.

Podľa tvorcov sa jedná o malú knižnicu vhodnú na použitie vo vnorených systémoch (embedded systems) a malých aplikáciach. Je vydaná pod licenciou GNU GPL a za poplatok je možné získať aj vývojársku licenciu pre použitie v aplikáciach, v ktorých nie je možné použiť GPL verziu.

3.2.2 GNU libmicrohttpd

GNU libmicrohttpd je knižnica na tvorbu HTTP serverov priamo v aplikácii. Vyvíjaná je v rámci projektu GNU, licencovaná je pomocou GNU LGPL. V čase tvorby tejto práce bola prístupná vo verzii 0.9.23.

Je plne vyhovujúca HTTP špecifikácii, umožňuje počúvanie na viacerých portoch, má podporu pre šifrovanie, basic authentication, ipv6 a iné. Nemá žiadne ďalšie závislosti a nachádza sa v repozitároch Fedory.

3.3 Zhodnotenie výberu

Knižnica LibHTTPD bola zvažovaná aj napriek tomu, že sa nenachádza v repozitároch Fedory. Vlastnosťami totiž takmer ideálne vyhovovala našim požiadavkám. Fakt, že projekt je dlhšie neaktívny[i], viedol k rozhodnutiu nevyužiť túto knižnicu.

Druhá knižnica je aj napriek svojmu názvu príliš komplexná pre potreby abrt-http. Disponuje množstvom vlastností, ktoré nie sú potrebné a jej používanie by sa teda mohlo stať záťažou. Ďalším negatívom bolo naviazanie na jedinú SSL knižnicu.

Rozhodli sme sa teda nepoužiť ani jednu zo zvažovaných knižníc. Vzhľadom na potreby abrt-http je vlastná implementácia servera HTTP dostačujúca. Dočasnou nevýhodou je neúplná kompatibilita s HTTP. Pozitívom tohto rozhodnutia je nezavádzanie ďalších závislostí do projektu ABRT.

---

i Posledná vydaná verzia má viac ako päť rokov a archív užívateľskej komunikácie je zverejnený naposledy z roku 2002.

4 Autentifikácia užívateľa

Autentifikácia je proces overenia pravosti, prípadne identity. V prípade užívateľov sa vykonáva za účelom sprístupnenia služieb či informácií, ktoré bežne nie sú dostupné každému.

V operačných systémoch postavených na jadre Linux je každý užívateľ identifikovaný užívateľským menom. K tomuto menu býva priradená istá forma informácie, ktorou keď sa užívateľ úspešne preukáže, je považovaný za autentifikovaného. Existuje množstvo spôsobov, akými sa môže užívateľ preukázať, ale nás bude zaujímať heslo.

Heslo je reťazec znakov, ktorý nemusí byť unikátny, ale mal by byť známy iba užívateľovi. Heslá v systéme sa z bezpečnostných dôvodov neukladajú v pôvodnej forme, ale ukladá sa iba ich hash[i]. Heslo je pred hashovaním rozšírené o dodatočné znaky (salt), ktoré slúžia na ochranu pôvodného hesla v prípade prelomenia hashovacej funkcie. Na overenie hesla užívateľa použijeme PAM.

4.1 PAM

Názov je skratkou z anglického Pluggable Authentication Modules. Ide o nástroje na zjednodušenie procesu autentifikácie.[6] Pomocou PAM je možné vytvárať nové identifikačné procesy, napríklad identifikáciu užívateľa podľa spevu, a jednoducho ich zapájať do existujúceho systému. Ďalšou doménou PAM je poskytovanie API na autentifikáciu užívateľov pomocou dostupných identifikačných procesov. Práve túto druhú funkcionalitu využijeme.

Pri aktivácii PAM poskytneme adresu funkcie `PAM_conv`, ktorá má byť volaná pri autentifikácii. V tejto funkcii nám bude poskytnutá štruktúra `pam_response`[ii], do ktorej vyplníme heslo, zvolený PAM modul sa postará o overenie hesla a my budeme informovaní o úspechu, prípadne neúspechu.

4.2 Prepojenie s HTTP

HTTP hlavičky poskytujú viacero možností na prenos autentifikačných údajov. Vzhľadom na spôsob, akým ďalej overujeme získané údaje, potrebujeme meno aj heslo získať nezašifrované. Na toto slúži HTTP Basic Access Authentication, nazývaná tiež jednoducho basic autentifikácia.

Pri basic autentifikácii klient posiela v každom dotaze hlavičku Authorization. V tejto hlavičke je uvedené kľúčové slovo Basic a base64[iii] reťazec, ktorý obsahuje zakódované meno a heslo vo formáte `meno:heslo`. Tento reťazec prijímateľ rozkóduje, rozdelí podľa prvej dvojbodky a tak získa údaje potrebné na autentifikáciu.

