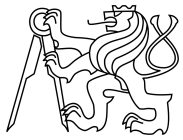
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Virtuální emulace rozhraní pro SOHO embedded zařízení typu router
Student:	Martin Bednár
Vedoucí:	Ing. Michal Vaněk
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Nastudujte možnosti hledání softwarových zranitelností v SOHO zařízeních se zaměřením na domácí routery pomocí virtualizované emulace jejich webového rozhraní. Jednotlivé možnosti analyzujte. Vybranou metodu na základě domluvy s vedoucím práce experimentálně aplikujte.
Postup:
1. Vyhodnoťte možnosti hledání zranitelností v SOHO zařízeních (se zaměřením na routery) s důrazem na virtuální emulaci daného prostředí.
2. Analyzujte strukturu firmware (file system, web server, způsob uložení a načítání konfigurace) u několika modelů routerů od různých výrobců.
3. Dle dohody s vedoucím práce u jednoho modelu emulujte jeho webový server.
4. Porovnejte počet známých nalezených zranitelností na reálném zařízení a na jeho emulovaném prostředí.
5. Diskutujte možnosti automatizace procesu analýzy a emulace dalších SOHO zařízení.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 21. září 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Virtuální emulace rozhraní pro SOHO embedded zařízení typu router

Martin Bednár

Vedoucí práce: Ing. Michal Vaněk

17. února 2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 17. února 2017 .....

Odkaz na tuto práci

Bednár, Martin. *Virtuální emulace rozhraní pro SOHO embedded zařízení typu router*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

S rostoucí dostupností internetu po celém světě roste i počet užívaných SOHO routerů, centrální prvek každé domácí sítě. SOHO routery často obsahují bezpečnostní chyby, čímž ohrožužují všechna zařízení na síti. Vysoké množství výrobců a koncových zařízení činí manuální analýzu nepraktickou. V této práci budeme prezentovat základ jednoho přístupu pro hledání těchto chyb: emulace služeb routeru ve virtuálním prostředí.

Klíčová slova router,virtualizace,web,server,SOHO,zranitelnost,qemu

Abstract

With the rise of Internet availability worldwide, so is the amount of SOHO routers, the central point in every home network, in use. These routers often contain security vulnerabilities, jeopardizing the security of all devices on the network. The high number of different device manufacturers and models makes an analysis of every device impractical. In this paper we will present the basic elements necessary to launch router services in a virtual machine.

Keywords router,virtualization,web,server,SOHO,vulnerability,qemu

Obsah

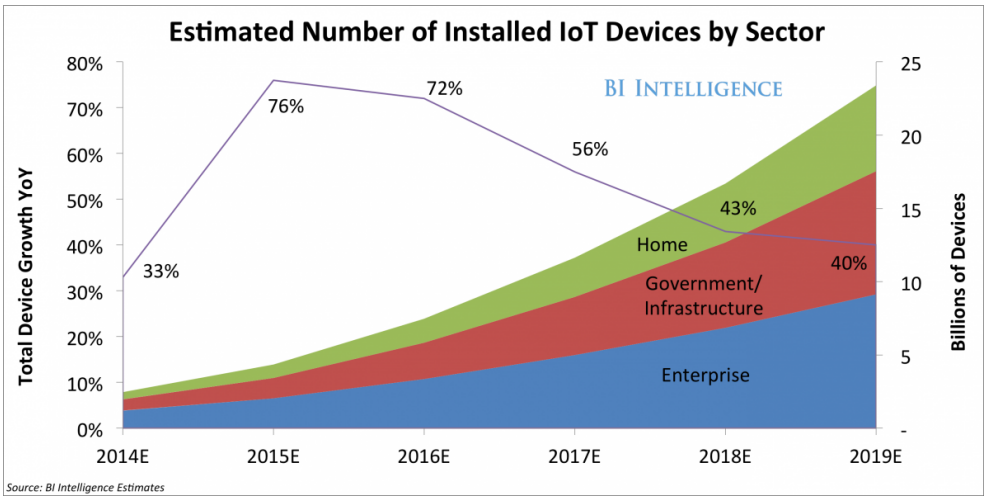
Seznam obrázků

Úvod

V dnešní době se rychle rozšiřuje[1] počet koncových uživatelských zařízení připojených do sítě internet, na které z různých důvodů (nemožnost přidání software, nedostatečný výkon, ...) nelze aplikovat tradiční obranné techniky, jako jsou například antiviry. Tato zařízení jsou často špatně zabezpečená; jejich nejčastější chybou je otevřený telnet s výchozím/jednoduchým heslem [2, 3]. Těchto zranitelností využívají různí aktéři, od NSA[4, 5] po různé hackerské skupiny. Důvodem těchto zranitelností často bývá kombinace snahy výrobce o optimalizaci nákladů spojených s výrobou a snahy udělat zařízení co nejvíce *Plug & Play*[1]. Trh s těmito *IoT*[2] zařízeními nejeví známky zpomalování.

Centrálním prvkem každé domácí sítě je domácí router. Ten slouží jako

[1] Možnost začít používat ihned po zapojení
[2] Internet of Things



Obrázek 0.1: Odhadovaný růst IoT zařízení

první (a často i poslední) obrana proti útokům na uživatelská zařízení ze sítě internet. Přes tuto kritickou funkci se výrobci routerů málokdy zabývají bezpečností, a firmware je často plný (nejen) bezpečnostních chyb. Málokdy se zabývají možností aktualizace již prodaných zařízení; většina dnes dostupných routerů má k datu prodeje několik let starý firmware, málokdy automaticky aktualizovatelný[3]. Jako příklad bezpečnostních chyb se dá například uvést implementace protokolu TR-069, která byla terčem velkého útoku na konci roku 2016[6]. Často se ale bezpečnostní chyby týkají webového rozhraní, v dnešní době hlavní způsob nastavování SOHO routeru.

Díky této špatné bezpečnosti domácích sítí je velice lukrativní si z nich stavět botnet jako je například Mirai[7] nebo nově Leet[8]. Jiný typ útoků představuje DNS hijack, kde útočník změní nastavení routeru, aby prováděl DNS dotazy na DNS server s podvrženými záznamy, a tím zjednodušil phishing útoky[9]. Phishing útoky jsou zjednodušeny tím, že útočník přesměruje síťovou komunikaci na jím kontrolovaný server, kde například připraví stránku vypadající jako skutečná stránka banky. Nic netušící uživatel tak útočníkům dobrovolně dá jméno a heslo.

Vzhledem k tomu, že běžný uživatel nesleduje novinky ze světa počítačové bezpečnosti, natož aby sledoval, zda jím vlastněné zařízení je zasaženo nějakou bezpečnostní dírou, měli by se o bezpečnost starat sami vývojáři. To se ale neděje, aktuálně je bezpečnost plně v rukou uživatelů. Vzhledem k počtu různých zařízení není reálné, aby se každé analyzovalo ručně. Aby alespoň existovala možnost uživatele informovat o potenciálních problémech, je potřeba o problémech vědět, a tudíž je potřeba mít možnost automaticky analyzovat firmware různých zařízení.

V této práci manuálně zprovozním jednu službu jednoho routeru, s inspirací pro návod branou ze systému Firmadyne[10]. Experimentálně zkusím, zda je reálné pomocí virtualizace zkoumat jednotlivé služby routerů bez fyzického přístupu k modelu. Nejprve provedu diskuzi ohledně zvolené metody emulace a virtualizace, a následně metody aplikuji na jednu službu z jednoho konkrétního modelu routeru. Po spuštění služby ji zkusím porovnat se stejnou službou na reálném routeru a nakonec provedu diskuzi ohledně možností automatizace celého procesu.

[3] uživatel většinou musí ručně firmware stáhnout ze stránek výrobce, a následně ručně nahrát do routeru přes webové rozhraní

Kapitola 1

Cíl práce

Cílem práce je prozkoumat možnost využití virtuálních strojů pro analýzu routerů, diskutovat zda je tento způsob analýzy použitelný, a zda je automatizovatelný.

Budu se snažit ve virtuálním kontrolovaném prostředí spustit webový server routeru DHP-1565. Tímto způsobem se sice z výsledků vyloučí veškeré zranitelnosti závislé na hardware, ale vzhledem k poměru počtu chyb v software k počtu chyb v hardware[4] nás hardwarové chyby v kontextu této práce nezajímají. Webový server nás zajímá, protože je to služba dostupná na všech SOHO[5] routerech, a protože i na různých dalších *embedded* zařízeních nahradil protokoly jako jsou **SNMP** nebo **telnet** jakožto primární způsob konfigurace, a jedná se tudíž o prvotní útočný vektor. Konfigurační rozhraní bývají webové aplikace, které jsou častým zdrojem bezpečnostních chyb, jak lze vyčíst z `http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database`.

Po přečtení této práce by měl být čtenář schopný provést základní analýzu jednotlivých firmware routerů (možná i jiných zařízení) a vědět, jak si připravit prostředí pro vlastní pokusy o spuštění služby. Zároveň tato práce slouží jako návod pro spuštění webového serveru z routeru DHP-1565 ve virtuálním prostředí.

[4] čtenář si sám může porovnat kolikrát narazil na chybu, kterou dostopoval až do hardware, a kolikrát narazil na chybu čistě softwarového rázu.

[5] Small office/Home Office

Kapitola 2

Analýza routeru

Spouštění jednotlivých služeb routeru mimo samotný hardware bude vyžadovat vědomosti o interním fungování softwarového vybavení routeru. Tyto vědomosti by bylo možné vyčíst ze zdrojových kódů, ale o těch nemůžeme předpokládat, že budou dostupné a kompletní. Navíc by to bylo časově velice náročné. Další způsob je analyzovat samotný firmware routeru, a z něj pozorováním zjistit, jak co funguje.

2.1 Typy analýz

2.1.1 Automatická analýza

Analýzu software můžeme rozdělit na 2 druhy:

- *Statická analýza*, kde nějakému nástroji předám zdrojové kódy, a na výstupu dostanu seznam potenciálních problémů od nejasného uzávorkování podmínky, po detekci race conditiony. Příklady těchto nástrojů jsou veškeré kompilátory, clang-analyzer, coverity pro C++; pylint, pyflake pro python, exakat[6] (jeden z mnoha pro PHP), a další pro různé programovací jazyky.
- *dynamická analýza* spočívá ve spuštění programu ať už na reálném hardware nebo ve virtuálním prostředí a pozorování jeho chování. Do této kategorie lze zařadit známé nástroje jako gdb, valgrind, ale i pokročilé fuzzing nástroje jako je například AFL[7]

Ve světě routerů je statická analýza relativně jednoduše proveditelná vzhledem k tomu, že většina výrobců používá firmware založený na prostředí GNU/-Linux. Toto prostředí je licencováno licencí GPL, což výrobce nutí zveřejňovat zdrojový kód. Pro potřeby této práce se ale budu věnovat dynamické analýze.

[6] `https://github.com/exakat/php-static-analysis-tools`
[7] American fuzzy loop `http://lcamtuf.coredump.cx/afl/`

2.1.2 Dynamická analýza

Dynamická analýza routeru v našem případě spočívá ve zkoušení různých známých zranitelností, které jsou většinou využitelné zasláním speciálně udělaného síťového packetu, nebo webového požadavku. K tomu slouží například Nessus vulnerability Scanner[8], nebo OpenVAS[9]

K dynamické analýze je tedy potřeba mít funkční router, respektive tu část jeho software, kterou chceme zkoušet.

První možnost je pořídit všechny modely, a zapojit je do jedné obrovské, kontrolované sítě. Vzhledem k vysokému počtu různých výrobců, modelů a jejich revizí není tento přístup reálný. (kvůli pořizovací ceně, a kvůli potřebné infrastruktuře)

Druhá možnost je využít faktu, že jednotlivé modely se skutečně vyskytují na internetu, a sprostě provádět sken celého internetu, s tím, že by se mohly nalezené zranitelnosti opravit, jak to dělal Linux.WiFatch[11]. Tím by se zbytečně zatížila síť a v některých případech by mohlo zařízení být úplně znefunkčněno. Navíc je to počínání ve většině světa nelegální a obecně neetické.

Třetí možností je jednotlivé firmware, případně jejich části které mě zajímají, spustit ve virtuálním prostředí. To obnáší, připravit si virtuální stroj (kontrolované prostředí), ve kterém se posléze budu snažit zprovoznit dané služby. Toto řešení je celkem nenáročné na hardware, a proto vypadá nejzajímavěji.

Dynamickou analýzu tedy budu dělat ve virtuálním prostředí, ve kterém si spustím tu část routeru, která mě právě zajímá.

[8] `https://www.tenable.com/products/nessus-vulnerability-scanner`
[9] `http://www.openvas.org/`

Kapitola 3

Způsoby Virtualizace

Virtualizace je emulace počítače pomocí software na jiném počítači (fyzickém nebo virtuálním). Dělí se na několik druhů:

- *Plná Virtualizace*, kde je emulováno dostatek hardware aby mohl být použit nezměněný operační systém – VMWare, Qemu, VirtualBox
- *Paravirtualizace*, kde hostovaný operační systém volá speciální API, které je definováno hypervizorem – Xen, Parallels
- *Virtualizace na úrovni operačního systému*, kde hostující operační systém poskytuje prostředky pro izolaci aplikací mezi sebou. Tzn, že jádro je sdílené mezi všemi aplikacemi, ale ty jsou od sebe logicky oddělené v uživatelském módu. – FreeBSD jails, Linux containers (Docker)

Pro naše potřeby se nejlépe hodí plná virtualizace, převážně z důvodu že potřebujeme virtualizovat kompletní operační systém routeru, pokud možno beze změny. Druhý důvod je potřeba mít možnost přidávat hardwarová zařízení jako jsou síťové řadiče, disky apod dle požadavků jednotlivých firmware a služeb.

K plné emulaci se nabízí hned několik jednoduše dostupných a známých produktů:

- VirtualBox
- VMWare
- Qemu

Zde jsme zvolili Qemu protože jako jediné podporuje všechny architektury na kterých je postavena většina routerů. (jmenovitě x86, arm, armel, mips, mipsel)[12] Navíc je jako jediný z trojice uvolněn pod licencí GPL, což umožňuje jednoduché úpravy v případě potřeby nebo nalezení chyby.

3.1 QEMU

Qemu umožňuje dva typy virtualizace: *usermode emulation* a *full system emulation*.

- *usermode emulation* - Mód qemu, kde není potřeba spustit kompletní virtuální stroj, ale Qemu za běhu překládá systémová volání. Stačí použít již existující hierarchii souborů kde jsou veškeré potřebné knihovny pro chtěný binární soubor. K tomu můžu buď Qemu nasměrovat na linker správný pro chtěnou platformu, nebo použít kompletní chroot[13]. Vzhledem k naší potřebě použít co nejvíce částí firmware routeru se více hodí metoda chroot. Jelikož se binární soubor spouští z chrootového prostředí, je potřeba aby emulátor buď uměl chroot zařídit sám, což Qemu neumí, nebo byl v cestě chrootu. Qemu tedy musí být v samotném chrootu, a tedy musí být staticky linkován. Toto řešení neumožňuje měnit hardware, který vidí spouštěný program bez změny reálného hardware (či jeho nastavení), na kterém se má spouštět. Jako velikou nejistotu tohoto řešení vnímám fakt, že qemu nemá podporu pro všechna systémová volání[14], a tak může občas nefungovat. Navíc v případě interakce zkoumané aplikace s jádrem, může shodit celý systém.

- *full system emulation* - Jedná se o emulaci celého počítače, při které můžeme jednoduše měnit viditelný hardware. Není zde žádná náhodnost z hlediska podpory systémových volání, a v případě nežádoucí interakce s jádrem stačí znova spustit virtuální stroj.

Při využití plné systémové emulace se nám nabízejí opět dvě možnosti: použít z firmware všechno včetně jádra, nebo použít standardní linuxovou distribuci na správné architektuře, rozbalit firmware, a využít funkcionality chroot pro použití knihoven z firmware. Využití jádra z firmware skýtá hned několik problémů: jednak není úplně jednoznačné jak jádro ze zabaleného firmware vybalit, dále pravděpodobnost, že jádro z routeru bude mít potřebné ovladače pro fungování ve virtuálním stroji není vůbec veliká. Důvodem je stáří jader ve firmware, a snaha o úsporu místa, kvůli které jádra mívají zakompilované ovladače pouze pro hardware z konkrétního modelu.

Kapitola 4

Příprava prostředí

Pro spuštění chtěné služby je potřeba mít připravený virtuální stroj a být s ním schopen komunikovat. Dále potřebuji mít rozbalený souborový systém routeru a potřebuji znát alespoň základní rozložení firmware, abych věděl, kde hledat binární soubor služby.

Předpokladem pro dynamickou analýzu jsou tedy obecné vědomosti o struktuře firmware, a je nezbytné provést krátkou statickou analýzu.

4.1 Výběr modelu

Pro experiment jsem vybral model D-Link DHP-1565, a to ze dvou důvodů: jednak protože ho mám doma, a mám předchozí znalosti o tomto modelu, a jednak proto, že údajně má známou zranitelnost, zneužitelnou přes webový server[15].

4.2 Analýza modelu D-Link DHP-1565

Výrobci routerů většinou zpřístupňují různé verze firmware pro různé modely routeru na svých stránkách v podobě binárního souboru, který se pak nahraje na router. Tento soubor obsahuje celý souborový systém, jádro (většinou nějak upravený linux) a v některých případech i zavaděč.