---

i Neunikátny reťazec znakov vytvorený z hesla pomocou jednosmerného algoritmu.

ii Podrobnejšie informácie o tejto štruktúre sa dajú nájsť v manuálovej stránke `pam_conv(3)`.

iii Base64 je formát, v ktorom sú dáta zakódované pomocou ASCII znakov. Nejde o funkciu na zabezpečenie dát.

5 Zabezpečenie komunikácie

Z dôvodu prenosu citlivých údajov v HTTP hlavičkách je potrebné poskytnúť užívateľom možnosť zabezpečenia dátového prenosu. To je obzvlášť dôležité, ak sa nedá spoľahnúť na bezpečnosť nižších sieťových vrstiev. Na zabezpečenie komunikácie sme vybrali protokol SSL (respektíve TLS[i]), ktorý zabezpečuje ochranu a integritu prenášaných dát. Protokol bol pôvodne špecifikovaný firmou Netscape. Nové verzie sú definované v RFC, aktuálne RFC číslo 5246[5]. V tomto dokumente sú popísané všetky technické náležitosti reprodukované v nasledujúcej kapitole.

5.1 Princíp fungovania SSL

Prvým krokom je, že klient inicializuje spojenie na server podporujúci SSL a pošle zoznam podporovaných šifrovacích algoritmov. Z tohto zoznamu si server vyberie také, ktoré podporuje aj on a z nich vyberie ten najsilnejší. Informuje klienta o šifrách pre nadchádzajúce spojenie a pošle svoj certifikát. Klient sa rozhodne či certifikát prijme – rozhodnutie mu môže uľahčiť možnosť overiť si certifikát u certifikačnej autority. Po rozhodnutí pokračovať v spojení si klient vygeneruje náhodné číslo. Toto číslo následne zašifruje použitím verejného kľúča servera. Server zašifrovanú správu prijme a pomocou svojho privátneho kľúča ju dešifruje. Teraz majú obe strany k dispozícii číslo, na základe ktorého môžu vygenerovať kľúče potrebné pre komunikáciu.

5.2 Knižnice implementujúce SSL

Potrebujeme serverovú implementáciu SSL, ktorá sa dá použiť v jazyku C. Opäť máme podmienku, že musí existovať v repozitároch distribúcie Fedora. Ideálne by bolo, aby nemala žiadne ďalšie závislosti. Je tiež potrebné aby implementovala novšie verzie protokolov SSL, konkrétne SSLv3.

5.2.1 OpenSSL[8]

Knižnica podporuje SSL verzie 1 a 2 ako aj TLS verzie 1. Jedná sa o pokračovanie projektu SSLeay. Licencovaná je pod vlastnou open source licenciou, ktorá je vhodná aj pre komerčné použitie.

---

i TLS je následníkom protokolu SSL. Avšak TLSv1.x býva tiež označované ako SSLv3.x, teda v nasledujúcom texte sa pod pojmom SSL budú myslieť oba tieto protokoly.

5.2.1 GnuTLS[7]

Knižnica je vyvíjaná v rámci projektu GNU. Nepodporuje SSL verzie 2. Licencovaná je pomocou GNU LGPLv2.1+.

5.2.3 NSS[9]

NSS, skratka z anglického Network Security Services, je sada knižníc, ktorá bola pôvodne súčasťou produktov firmy Netscape (Navigator, Communicator) a neskôr sa stala samostatne dostupnou. Okrem Netscape sa na vývoji podieľali aj iné veľké firmy ako Google či Sun Microsystems. V súčasnosti je používaná v rade produktov, napríklad Mozilla Firefox, Evolution, Red Hat Directory Server a Sun Java System Application Server. Niektoré moduly sú certifikované pre použitie vo vládnych organizáciách v Spojených Štátoch Amerických (certifikát FIPS 140).

Podporuje množstvo bezpečnostných štandardov vrátane SSL verzie 2 aj 3. Je dostupná pod tromi rôznymi licenciami, z ktorých si užívateľ môže vybrať. Sú nimi Mozilla Public License (MPL), GNU GPL a GNU LGPL.

5.3 Zhodnotenie výberu

Všetky tri knižnice spĺňajú vopred stanovené požiadavky a z pohľadu tejto práce nemá ani jedna navrch. Vybrali sme knižnicu OpenSSL, avšak pri jej použití sme dbali na to, aby program abrt-http nebol na ňu príliš naviazaný a dala sa v prípade potreby (napríklad bezpečnostné alebo licenčné problémy) jednoducho vymeniť za inú.

6 API

API je skratkou pre Application Programming Interface. Vo všeobecnosti je to rozhranie slúžiace na komunikáciu aplikácií[i] v tom zmysle, že jedna aplikácia môže pristupovať k dátam druhej aplikácie a používať jej funkcie. V súvislosti s HTTP sa pod API rozumie sada pravidiel definujúcich spôsob komunikácie za účelom dosiahnutia určitého výsledku. API definované nad HTTP sa nazýva aj webové API.

6.1 Návrhové štýly aplikačných rozhraní

Existuje viacero architektonických štýlov, ktoré sa dodržiavajú pri návrhu webovej API. Medzi najčastejšie používané používané patria SOAP (Simple Object Access Protocol), RPC (Remote Procedure Call) a REST (Representational State Transfer).

- **SOAP**

SOAP nie je len návrhový štýl, jedná sa o protokol definovaný konzorciom W3C. Nie je viazaný na určitý komunikačný protokol, často sa však používa práve v spojení s HTTP. Na komunikáciu používa správy formátované v XML.

- **RPC**

RPC znamená vzdialené volanie procedúry. V kontexte webových API to často znamená volanie vzdialených metód. Jednotlivé funkcie servera sú naviazané na konkrétne povolené metódy v API.

- **REST**

REST je architektonický štýl, ktorý sa používa pri návrhu aplikačných rozhraní využívajúcich HTTP. Vo svojej dizertačnej práci[3] ho v roku 2000 definoval Roy Fielding, jeden z autorov HTTP špecifikácie. Naplno využíva dostupné možnosti protokolu HTTP (URI, metódy, návratové kódy) a v kontraste s vyššie uvedenými návrhmi sa tak často zaobíde bez tela správy v dotazoch.

6.1.1 Výber vhodného návrhového štýlu

V tejto práci sme sa ako základ rozhodli použiť práve REST. Treba však pripomenúť, že úplne čistý štýl návrhu môže niekedy viesť k nie úplne praktickým riešeniam. Preto sme sa rozhodli preferovať praktickosť. To znamená, že sa pri návrhu môžeme odkloniť od pravidiel, ktoré súvisia s REST.

---

i Pod pojmom aplikácia sa v tomto kontexte skrývajú aj iné významy. Napríklad knižnice.

6.2 Návrh API pre abrt

V texte budú ukážky odpovedí uvedené v XML. V reále však reprezentácia závisí od podporovaných formátov. Pre skrátenie zápisu nebudeme uvádzať hlavičku XML dokumentu.

6.2.1 Vstupný bod API

RESTful API by mala mať práve jeden vstupný bod. Znalosť URI tohto bodu by mala byť dostačujúca na využitie celej API. K ďalším častiam nás navedú odkazy. V našom prípade chceme, aby vstupný bod slúžil ako rozhranie medzi navzájom nezávislými časťami aplikačného rozhrania. Jednotlivými časťami sú databáza problémov, konfigurácia abrt a správa pluginov. Odpoveď na GET dotaz na vstupný bod by teda mohla vyzerať takto:

```
<api name="abrt-http" version="0.1">
  <link rel="problems" href="/problems/"/>
  <link rel="configure" href="/configure/"/>
  <link rel="plugins" href="/plugins/"/>
</api>
```

Už aj na prístup k tomuto bodu API je potrebné sa autentifikovať. Je to z dôvodu, aby náhodným dotazom nebolo možné zistiť aká služba je poskytovaná na danej adrese/porte. Metódy POST, PUT či DELETE nad týmto bodom nie sú povolené, nedávali by totiž zmysel.

6.2.2 Zoznam problémov

Táto časť API sa nachádza na URI /problems/ a má reprezentovať databázu problémov vytvorenú programom ABRT. Databáza je uložená v súborovom systéme disku a každý problém má svoj vlastný adresár, ktorý zodpovedá identifikátoru (id) problému. V danom adresári sa nachádzajú súbory s jednotlivými detailami o danom probléme. Medzi bežne sa vyskytujúce detaily patrí chybová hláška, obsah pamäte, čas pádu a iné.

ABRT monitoruje viacero druhov pádov a je možné, že v budúcnosti pribudnú ďalšie. Rôzne typy pádov majú svoje špecifiká a nie každý detail sa vyskytuje pri každom probléme. Detailov tiež môže byť rôzne množstvo a niektoré detaily sú pomerne rozsiahle. Na toto treba myslieť pri návrhu odpovede na túto časť API. Po komunikácii s vývojárom ABRT sme zistili, že jediné detaily, ktoré sa nachádzajú pri každom probléme, sú „reason“ a „time“. Reason popisuje dôvod pádu a time obsahuje unixový čas pádu. Toto budú jediné dve veci (okrem id), ktoré budeme zobrazovať pri výpise všetkých problémov.