4.2.1 Firmware

Pro prvotní analýzu nám pomůže spuštění příkazu *file* na firmware pro D-Link DHP-1565 stažený ze stránek výrobce[10])

[10] `http://support.dlink.com/ProductInfo.aspx?m=DHP-1565`

```
DHP1565A1_FW101B10.bin: u-boot legacy uImage, Linux
    Kernel Image, Linux/MIPS, OS Kernel Image (lzma),
    1286741 bytes, Thu Mar 15 12:42:12 2012, Load Address
    : 0x80002000, Entry Point: 0x802ACD30, Header CRC: 0
    x7F98CEC1, Data CRC: 0x9316E3BE
```

Dle výstupu **file** se jedná o architekturu MIPS, o linux, a router používá zavaděč u-boot. Z popisu virtualizace v předchozí kapitole je zřejmé, že zavaděč je pro naše potřeby čistě informační záležitost, a jeho znalost nám nepomůže při virtualizaci služeb.

Jako další způsob zjištění informací ohledně firmware využijeme nástroj binwalk[11]. Spuštění binwalk na firmware D-Link DHP-1565 podalo následující informace:

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size
    : 64 bytes, header CRC: 0x7F98CEC1, created:
    2012-03-15 12:42:12, image size: 1286741 bytes, Data
    Address: 0x80002000, Entry Point:0x802ACD30, data CRC
    : 0x9316E3BE, OS: Linux, CPU: MIPS, image type: OS
    Kernel Image, compression type: lzma, image name: "
    Linux Kernel Image"
64            0x40            LZMA compressed data,
    properties: 0x5D, dictionary size: 8388608 bytes,
    uncompressed size: 3781120 bytes
1310720       0x140000        Squashfs filesystem,
    little endian, version 4.0, compression:lzma, size:
    8120492 bytes, 1240 inodes, blocksize: 65536 bytes,
    created: 2012-03-15 12:42:24
```

Binwalk celý soubor přečte, a hledá v něm jemu známé sekvence bajtů. Jak se můžeme přesvědčit v hexadecimálním editoru, `DHP1565A1_FW101B10.bin` začíná sekvencí

```
0x27051956
```

To je *magic number* u-boot hlavičky.

[11] `http://binwalk.org`

```
typedef struct image_header {
 uint32_t        ih_magic;        /* Image Header Magic
    Number */
 uint32_t        ih_hcrc;         /* Image Header CRC
    Checksum */
 uint32_t        ih_time;         /* Image Creation
    Timestamp */
 uint32_t        ih_size;         /* Image Data Size */
 uint32_t        ih_load;         /* Data  Load  Address
    */
 uint32_t        ih_ep;           /* Entry Point Address
    */
 uint32_t        ih_dcrc;         /* Image Data CRC
    Checksum */
 uint8_t         ih_os;           /* Operating System */
 uint8_t         ih_arch;         /* CPU architecture */
 uint8_t         ih_type;         /* Image Type */
 uint8_t         ih_comp;         /* Compression Type */
 uint8_t         ih_name[IH_NMLEN];       /* Image Name */
} image_header_t;
```

magic number je hned první položka struktury. Díky známosti struktury hlavičky může binwalk zjistit další informace o firmware. Zde indikuje, že ve firmware je jedna položka činící data komprimovaná pomocí LZMA, a druhá Squashfs systém souborů, též komprimovaný pomocí LZMA. Ve firmware očekáváme jádro a souborový systém. Souborový systém binwalk identifikoval, a druhá položka bude asi jádro.

Přepínač *-e* binwalku jednotlivé části rozdělí do samostatných souborů, které rozbalí pokud může, a můžeme pokračovat v analýze. U firmware D-Link DHP-1565 vzniknou soubory

```
140000.squashfs
40
40.7z
```

Názvy souborů odpovídají položce "HEXADECIMAL" ve výstupu binwalk, a to znamená, že toto číslo odpovídá offsetu počátku dat ve firmware.

4.2.1.1 Soubor 40

Soubor 40 je rozbalený soubor 40.7z, odhadem se jedná o jádro. Tento odhad je potvrzen po jeho analýze pomocí **strings**. Ve výstupu **strings** jsem našel řetězce obsahující "Linux". Jeden jádro udal i s verzí:

```
Linux version 2.6.31--LSDK-9.2.0.312 (root@fred-laptop)
    (gcc version 4.3.3 (GCC)) #1 Thu Mar 15 20:34:29 CST
    2012
```

Pro potvrzení, že se jedná o jádro, jsem hledal řetězce, které vypadají jako argument funkce printf() (linuxová funkce printk() má podobné formátovací řetězce jako printf()) a některé náhodně vybrané jsem hledal ve zdrojovém kódu linuxu (verze 4.4.39). Dva příklady jsou zde uvedené.

```
"%s: module license '%s' taints kernel."
kernel/module.c:2310:  pr_warn("%s: module license '%s'
    taints kernel.\n"
```

```
"crashkernel: Memory value expected"
kernel/kexec_core.c:1066:                              pr_warn
    ("crashkernel: Memory value expected\n");
```

Mohu tedy s velikou jistotou říct, že soubor 40 obsahuje jádro pro router D-Link DHP-1565.

4.2.1.2 Soubor 140000.squashfs

`140000.squashfs` obsahuje souborový systém routeru. Někde v tomto archivu je nejspíš ukryt webový server, který chceme spustit. Dle příkazu **file** soubor `140000.squashfs` obsahuje squashfs souborový systém.

```
140000.squashfs: Squashfs filesystem, little endian,
    version 4.0, 8120492 bytes,
1240 inodes, blocksize: 65536 bytes, created: Thu Mar 15
    12:42:24 2012
```

Pro squashfs existuje nástroj **unsquashfs**, který ze souboru vybalí adresářovou hierarchii a soubory. První pokus s verzí z repozitáře distribuce nedopadl úspěšně:

```
lzma uncompress failed with error code 0
read_block: failed to read block @0x7bdc1d
read_fragment_table: failed to read fragment table index
FATAL ERROR: failed to read fragment table
```

Běžně dostupné nástroje pro Squashfs zjevně nedokážou soubor rozbalit. Díky licenci GPL musí D-Link zveřejňovat změny provedené k nástrojům Squashfs. Ty najdeme v balíčku zdrojových kódů na serverech D-Link[12]. Tím je možné zkontrolovat zda není squashfs firmou D-link modifikovaný. Ze staženého archivu vidíme, že použitá verze squashfs je 4.0. (archív této verze

[12] `ftp://ftp.dlink.de/dhp/dhp-1565/driver_software/DHP-1565_reva_GPL_code.rar`

můžeme najít na stránkách projektu[13]) Pomocí jednoduchého skriptu zjistíme jak se squashfs od D-Link liší od oficiální verze:

```
cd squashfs-4.0/squashfs-tools
for i in *;
    do diff -s -d "$i" ../../../src/AthSDK/tools/
       squashfs-4.0/squashfs-tools/"$i";
done
```

Z rozdílů (Příloha B) lze usoudit, že D-Link přidal do squashfs možnost LZMA komprese. Squashfs-tools sice od verze 4.1 LZMA kompresi podporují, nicméně se dá předpokládat, že oficiální implementace se bude lišit od implementace D-Link[16]. Proto lokální unsquashfs verze 4.3 si s tím neporadí. Nabízí se několik řešení:

- Lokálně si upravit a sestavit Squashfs.
- Použít nástroj **Sasquach**[14].
- Použít **firmware-mod-kit**[15], kde jsou shromazděné úpravy squashfs a jiných nástrojů, a to jak pro výrobce D-Link, tak i pro řadu jiných.

Použiji Fw-mod-kit, protože ulehčí práci při potenciální analýze jiných modelů. Po rozbalení archivu pomocí Fw-mod-kit vidíme téměř standartní hierarchii souborů.

[13] `https://sourceforge.net/projects/squashfs/`
[14] `https://github.com/devttys0/sasquatch`
[15] `https://bitsum.com/firmware_mod_kit.htm`

```
bin
dev
etc
https
lib
libexec
linuxrc
mnt
plc
proc
root
sbin
share
sys
tmp
usr
var
www
```

Oproti standardní hierarchii souborů dle FHS[16][17] zde máme v adresáři `/` navíc adresáře `https`, `libexec`, `share`, `www` a `plc`. Adresáře `libexec` a `share` se standartě nacházejí v adresáři `/usr`, a adresář `www/` v `/var`. Adresáře `https` a `plc` jsou zde navíc. Adresář `https` obsahuje pouze odkaz na `www/cgi/ssi`, což je binární spustitelný soubor:

```
# file www/cgi/ssi
www/cgi/ssi: ELF 32-bit MSB executable, MIPS, MIPS32
    rel2 version 1 (SYSV), dynamically linked,
    interpreter /lib/ld-uClibc.so.0, stripped
```

SSI (Server Side Includes) je služba, která mění obsah HTML stránky ještě už straně serveru. To je první indíce jak webový server funguje.

Adresář plc obsahuje několik binárních souborů s neidentifikovaným obsahem:

```
# file plc/plc.*
plc/plc.ceb.pib:    data
plc/plc.na.mp.pib:  data
plc/plc.na.pib:     data
plc/plc.nvm:        data
```

Už samotný název adresáře "`plc`" (zkratka pro *Power-line communication*[17]) lze usoudit, že se jedná o soubory potřebné pro funkcionalitu powerline. Ty

[16] Filesystem Hierarchy Standard

[17] *PLC* je způsob přenosu dat přes elektrické rozvody. Funguje na vrstvě OSI 1.

jsou z hlediska virtualizace služeb nezajímavé, protože potřebný hardware nebudeme mít k dispozici. Obsah ostatních adresářů budu prověřovat až podle potřeby, vzhledem k dodržení Filesystem Hierarchy standartu.

4.2.1.3 HTTP server

Příkazem **find** zkusíme zjistit, o jaký http server router používá.

```
find -iname "*http*"
./bin/http_login
./etc/lighttpd.conf
./etc/lighttpd.user
./etc/lighttpd_base.conf
./https
./mnt/lighttpd.conf
./mnt/lighttpd.user
./mnt/lighttpd_base.conf
./sbin/lighttpd
./sbin/lighttpd-angel
./www/http_storage.asp
```

Výstup napovídá, že se jedná o lighttpd. Další analýza spočívá ve sledování serverem linkovaných knihoven, a bude provedena až ve virtuálním stroji. Příkaz **ldd** pro svoji funkčnost využívá výchozí linker daného systému[18]. Pro zatím je analýza prováděna na pracovní stanice na architektuře x86, a router je na architektuře MIPS.

4.3 Příprava virtuálního prostředí

K dynamické analýze ve virtuálním stroji je potřeba funkční operační systém stejného typu, jako má router, a na stejné architektuře. V tomto případě je potřeba Linux na architektuře MIPS. Ač by stačil jakýkoliv linux, pro jednoduchost jsem zvolil populární Debian.

Jak již bylo uvedeno v 3.1 , jako virtuální stroj jsem zvolil Qemu. V této kapitole se budeme zabývat spuštěním stroje s požadovanými vlastnostmi.

4.3.1 Příprava stroje

Z nalezeného návodu se inspiruji pro způsob přípravy MIPS virtuálního stroje: nainstalujeme a spustíme Debian pro architekturu MIPS do virtuálního stroje[19]. Tento připravený virtuální stroj bude sloužit jako šablona, aby pro všechny následující pokusy bylo jednoduché systém navrátit do původního, čistého stavu. Za tímto účelem využiji *Redirect on Write* vlastnosti formátu **qcow2**.

4.3.2 Disk

Formát virtuálních pevných disků **qcow2** umožňuje využití jednoho virtuálního disku jako základ, od kterého se větví další pevné disky. Způsob využití je jednoduchý: Připraví se základní virtuální disk, a následně se vytvoří nový virtuální disk, kterému se do parametru *backing_file* předá cesta k základnímu pevnému disku.

```
$ qemu-img create -f qcow2 -o nocow=on,backing_file=
    base.qcow2 new.qcow2
```

Qemu interně používá mechanismus *Redirect on Write*. Základní virtuální disk používá pouze pro čtení, zatím co nový používá pro veške zápisy, a čte z něj veškerá změněná data. Oproti *Copy on Write*, které využívá například souborový systém **BTRFS** má *Redirect on Write* tu výhodu, že se pozměněná data nemusí nejdřív vykopírovat[20].

Využití *backing store* vlastnosti formátu **qcow2** má tu výhodu, že rozdíly různých instancí jsou v samostatných souborech, a že základní virtuální disk může zůstat relativně malý.

4.3.3 Komunikace s OS

Vzhledem k síťové povaze routeru, a vědomosti, že z jeho webového rozhraní se nastavují síťová rozhraní, se domnívám, že samotný web server může činit změny v nastavení sítě. Proto nemohu považovat komunikaci s virtuálním strojem přes jakýkoliv síťový protokol (jako jsou např **telnet**, **ssh**) za spolehlivou. Rozhodl jsem se tedy použít jako komunikační kanál sériovou linku. V případě potřeby můžeme přes sériovou linku pomocí **SLiRP** zavést TCP spojení. Pro naše účely držím tuto možnost pouze v patrnosti, jelikož si myslím, že jedna sériová linka bude stačit.

Qemu nastaví ve virtuálním stroji zařízení sériové linky, a vyvede jej do hostitelského stroje. K tomu slouží následující přepínače.

```
-chardev pty,id=1 -serial chardev:1
```

přepínač *-serial* vytvoří ve virtuálním stroji zařízení sériové linky, a přepínač *-chardev* propojí qemu s PTY (Pseudoterminál) na hostitelském stroji. Navzájem tyto dvě zařízení propojí pomocí společného identifikátoru. (Zde **1**)

Pseudoterminály na hostitelském stroji jsou v `/dev/pts/`. Zjistíme, na který Qemu sériovou linku připojí buď z výstupu Qemu na STDOUT, nebo pomocí protokolu QMP.

Vzhledem k tomu, že Debian používá systemd, je ještě potřeba systému říct o sériové lince příkazem

```
systemctl enable serial-getty@ttyS0
```

4.3.4 Qemu Machine Protocol

QMP[18] slouží ke komunikaci a ovládání Qemu přes socket pomocí předávání informací ve formátu **JSON**. Tento protokol nám bude užitečný hlavně později při automatizaci spouštění virtuálních strojů. QMP má jednoduchou implementaci v Pythonu[19], pomocí které se samotného Qemu dotážeme, kam připojil sériovou linku v hostitelském stroji:

```
import qmp

mon = qmp.QEMUMonitorProtocol(sys.argv[1])
greeting = mon.connect()
print(greeting)
chardevs = mon.cmd("query-chardev")
pts = None
for dev in chardevs['return']:
    if dev['frontend-open'] and 'filename' in dev:
        print(dev['filename'])
```

4.3.5 Sdílení souborů

Vzhledem k omezení na nesíťové prostředky nemůžeme použít běžné způsoby sdílení mezi počítači, jako jsou například síťové protokoly **Samba** či **NFS**.

4.3.5.1 Sériová linka

Jedna z možností je posílat data přímo přes sériovou linku a následně je přijmat ve virtuálním stroji pomocí:

```
$cat -> file.out
```

Tato možnost se ale v experimentech neosvědčila. Komunikace s virtuálním strojem přerušila hláškou:

```
serial8250: too much work for irq3
```

Během hledání původu chybové hlášky jsem dospěl k názoru, že se nejspíš jedná o chybu v jádře[20], a že bude daleko jednodušší najít jiný způsob sdílení souborů s virtuálním strojem.

[18] Qemu Machine Protocol

[19] `http://git.qemu.org/?p=qemu.git;a=blob;f=scripts/qmp/qmp;h=514b539a6b202051410119cce3ecdb39307c53d7;hb=HEAD`

[20] `https://bugzilla.redhat.com/show_bug.cgi?id=986761`

4.3.5.2 Virtfs

Qemu nám nabízí takzvaný **virtfs**, který exportuje do virtuálního stroje specifikovaný adresář přes protokol 9P[21]. Tento způsob nevyžaduje síť, a je na linuxových distribucích běžně dostupný.

```
-virtfs local,path=/tmp/extracted-firmware,mount_tag=1,
    security_model=none,id=1
```

Ve virtuálním stroji je pak možné se do sdíleného adresáře dostat po jeho připojení; zde nasdílený adresář připojíme do `/mnt/fw_orig`:

```
mount -t 9p /mnt/fw_orig -oversion=9p2000.L
```

Originální firmware je potřeba nakopírovat do virtuálního stroje, a potom už jenom nastavit síť.

4.3.6 Síťová připojení

Síťové připojení je nezbytné, protože bez ně by nebylo jak se připojit k webovému serveru. Qemu nabízí hned několik možností síťování:

- *User mode networking* – qemu se stará o přidělení adres hostovi. Nevýhodou je, že musíme předem znát porty na které se budeme do virtuálního stroje připojovat a ty správně přesměrovat.
- *Tap* – kde se použije zařízení typu TAP z hostitele. Přidělení IP adres je potřeba udělat manuálně, ale zato budeme mít přístupné všechny porty.

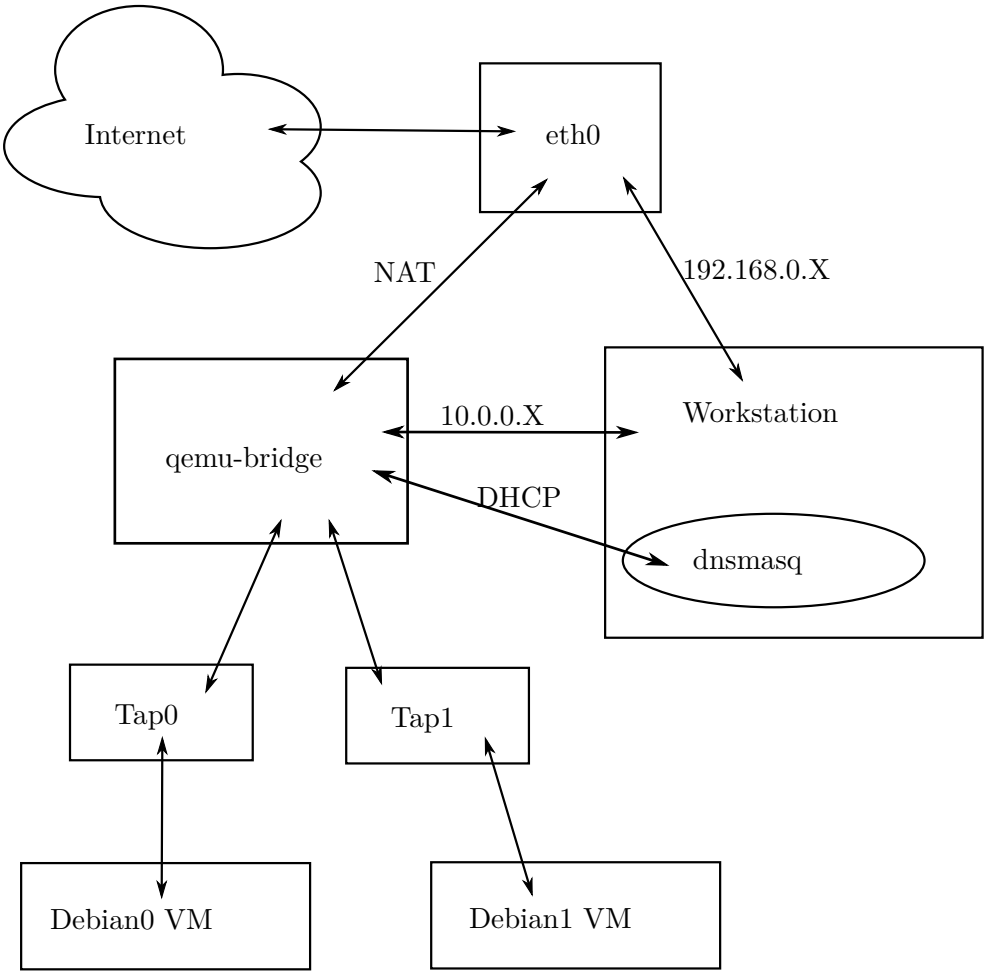
Zde jsem vybral metodu *tap* zařízení, protože nemusím předem znát všechny porty, které budu potřebovat.

Vytvořené TAP zařízení přidám do bridge zařízení, na kterém běží DHCP server *dnsmasq* (Obr 4.1).

Konečný příkaz pro spuštění virtuálního stroje je tedy

```
qemu-system-mips -M malta -drive file=/home/martin/
    projects/REMUs/virt2/HNS-MIPS-3081.qcow2,format=qcow2
    ,if=ide,id=drive-0 -qmp unix:/tmp/qmp.sock,server -
    virtfs local,path=/home/martin,mount_tag=s0,
    security_model=none,id=s0 -netdev tap,ifname=tap0,id
    =net0,script=no,downscript=no -device e1000,netdev=
    net0 -chardev pty,id=serial0 -serial chardev:serial0
    -kernel ./vmlinux-3.16.0-4-4kc-malta -append '"root=/
    dev/sda1"' -nographic
```

[21] 9P je protokol vyvinut pro potřeby operačního systému *Plan9* vyvinutého v 90. letech původními autory systému Unix.



Obrázek 4.1: Zapojení sítě

4.4 Dynamická analýza běžícího procesu

Pro dynamickou analýzu procesů existuje pro Linux několik nástrojů

- strace
- eBPF
- ltrace
- systemtap
- ftrace

4.4.1 strace

Strace je nástroj pro monitorování interakce mezi aplikací v uživatelském režimu a jádrem. Je možné jím sledovat provedená systémová volání, signály a změny stavu procesů. Jedná se o známý a běžně dostupný nástroj.

4.4.2 eBPF

eBPF[22] je virtuální stroj přímo v jádře linux pro který se píší specifické programy, které se následně přeloží do eBPF bytecode a předají zmíněné virtuální mašině. Hlavní užitek je při performance analýzách, jelikož má malou režii. Jedná se o celkem novou technologii, a je dostupná pouze na novějších jádrech. Navíc není podporována architektura MIPS.

4.4.3 ltrace

ltrace umožňuje sledovat jaké funkce daný program volá z nalinkovaných knihoven. V kombinaci s strace to umožní vidět celou hierarchii volání pro uskutečnění nějaké akce. Vzhledem k použití strace nechci ltrace v prvním přiblížení použít protože vnímám jako reálnou možnost kolizi mezi strace a ltrace. Konkrétně v případě použití strace a ltrace zároveň by použití ltrace nejspíš nezanedbatelně ovlivnilo výstup strace.

4.5 Použití *strace*

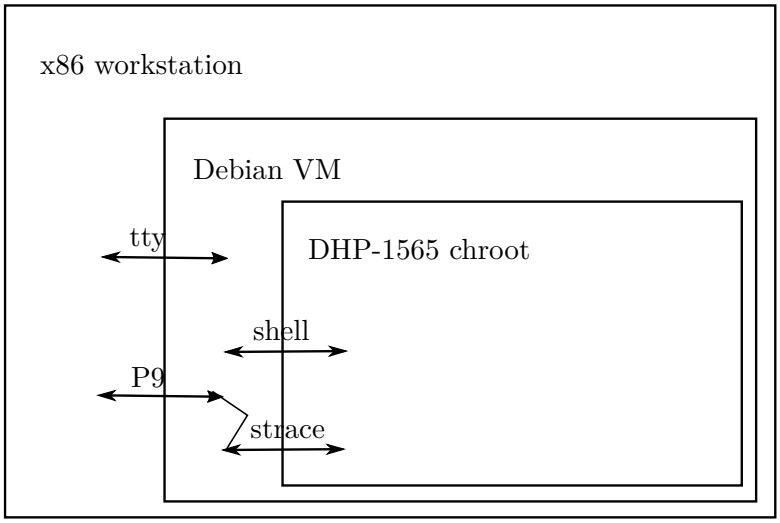
Samotný výstup z *Strace* není vzhledem k množství informací sám o sobě moc užitečný, a jeho výstup by se nám hodilo agregovat. K tomu jsem našel a rozšířil o potřebné moduly framework Stana[23]. Pro prostředí co nejbližší prostředí firmware provedeme změnu kořenového adresáře na adresář rozbaleného firmware. K tomu nám poslouží příkaz **chroot**. Chroot nás nepustí mimo adresář do kterého jsme se přepnuli, musíme přes strace spouštět už chroot.

Pro analýzu jsem vybral strace čistě z důvodu jednoduché dostupnosti a protože mně pro začátek zajímají převážně systémová volání týkající se přístupu k souborům. (Pro eBPF bych musel najít či přeložit novější jádro pro MIPS)

4.6 Shrnutí

V této části jsem popsal, jak si připravít virtuální stroj s distribucí Debian, připojení adresáře z hostitelského stroje obsahující rozbalený firmware, a připravení sítě a komunikace s virtuálním strojem (Obr 4.2). Nyní nakopírujeme

[22] `https://www.iovisor.org/technology/ebpf`
[23] `https://github.com/johnlcf/Stana`



Obrázek 4.2: Virtualizační architektura

rozbalený firmware a přepneme se do kořenového adresáře rozbaleného firmware pomocí chroot, který spouštíme skrze strace.

Kapitola 5

Spouštění webového serveru lighttpd ve virtuálním stroji

V předchozích kapitolách jsem prozkoumal jak by šlo virtualizovat služby z routeru, našel jednu takovou službu v konkrétním modelu routeru, a nyní se budu věnovat tomu, jak službu spustit. Jedná se o webový server, konkrétně lighttpd. Pro připomenutí je zde stručný seznam přípravných kroků:

1. Spustit připravený virtuální stroj

```
qemu-system-mips -M malta -drive file=/home/martin/
    projects/REMUs/virt2/HNS-MIPS-3081.qcow2,format=
    qcow2,if=ide,id=drive-0 -qmp unix:/tmp/qmp.sock,
    server -virtfs local,path=/home/martin,mount_tag
    =s0,security_model=none,id=s0 -netdev tap,
    ifname=tap0,id=net0,script=no,downscript=no -
    device e1000,netdev=net0 -chardev pty,id=serial0
     -serial chardev:serial0 -kernel ./vmlinux
    -3.16.0-4-4kc-malta -append '"root=/dev/sda1"' -
    nographic
```

2. Připojit se na ni přes sériovou konzoli

```
socat /dev/pts/XX -
```

3. Připojit sdílené úložiště rozbalených firmware

```
mount -t 9p s0 /mnt/
```

4. Udělat lokální kopii firmware

5. provést chroot do firmware s strace