Okrem tohto treba myslieť na to, že API bude používať viac ako jeden užívateľ. Každý užívateľ má však vidieť iba svoje problémy, prípadne superužívateľ (root) môže vidieť všetky. Odpoveď na GET request na /problems/ by teda mohla vyzerať takto:

```
<problems>
  <problem id="id123" href="/problems/id123">
    <item name="time" type="unixtime">1303846731</item>
    <item name="reason">Process /usr/lib/x was \
     killed by signal 11 (SIGSEGV)</item>
  </problem>
      ...
</problems>
```

Vidíme čas a dôvod pádu. Tri bodky naznačujú, že problémov môže byť vylistovaných viac. Metódy PUT, POST a DELETE opäť nie sú dovolené. Teoreticky by mohla byť povolená metóda DELETE na jednorázové zmazanie všetkých problémov, ale z pohľadu užívateľa API nie je problém zmazať problémy jednotlivo. Každý záznam o probléme obsahuje odkaz s id daného problému. Tento odkaz smeruje na detaily problému – problem details.

6.2.3 Detaily problémov

Ako už bolo spomenuté, každý problém môže obsahovať množstvo detailov. Tieto detaily budú užívateľovi dostupné po odoslaní GET dotazu na URI uvedenú v zozname problémov. Medzi detailami sa niekedy nachádzajú aj iné ako textové súbory – napríklad obsah pamäte, ktorou disponoval problémový program pri páde. Takéto informácie sa nedajú priamo zobraziť v textovej podobe a tiež nie sú dôležité pre užívateľa. Preto sa nebudú vypisovať spolu s ostatnými detailami, ale bude na ne iba ďalší odkaz. Po zavolaní GET na tento odkaz sa pošle priamo daný súbor už bez XML.

Odpoveď na GET /problems/id123 by mohla vyzerať takto:

```
<problem id="id123">
  <item name="uid" type="txt">500</item>
  <item name="time" type="unixtime">1303846731</item>
  <item name="executable" type="txt">/usr/lib/x</item>
  <item name="uuid" type="txt">a9ff2b338c453b0353</item>
<item name="coredump" type="bin"
     href="/problems/id123/coredump" rel="download"/>
<action name="delete" method="DELETE"
     href="/problems/id123"/>
</problem>
```

Vidíme, že položka typu „bin“ obsahuje odkaz na súbor, v tomto prípade coredump. Ďalej sa vo výpise nachádza položka „action“, ktorá má uvedenú metódu DELETE. Po prijatí dotazu s metódou DELETE na danú URI (zhodná s URI problému) bude záznam o probléme

odstránený.

Na reportovanie problému bude slúžiť URI /problems/id123/*plugin*, kde *plugin* je meno použitého pluginu. Jednotlivé programy môžu mať svoje vlastné reportovacie pluginy, ktoré doplnia špecifické informácie vzťahujúce sa k danému programu (napríklad konfiguračné súbory a pod.). Táto vlastnosť nebude implementovaná v rámci tejto práce.

6.2.4 Konfigurácia

Táto časť API, nachádzajúca sa na URI /configure/, má slúžiť na konfiguráciu samotného ABRT. Odpoveďou na GET teda bude konfiguračný súbor ABRT prevedený do požadovaného formátu. Zmena nastavení bude prebiehať tak, že sa celý súbor pošle v dotaze s metódou PUT. Ďalšou možnosťou by bolo posielať jednotlivé konfiguračné položky zvlášť, avšak takýto spôsob je zložitý z pohľadu klienta, nakoľko si musí kontrolovať stav aktuálneho konfiguračného súboru. Naviac by sa museli riešiť duplicitné položky.

Na strane serveru by pred aplikáciou nastavení mala prebehnúť minimálne syntaktická kontrola konfigurácie pred tým, než sa aplikuje.

Je otázne či je možnosť konfigurácie démona naozaj nutnou súčasťou API.

6.2.5 Zásuvné moduly – plugins

Po zaslaní GET dotazu na tento bod API sa vráti zoznam dostupných pluginov. Zoznam bude obsahovať iba mená a popisy jednotlivých pluginov. Každý plugin bude mať unikátne meno, ktoré bude použité v URI na získanie jeho konfigurácie. Spôsob konfigurácie bude prebiehať rovnako ako bol popísaný pri konfigurácii ABRT. Toto unikátne meno pluginu bude rovnaké ako to použité pri tvorbe URI na reportovanie problémov.

6.2.6 Ďalšie funkcie

V budúcnosti by mohlo byť navrhnuté rozhranie na podrobné vyhľadávanie problémov v databáze, ktoré by sa dalo implementovať pomocou parametrov URI[i]. Pri väčšom počte problémov v databáze, prípadne pri agregovanej správe viacerých staníc, by takáto funkcionalita bola nevyhnutná. Momentálne sa vyhľadávanie a filtrovanie dá dosiahnuť na strane klienta.

i Časť URI nachádzajúca sa za otáznikom.

7 Implementácia

V rámci tejto práce sme implementovali jadro programu abrt-http a podmnožinu z API definovanej v kapitole 5. Presnejšie vstupný bod API, zoznam problémov, zobrazovanie detailov jednotlivých problémov a ich mazanie.

Program je logicky rozdelený do viacerých zdrojových súborov. Súbor `abrtapi.c` obsahuje funkciu `main()`, funckie na inicializáciu siete, pomocné funkcie a funkciu `serve()`, ktorá je volaná na každé prichádzajúce spojenie. Súbory `abrtapi_http.c`, `abrtapi_api.c` a `abrtapi_pam.c` obsahujú časti programu zodpovedajúce časti názvu uvedeného za znakom „_“. Každý zdrojový súbor má príslušný hlavičkový súbor s rovnakým názvom a príponou .h.

7.1 Použité knižnice a nástroje

Pri vývoji sme použili nasledovné knižnice a aplikácie.

7.1.1 Knižnica GLib

GLib je rozsiahla knižnica, ktorá vznikla v rámci projektu GTK+ a neskôr sa osamostatnila. Je vyvíjaná pod hlavičkou GNOME Foundation. Poskytuje rozsiahlu škálu funkcií, no v našom projekte sme využili hlavne pokročilé dátové štruktúry známe hlavne z vyšších programovacích jazykov.

- gchar – Zodpovedá štandardnému typu char z jazyka C, avšak so sebou prináša vlastnú sadu pomocných funkcií. Tieto prinášajú zvýšenú bezpečnosť a taktiež novú funkcionalitu.
- GString – Ekvivalent typu string z niektorých vyšších jazykov. Je to vlastne obal nad gchar, ktorý sa stará o správu pamäte a opäť tak zjednodušuje prácu s textovými reťazcami.
- GList – Ide o obojstranne zreťazený zoznam. Podporuje odoberanie a vkladanie záznamov aj ich vyhľadávanie.
- GhashTable – Hashovacia tabuľka, niekedy tiež nazývaná slovník, umožňuje ukladanie dát pod zvoleným kľúčom. K takto uloženým dátam sa dá potom veľmi jednoducho a rýchlo pristupovať.

Knižnica GLib bola použitá už v samotnom projekte ABRT a teda nepriniesla žiadnu novú závislosť.

7.1.2 Knižnica na prácu s XML

Na prácu s XML sme použili knižnicu libxml2. Obsahuje všetko potrebné na vytváranie XML stromu. Okrem toho podporuje aj jeho čítanie a teda môže byť v budúcnosti využitá aj na spracovávanie požiadavkov vo formáte XML.

7.1.3 Detekcia úniku pamäte pomocou programu Valgrind

Pri tvorbe HTTP servera je potrebné spracovávať veľké množstvo dopredu neznámych dát. Nie je teda iná možnosť než používať dynamické dátové štruktúry a dynamickú alokáciu pamäte. Aj napriek zjednodušeniu, aké poskytujú knižnice ako Glib, sa občas môže stať, že dôjde k takzvanému úniku pamäte. To znamená, že si od operačného systému vyžiadame pamäť, ktorej sa neskôr nevzdáme aj napriek tomu, že už ju nevyužívame. Môže sa nám stať, že ju iba zabudneme vrátiť alebo ju úplne stratíme.

Takéto programátorské chyby sa v praxi môžu ťažko detekovať. Aplikácia môže fungovať bezchybne po dlhý čas kým sa chyba navonok prejaví. Preto existuje viacero nástrojov, ktoré pomáhajú takéto chyby detekovať.

My sme pri vývoji použili program Valgrind. Ten pri simulovanom behu aplikácie zaznamenáva každú vyžiadanú a vrátenú pamäť. Dokáže tak detekovať únik pamäte a určiť kde vznikol.

7.1.4 Verzovací systém git

Projekt ABRT pri vývoji používa verzovací systém git. Jedná sa o distribuovaný verzovací systém, to znamená, že každý užívateľ má svoju vlastnú kópiu celého repozitáru s kódom aj históriou zmien. Esenciálna vlastnosť takéhoto typu verzovacieho systému je takzvané spájanie (merge), ktoré sa vykonáva pri každom získaní zmien z centrálneho repozitára (pull).

Vývoj na abrt-http prebiehal tak, že sme si vytvorili klon centrálneho repozitára projektu ABRT[i] a v ňom následne prebiehal vývoj. Naše zmeny sme ukladali do iného, vzdialeného repozitára, konkrétne na službu github[ii]. Priebežne sme však získavali aj zmeny z vývojovej vetvy ABRT, aby sme za projektom nezaostali a dokázali tak včas reagovať na prípadné zmeny v častiach projektu, ktoré sa nás týkali.

---

i URL na git repozitár projektu ABRT git://git.fedorahosted.org/git/abrt.git

ii URL na webové rozhranie služby github s našou verziou stromu ABRT https://github.com/rplnt/abrt/

7.2 Štart démona

Program abrt-http prijíma parametre pri spustení z terminálu[i]. Tieto parametre sú po štarte skontrolované ako prvé. Následne sa pomocou funkcií, ktoré poskytuje knižnica ABRT, načítava konfiguračný súbor[ii]. V tomto konfiguračnom súbore sa nachádza adresa, na ktorej má démon prijímať spojenia; predvolený formát odpovede; možnosť vypnúť SSL a cesty k certifikátu a kľúču, ktoré sú potrebné pre funkčnosť SSL. Parametre zadané pri spustení majú z praktických dôvodov prioritu nad týmto konfiguračným súborom.

Po úspešnom načítaní konfigurácie prebehne inicializácia sieťového spojenia a prípadné nastavenie SSL. Po tomto kroku sa z programu stáva démon (pokiaľ nie je zapnutý debug mód) a začína počúvať na zvolenom sieťovom rozhraní.

7.3 Prijímanie spojenia

V prípade prichádzajúceho spojenia démon okamžite zavolá systémovú funkciu `fork()`. Pomocou nej sa vytvorí identická kópia procesu, ktorá bude spracovávať prijaté spojenie aby sa hlavný démon nezablokoval. O prvotné spracovanie sa stará funkcia `serve()`, ktorá načíta celý dotaz do zásobníku. Po načítaní prázdneho riadku, ktorý značí koniec hlavičiek, sa prechádza k validácii a spracovaniu HTTP dotazu.

7.4 HTTP dotaz v abrt-http

Prvotná kontrola dotazu sa vykonáva vo funkcii `parse_head()`, ktorá okrem syntaktickej kontroly načíta telo dotazu do štruktúry `http_req`, ktorá bude používaná počas celého ďalšieho spracovania dotazu. Štruktúra `http_req` vyzerá nasledovne:

```
struct http_req {
    http_method method;
    gchar *uri;
    gchar *version;
    GHashTable *header_options;
    void *body;
};
```

Medzi významné veci uložené v tejto štruktúre patrí metóda použitá pri dotaze, požadovaná URI a prípadné hlavičky dotazu. V aktuálnej verzii abrt-http sa po spracovaní hlavičky dotazu preruší čítanie zo spojenia[iii] a zavolá sa funkcia `generate_response()`.

---

i Viď prílohu C.
ii Vzorová konfigurácia je priložená ku zdrojovým súborom.
iii Implementácia nepodporuje žiadnu funkciu, ktorá by využila telo správy.

7.5 HTTP odpoveď

HTTP odpoveď má tiež svoju vlastnú štruktúru. Vyzerá nasledovne:

```
struct http_resp {
    int code;
    gchar *response_line;
    GString *head;
    gchar *body;
    int fd;
    content_type format;
};
```

Obsahuje kód odpovede, reťazec zodpovedajúci prvému riadku odpovede[i], všetky hlavičky v jednom reťazci, prípadne telo správy alebo otvorený súbor a formát odpovede. Na vyplnenie prvých troch položiek existujú nasledujúce funkcie.

- http_response(struct http_resp* *resp*, short *code*)

Táto funkcia ako parametre prijíma odkaz na štruktúru s odpoveďou *resp* a kód odpovede *code*. Vyplní `code` a `response_line` v štruktúre na ktorú odkazuje *resp*. Používa sa na „úspešné“ kódy, kedy sa telo odpovede vypĺňa na inom mieste programu.

- http_error(struct http_resp* *resp*, short *error*)

Ako parametre opäť prijíma odkaz na štruktútu odpovede a chybový kód *error*. Podľa zadaného kódu vyplní `code` a `response_line`. Taktiež vypĺňa telo správy v závislosti od požadovaného formátu. V prípade niektorých chýb vyplní aj potrebné hlavičky.

- http_add_header(struct http_resp* *response*, const gchar* *header_line*, ...)

Funkcia pridá zadanú hlavičku *header_line* do `head` štruktúry, na ktorú odkazuje *response*.

7.6 Autentifikácia

Vo funkcii `generate_response()` nasleduje autentifikácia užívateľa. Z hlavičiek sa extrahuje meno a heslo, ktoré sa následne overuje pomocou PAM. Meno nesmie obsahovať znak „:“. V prípade, že sa užívateľa podarí úspešne autentifikovať, aplikácia sa vzdá superužívateľských práv a prepne sa na práva užívateľa, ktorého sa podarilo autentifikovať. Toto sa robí z dvoch hlavných dôvodov. Kvôli bezpečnosti a jednoduchému filtrovaniu problémov pre daného užívateľa. Keďže sa problémy nachádzajú v súborovom systéme, tak sa o práva prístupu postará operačný systém.

i HTTP verzia, kód a textová verzia kódu oddelené medzerami.

V prípade, že je autentifikácia neúspešná, alebo vôbec neprebehla (chýbali potrebné hlavičky), odpovedá sa chybou a spojenie sa ukončí.

7.7 API v abrt-http

Po úspešnej autentifikácii užívateľa nasleduje presmerovanie dotazu podľa zvolenej metódy a požadovanej URI. V prípade neznámej URI sa vracia odpoveď 404. Ak je žiadaná URI súčasťou API, tak sa zavolá príslušná funkcia.

- api_entry_point(struct http_resp* *response*)

Táto funkcia vyplní telo odpovede vstupným bodom API podľa požadovaného formátu.

- api_problems(const struct http_req* *request*, struct http_resp* *response*)

Funkcia je volaná na všetky validné dotazy, ktorých URI začína reťazcom `/problems/`. V závislosti od zbytku URI sa v prípadne korektného dotazu vykoná jedna zo štyroch možností.

1. Do tela odpovede sa vyplní list všetkých problémov, na ktoré má užívateľ právo. V prípade, že neexistujú žiadne problémy, ktoré by sa mohli vypísať vráti sa prázdna odpoveď.
2. Do tela odpovede sa vložia detaily zvoleného problému.
3. V prípade použitia metódy DELETE na konkrétny problém sa daný problém odstráni z databázy.
4. V prípade požiadavku na súbor z databázy sa tento súbor otvorí a po dávkach posiela užívateľovi.

- api_serve_static(const struct http_req* *req*, struct http_resp* *resp*)

Táto funkcia sa používa na poskytovanie súborov, ktoré naozaj existujú v súborovom systéme. Je však možné poskytovať súbory iba z dopredu definovaných adresárov. Momentálne to je iba pod-adresár v domovskej zložke. Zatiaľ sa využíva iba v spojení s formátom HTML, keď je potrebné poslať CSS súbor a javascript umožňujúci mazanie problémov priamo z webového prehliadača.

7.8 Zaslanie odpovede

Posledným krokom po prichystaní štruktúry odpovede je zaslanie odpovede do spojenia, z ktorého prišiel dotaz. Posiela sa hlavička HTTP správy a následne telo, alebo požadovaný súbor. Po tom, čo je odpoveď odoslaná sa vyprázdni pamäť a spojenie sa uzavrie.

8 Záver

V tejto práci sa podarilo navrhnúť aplikačné rozhranie na spravovanie databázy problémov programu ABRT, na správu samotného ABRT a jeho pluginov. Návrh rozhrania z veľkej časti spĺňa princípy návrhu takzvaného REST API. Rozhranie je samodokumentačné a jednoduché na používanie.

V rámci praktickej časti bola implementovaná aplikácia poskytujúca vybranú podmnožinu navrhnutej API. Aplikácia implementuje jednoduchý server, ktorý podporuje IPv4, IPv6 aj lokálne spojenia cez Unixové sockety. Pri všetkých typoch spojenia podporuje šifrovanie, ktoré zabezpečí ochranu užívateľských údajov. Server nie je blokujúci, teda je možné vykonávať množstvo paralelných dotazov.

Ďalej bola implementovaná základná množina protokolu HTTP schopná prijímať, validovať a spracovávať dotazy a tak isto generovať validné odpovede. Nie všetky nutné súčasti HTTP boli implementované, ale pri použití s ABRT API by nemal nastať žiaden nežiadúci či chybový stav.

Samotný program podporuje komunikáciu vo viacerých formátoch. Okrem XML je to napríklad aj HTML, ku ktorému bolo dotvorené užívateľsky prívetivejšie rozhranie a možnosť vlastného modifikovania aplikácie pomocou užívateľských skriptov.

Pri písaní kódu bolo myslené na to, že aplikáciu môže ďalej vyvíjať niekto iný. S týmto súviselo používanie zmysluplných anglických názvov pre funkcie a premenné, vytváranie dokumentácie a v neposlednom rade pomerne časté komentáre popisujúce jednotlivé časti zdrojového kódu.

9 Literatúra

1. *ALLAMARAJU, Subbu. RESTful Web Services Cookbook. 320s. 2010.*

2. FIELDING, Roy. *REST APIs must be hypertext-driven.* <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. 2008.

3. FIELDING, Roy. *Representational State Transfer (REST).* <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. 2001.

4. FIELDING, Roy et al. *Hypertext Transfer Protocol – HTTP/1.1.* <http://tools.ietf.org/html/rfc2616>. 1999.

5. DIERKS & RESCORLA. *The Transport Layer Security (TLS) Protocol.* <http://tools.ietf.org/html/rfc5246>. 2008.

6. *Pluggable Authentication Modules* [online]. <http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules>. 2011

7. The GnuTLS transport layer security library [online]. <http://www.gnutls.org/>. 2013.

8. OpenSSL Project [online]. <http://www.openssl.org/>. 2013.

9. Network Security Services (NSS) [online]. <http://www.mozilla.org/projects/security/pki/nss/>. 2013.

A Obsah priloženého CD

Na priloženom CD sa nachádza celý projekt ABRT tak ako bol dostupný na stránkach projektu v čase písania aplikácie abrt-http, zabalený v dvoch rôznych archívoch. V adresári src/abrt-http sa nachádzajú zdrojové súbory vypracované v rámci tejto práce. Okrem zdrojových súborov sa tam nachádza súbor s pravidlami pre PAM, konfiguračný súbor pre abrt-http, vzorový kaskádový štýl a manuálová stránka.

Jedná sa o verziu vo vývoji s nie úplne ideálnym užívateľským rozhraním. Konzultujte prílohu C.

B Inštalácia

Najprv je potrebné zaobstarať si zdrojové súbory. Nachádzajú sa buď na priloženom CD, v archíve záverečných prác Masarykovej Univerzity alebo na url https://github.com/rplnt/abrt. Program abrt-http nie je samostatne použiteľný. Treba teda skompilovať a nainštalovať celý projekt. V koreňovom adresári projektu treba spustiť nasledujúce príkazy:

```
autogen.sh
configure
make rpm
```

V prípade chybových hlášok treba nainštalovať chýbajúce súčasti, ktoré sú potrebné pre kompiláciu. Po úspešnom skončení príkazu make rpm sa v adresári x86_64 (v prípade 64-bit systému) budú nachádzať rpm balíčky jednotlivých súčastí abrt. Tieto treba nainštalovať štandardným spôsobom.

Ďalej je potrebné z adresáru `src/abrt-http/pam` skopírovať súbor `abrt-http` do adresáru `/etc/pam.d/` a z adresáru `src/abrt-http/conf` skopírovať súbor `abrt-http.conf` do adresáru `/etc/abrt/`. V prípade, že chceme používať abrt-http aj prostredníctvom webového prehliadača, je vhodné skopírovať adresár `src/abrt-http/static` do adresáru `~/.abrt/`.

Teraz je možné spustiť démona pomocou príkazu `abrt-http`.

C Manuálová stránka

```
abrt-http(1)
============

NAME
----
abrt-http - Daemon providing API to control abrt over HTTP.


SYNOPSIS
--------
'abrt-http' [option]

DESCRIPTION
-----------
The program listens on specified address and port or on unix socket.
Incoming connections must authentize with system username
and password using HTTP Basic Authentication. Authentized users
can list problems, view detailed information of problems, download
memory dumps and delete problems. Responses are returned in format
specified in HTTP headers. Supported formats at the moment are
XML, HTML and plain text.

OPTIONS
-------
The options can be given in any order.
-v
    Be more verbose.
-x FILE
    Config file to use instead of the default one.
-a ADDRESS:PORT/PATH
    Start server on address:port or on unix socket on given path.
    Address can be IPv4 or IPv6. Connection won't be secure.
-e ADDRESS:PORT/PATH
    Same as above but use SSL.
-d
    Debug mode. Don't daemonize.
```

```
-?
    Print usage and exit.

FILES
-----
/etc/abrt/abrt-http.conf
    Default config file. For details on how to configure it, visit it.
/etc/pam/abrt-http
    File containing PAM rules for abrt-http.
```