```
strace -f -o /mnt/BP/tests/strace_output/out.strace
    chroot squashfs-root /bin/sh
```

5.1 Spouštění

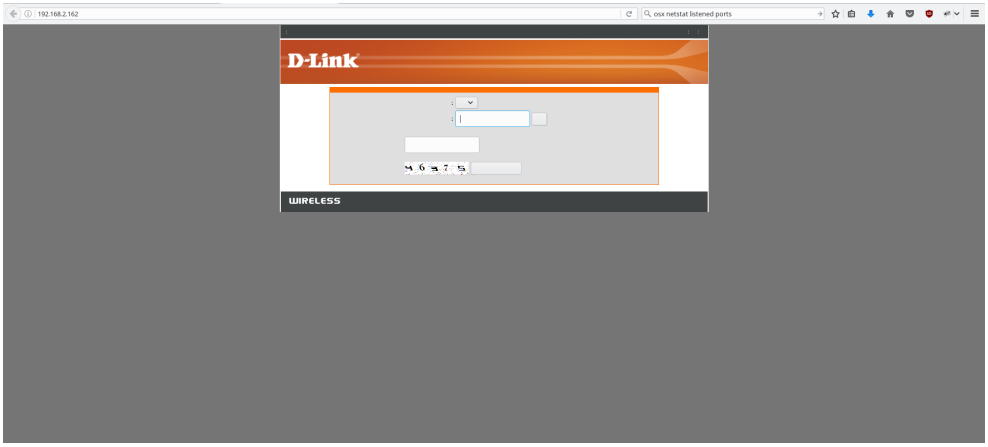
Při spuštění lighttp jsem zjistil, že lighttpd potřebuje zadat cestu ke konfiguračnímu souboru. Z výstupu výstupu v 4.2.1.3 se jako první kandidát jeví `/etc/lighttpd.conf`, což je ale symbolický odkaz. Jako druhý kandidát je `lighttpd_base.conf`. Nakopíruji tedy `lighttpd_base.conf` na místo cíle symbolického odkazu `/etc/lighttpd.conf`.

Druhý pokud o spuštění se též nezdařil:

```
/sbin/lighttpd -f /etc/lighttpd.conf
opening configfile /etc/conf.d/auth.conf failed: No
   such file or directory
```

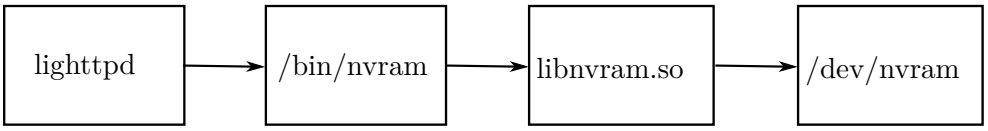
Pomocí příkazů **find** a **ls** jsem zjistil, že auth.conf je symbolický odkaz do `/tmp/lighttpd/auth.conf` a že v `/etc/conf.d` je soubor `auth_base.conf`. Tento soubor umístím na místo určené odkazem. Nakonec chybí lighttpd adresář pro logy (`/var/log/lighttpd`).

Po vytvoření adresáře pro logy se konečně lighttpd spustí. Webový server zobrazí přihlašovací okno, ale bez žádného textu. (Obr 5.1) Vzhledem k nepří-



Obrázek 5.1: Úvodní stránka lighttpd bez textu

tomnosti textu předpokládám, že chybí informace o jazyku. Zkusíme zjistit, co lighttp volá za systémová volání a další programy.



Obrázek 5.2: Cesta volání nvram

5.2 Systémová volání

Díky strace máme možnost vidět jaká systémová volání lighttpd dělal. Zejména nás zajímají soubory ke kterým přistoupil, které nenašel a které další programy spouštěl.

```
./strace_analyser -e ProcessFiles /Practical/out.strace
```

Z výstupu lze vyčíst, že se lighttpd snaží spustit `/bin/nvram`, který se snaží otevřít `/dev/nvram`, kterýžto neexistuje. `/bin/nvram` hledá následující knihovny:

```
$ LD_LIBRARY_PATH=/root/_DHP1565A1_FW101B10.bin.
    extracted/squashfs-root/lib/ ldd nvram
        libnvram.so => /root/_DHP1565A1_FW101B10.bin.
            extracted/squashfs-root/lib/libnvram.so (0
            x77e6f000)
        libgcc_s.so.1 => /root/_DHP1565A1_FW101B10.bin.
            extracted/squashfs-root/lib/libgcc_s.so.1 (0
            x77e34000)
        libc.so.0 => /root/_DHP1565A1_FW101B10.bin.
            extracted/squashfs-root/lib/libc.so.0 (0
            x77dc0000)
        ld-uClibc.so.0 => /root/_DHP1565A1_FW101B10.bin.
            extracted/squashfs-root/lib/ld-uClibc.so.0 (0
            x77daa000)
```

Součástí systému Firmadyne[10] je *libnvram*, kterým je nahrazen původní. Pro naše potřeby jsme ho ale nedokázali použít, a museli si napsat vlastní.

Známe tím ale jakým způsobem lighttpd načítá nastavení routeru. (Obr 5.2)

Pomocí příkazu **nm** nalezneme symboly obsažené v knihovně libnvram.

```
$ nm -D
00001bf0 T foreach_nvram_from
000017c0 T foreach_nvram_from_mtd
000019d8 T init_nvram_tuples
00001e10 T mac_plus_one
00000e5c T nvram_commit
000020cc T nvram_compatible_args
00001948 T nvram_config2file
00001598 T nvram_eraseall
0000123c T nvram_get
00001130 T nvram_getall
00001374 T nvram_get_nvramspace
00000d30 T nvram_init
000126e0 D nvram_reserve
00001f50 T nvram_restore_default
00001110 T nvram_set
000010f0 T nvram_unset
0000202c T nvram_upgrade
```

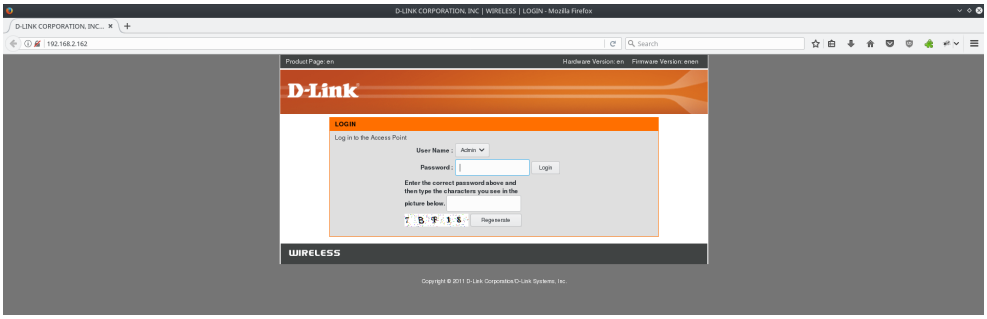
Emulace samotného hardwarového zařízení by znamenala modifikaci kernelu pomocí modulu, což obnáší riziko pádu celého systému. Pro zajištění co nejpřesnější emulace nahradíme libnvram.so. Za tímto účelem jsem využil vlastnost linkeru **ld** *LD_PRELOAD*, která umožňuje načtení a nahrazení symbolů z knihovny, předloženou uživatelem[21]. Nejprve předhodíme vlastní funkce pro **nvram_get** a **nvram_set**, jejichž signaturu najdeme v systému Firmadyne[10]. Vzhledem k tomu, že knihovnu nebudeme překládat v systému routeru, musí být přeložená staticky a umístěna do kořenového adresáře routeru.

```
gcc -shared -fPIC -nostdlib -o nvram.so nvram.c
```

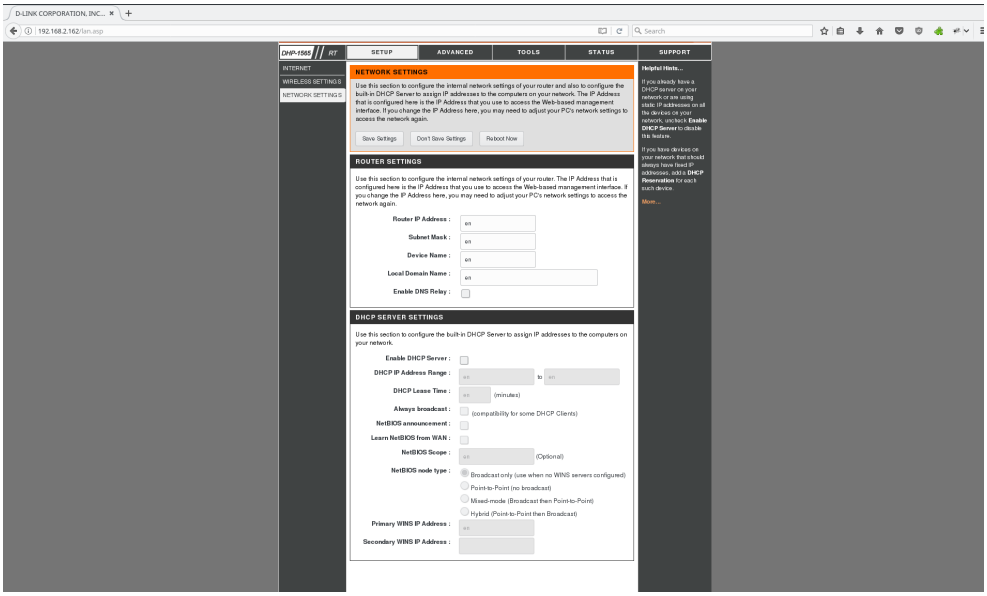
Pro ozkoušení konfigurace přes nvram zkusíme nastavit virtualizovanému systému jazyk. V `/www` existuje soubor `lang_en.js`. Na základě části "en" názvu souboru učiním pokus, a to že musí být jazyk nastaven na *en*. Pro jednoduchost testu bude první implementace funkce **nvram_get** bude vždy vracet *en*, a funkce **nvram_set** nebude dělat nic. Webový server spustím následně:

```
LD_PRELOAD="/nvram.so" /sbin/lighttpd -f /etc/lighttpd.
    conf
```

Tím dostaneme i text ve webovém rozhraní, a domněnka, že nastavení jazyka se načítá z nvram se potvrdila.



Obrázek 5.3: Plně funkční přihlašovací stránka



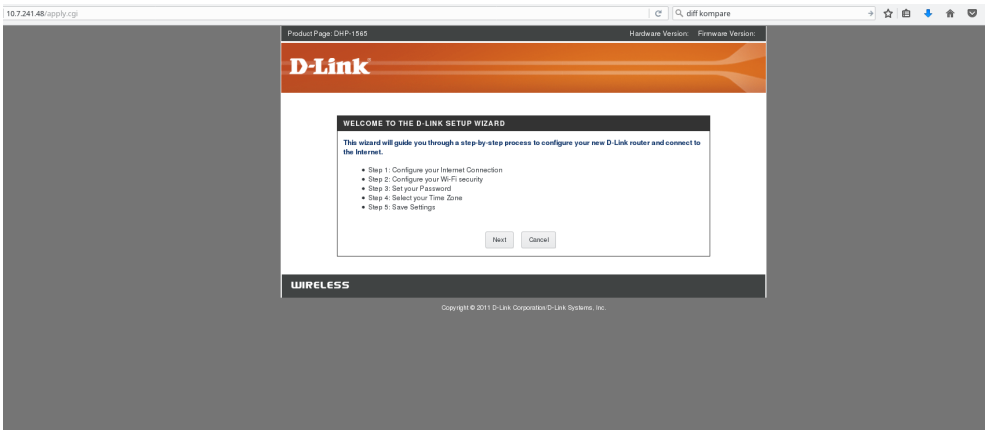
Obrázek 5.4: Funkční webové rozhraní po vrácení ”en ” z nvram

5.3 Používání lighttpd

Webové rozhraní v tuto chvíli umožňuje přihlášení bez hesla po vyplnění CAPTCHA. Chováním naši falešné nvram knihovny se všude tam, kde je obsah generovaný z aktuální konfigurace routeru se ve webovém rozhraní zobrazuje *en* (Obr 5.4).

5.3.1 Dynamický obsah

V `/etc/nvram.defaults` se nacházejí výchozí hodnoty nastavení nvram, jak ale zjistíme později, nejsou zde přítomny všechny validní hodnoty. Pomocí tohoto souboru můžeme vytvořit vlastní nvram knihovnu, která vrací poža-



Obrázek 5.5: Průvodce nastavením se zobrazí když je funkční nvram s výchozíma hodnotama

dované hodnoty. Implementaci `nvram.so`, jsem rozšířil o *nvram_set*, aby si konfigurační rozhraní mohlo hodnoty i pamatovat.

S kompletním nastavením nvram se objevil další veliký nedostatek jsme objevili v informacích o systému, jako je například uplynutý čas od zapnutí. Tyto informace udržuje jádro v systému souborů `/proc`, a stačilo tedy `/proc` připojit na správné místo:

```
$ mount -t proc none /root/_DHP1565A1_FW101B10.bin.
    extracted/squashfs-root/proc
```

5.3.2 Přihlašování

Dalším zkoumáním výstupu strace zjistíme, že při spuštění lighttpd hledá následující soubory:

- `/tmp/https.conf`
- `/tmp/ssl.users`
- `/tmp/password`

Vzhledem k tomu, že všechny naše testy probíhají přes HTTP, nejsou `/tmp/https` a `/tmp/ssl.users` zajímavé. `/tmp/password` vypadá zajímavější, a s přihlašováním ale nejspíš něco do činění má: když soubor vytvoříme, tak přestane fungovat přihlašování bez hesla. Ze zdrojových kódů přihlašovací stránky zjistíme, že z nvram získává sůl pro heslo z položky *login_salt*. I přes nastavení této položky na prázdný řetězec se přihlašování s heslem nedařilo. Nejspíš musí být heslo v nějakém specifickém formátu.

Kapitola 6

Srovnání reálného a virtuálního routeru

Tato kapitola se zabývá srovnáním nalezených zranitelností na reálném a na virtualizovaném routeru. Hned na počátku jsem zjistil, že zranitelnost popsanou v na `exploitdb.com`[15] se nedaří využít ani na jednom z webových rozhraní. Začali jsme tedy hledat další.

6.1 Zranitelnosti

Nalezené problémy nejsou přímo zneužitelné, jedná se o takzvané "information disclosure" problémy: umožní potenciálnímu útočníkovi velice jednoduše zjistit informace o prostředí a tím útok zjednodušit[22].

Jedná se o informace dostupné na webovém serveru v cestách `/chklst.txt`, `/device_status.xml` a `/HNAP1`. Všechny tři podávají bez nutnosti autentizace velice přesné informace o systému jako jsou například verze jádra, verze hardware, MAC adresy, zatížení CPU a nastavení bezdrátové sítě.

6.2 Rozdíly

6.2.1 Stahování

Rozdíly mezi reálným a virtuálním webovým serverem jsme hledali následovně:

- Z rozbaleného firmware jsme si vypsali obsah kořenu webového adresáře,
- všechny soubory jsme zkusili otevřít přes HTTP požadavek,
- srovnali jsme HTTP kódy a obsah obdržených souborů.

Jak je vidět z výstupu provedeného testu (Příloha B), HTTP status kódy jsou ve všech případech stejné. Obsah se ale liší ve většině případů. Po analýze těchto rozdílů jsme zjistili, že většina rozdílů je způsobena promněnnou nvram *graph_auth_state*, která je na virtuálním routeru prázdná a na reálném routeru má hodnotu **1**. Její nastavení na **1** skrze falešný *nvram.so* způsobilo na všech stránkách chybu 500.

Další rozdíly byly v identifikátorech HTTP session, což bysme viděli i při porovnávání dvou reálných routerů.

Souborů s většími rozdíly mnoho nebylo:

- `/wlan.txt` – na virtuálním routeru je prázdný. Což se dá vysvětlit neexistencí potřebného hardware.
- `/device_status.xml` – liší se v informacích o stroji. Jmenovitě zátěž procesoru, nastavení síťových rozhraní (rozhraní *WAN* ve virtuálním routeru vůbec neexistuje)
- `/error_404` – liší se v IP adresách.
- `/public.js` – liší se v MAC adresách, ve jménu uživatele a jedné hodnotě, která je nejspíš nastavení WPS.

6.2.2 *Pentesting* nástroje

Jako další nástroje jsme použili penetrační nástroje *Nessus*[24] a *nikto*[25]. Jedná se o obecně známé nástroje které patří mezi základní nástroje testování síťových zranitelností (*Nikto* je součást výbavy Linuxové distribuce Kali[26].

6.2.2.1 Nessus

Rozdíly ve výsledcích skenu pomocí nástroje *Nessus* (příloha D) se omezily na jeden informační informační test s identifikátorem 24260[27], který se táže webového serveru na jeho vlastnosti. Virtualizovaný webový server tyto informace nevrací.

6.2.2.2 Nikto

Výstup *Nikto* je zajímavější v tom, že poukazuje na možnost falešně pozitivních výsledků. Na virtualizovaném serveru nachází *Nikto*:

```
OSVDB-4314: /texis.exe/?-dump: Texis installation may
    reveal sensitive information.
```

[24] `https://www.tenable.com/products/nessus-vulnerability-scanner`
[25] `https://cirt.net/Nikto2`
[26] `https://www.kali.org/`
[27] `https://www.tenable.com/plugins/index.php?view=single&id=24260`

Takový výsledek je ale nesmyslný, protože soubor s tímto názvem ve firmware neexistuje. Teorie, která by vysvětlila i druhou potenciální chybu (existence víc jak jedné indexové stránky) je, že virtualizovaný router funguje trochu jinak při dotazu na neexistující stránku. Oba webové servery totiž na takový dotaz vracejí hlavní stránku, dokonce se stejnými HTTP hlavičkami. Pro kontrolu jsme učinili dotaz na `/texis.exe`, a v obou případech webový server vrátil **OK**.

Reálný:

```
HTTP/1.1 200 OK
Expires: Thu, 16 Feb 2017
    12:16:43 GMT
Cache-Control: max-age=0
Content-type: text/html
Transfer-Encoding: chunked
Date: Thu, 16 Feb 2017
    12:16:44 GMT
Server: lighttpd/1.4.28-
    devel-4927
```

Virtualizovaný:

```
HTTP/1.1 200 OK
Expires: Thu, 16 Feb 2017
    11:08:00 GMT
Cache-Control: max-age=0
Content-type: text/html
Transfer-Encoding: chunked
Date: Thu, 16 Feb 2017
    11:08:01 GMT
Server: lighttpd/1.4.28-
    devel-4927
```

Z tohoto krátkého seznamu rozdílů je vidět, že oba webové servery si jsou velice podobné. Virtualizovaný webový server nejspíš bude užitečný jako model pro hledání zranitelností, přesto je obezřetnost vůči falešně pozitivním výsledkům na místě.

Kapitola 7

Automatizace

V předchozích částech jsme si ukázali, jak spustit webový server jednoho modelu routeru. Jedná se o relativně náročnou a zdlouhavou operaci, a nabízí se otázka, zda si nebudou některé modely routerů natolik podobné, že by šlo celý proces automatizovat. V této části se pokusím přiblížit požadavky pro automatizaci a její možné řešení.

Jak jsme si ukázali v 4.2, základem pro spuštění virtuálního stroje jsou informace jako:

- Architektura (včetně endianity)
- Odhad kde je webový server a jeho nastavení
- Způsob ukládání a načítání konfigurace
- Obraz firmware
- Rozložení souborového systému

Tyto informace jsou ve své podstatě statická data, a nabízí se je uložit do databáze.

Dále je potřeba systém na správu virtuálních strojů, který umí spravovat snapshoty. Vzhledem k výběru Qemu, se nám možnosti v omezily na libvirt[28].

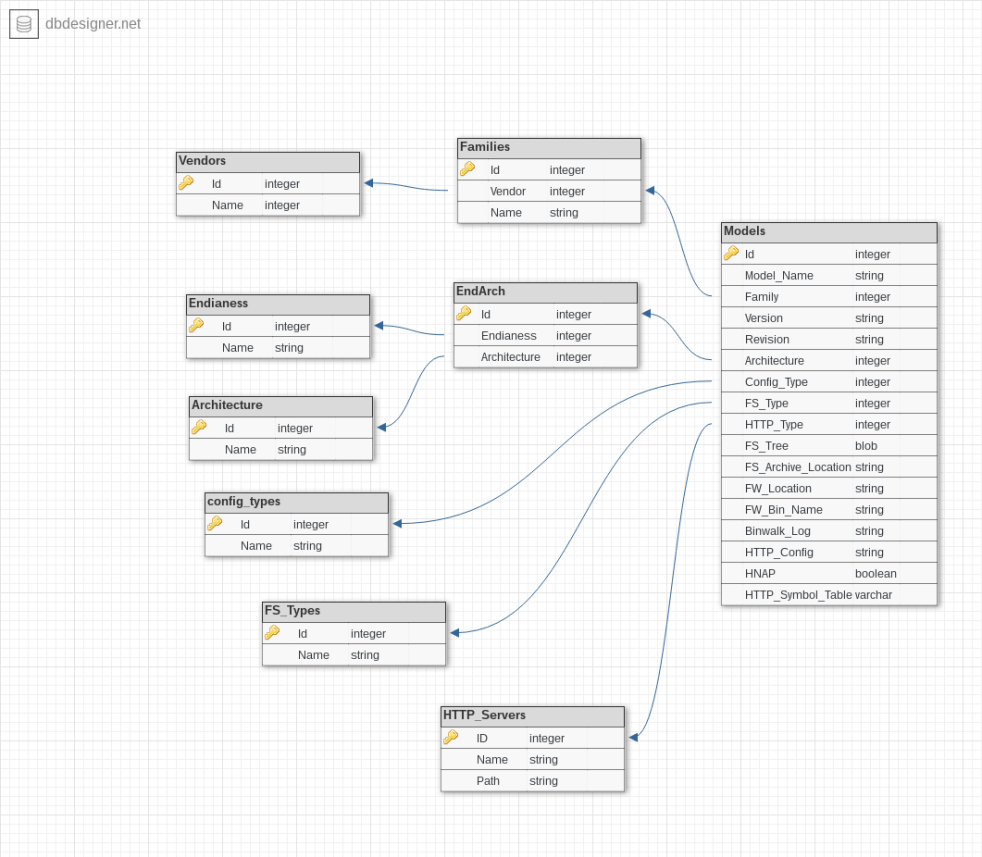
Konečný funkční systém by si tak podle žádaného modelu a služby našel informace v databázi, připravil správný virtuální stroj, pomocí různých pravidel a heuristik by spustil konkrétní službu a nástroj na hledání zranitelností.

7.1 Databáze routerů

K udržování informací o architektuře, výrobci, modelu apod. je vhodná relační databáze. Databázi postavím nad relačním strojem Postgresql[29], verze 9.6.

[28] `http://libvirt.org/`

[29] `https://www.postgresql.org/`



Obrázek 7.1: Schéma databáze routerů

Skripty na stažení firmware a plnění databáze napsal kolega Bc. Libor Bakajsa, moje práce je návrh databáze a použít ji pro získání zajímavých dat.

Databáze by měla převážně sloužit jako zdroj dat pro virtualizační automat ale může sloužit i při tvoření heuristických detekcí.

7.1.1 Vyhledávání modelů podle souborů

Soubory obsažené ve firmware nám poskytují značné množství informací o konstrukci firmware daného routeru. (Viz kapitola 4.2.1.2) Například pokud je zranitelnost závislá na existenci konkrétního souboru ve webovém adresáři, lze všechny modely s tímto souborem jednoduše najít a případně automaticky ozkoušet.

Souborový systém je v databázi uložen ve formátu JSON. Postgresql umožňuje s tímto formátem pracovat přímo z jazyka SQL. Tato vlastnost Postgresql je velice užitečná pro vyhledávání na základě souborů existujících ve firmware.

Je užitečné si vytvořit výčet typů souborů, které známe

```
CREATE TYPE md AS enum ('link','file','directory','char
    ','block','unknown');
```

Následně typy specifikované v **JSON** popisu systému souborů převedeme na typy z předem zavedeného výčtu.

```
CREATE OR REPLACE FUNCTION mode_to_enum(m text)
RETURNS md AS
$func$
BEGIN
        CASE m WHEN 'directory' THEN  RETURN 'directory
            '::md;
                WHEN 'link' THEN RETURN 'link'::md;
                WHEN 'file' THEN RETURN 'file'::md;
                WHEN 'char_device' THEN RETURN 'char'::
                    md;
                WHEN 'block_device' THEN RETURN 'block
                    '::md;
                ELSE RETURN 'unknown'::md;
        END CASE;
END
$func$ LANGUAGE plpgsql;
```

A nakonec vytvoříme pomocnou funkci na prohledávání souborů. Protože je tato funkce je komplikovanější a specifická pro postgresql, zaslouží si delší komentář[30]:

```
CREATE OR REPLACE FUNCTION search_file_in_tree(u jsonb)
RETURNS TABLE (path text, filename text, type md)
AS
$$
WITH RECURSIVE flatfiles(path, filename,type,json) as (
        SELECT '/'::text,j->>'name',mode_to_enum(j->>'
            mode'),j  from (SELECT jsonb_array_elements(u
            ->'content') as j) as b
```

Nejprve je potřeba vytvořit SQL funkci **search_file_in_tree** vracející tabulku o třech sloupcích, ve které definujeme CTE[31] jménem **flatfiles** s vlastností *RECURSIVE*, což funkci umožní znovu využívat svůj vlastní výstup. Pomocí příkazu *SELECT* vybereme počáteční data.

[30] operátory specifické pro práci s formátem JSON v SQL zde nepopisuji, protože jsou popsané v dokumentaci postgresql `https://www.postgresql.org/docs/current/static/functions-json.html`

[31] Common Table Expression

```
        UNION ALL
        select CASE a.path WHEN '/' THEN '/'::text || a.
            filename ELSE a.path || '/' || a.filename END
            , j->>'name', mode_to_enum(j->>'mode'),j from
             (select path,filename,jsonb_array_elements(
            json->'content') as j from flatfiles where
            type = 'directory'::md) as a
)
SELECT  path,filename,type FROM flatfiles;
$$
LANGUAGE sql;
```

Direktivou **UNION ALL** databázovému stroji specifikujeme, že má sloučit původní data s novými, a že má ve výsledné tabulce ponechat duplicitní záznamy. Následně do výsledků přidá všechny soubory a adresáře z již nalezených adresářů. A tak pokračuje dokud má co přidávat. Nakonec funkce vrátí celou vytvořenou tabulku.[32]

Samotné hledání lze provést dotazem

```
select id,model_name,fileTree.path,fileTree.filename,
    fileTree.type from fw_table_new t,
    search_file_in_tree(t.fs_tree) fileTree WHERE
    fileTree.filename ~ '.*nvram.*';
```

Operátor ~ je srovnávání řetězce s regulárním výrazem. Data z funkce **search_file_in_tree** se objeví ve výsledku pomocí LATERAL JOIN[33].

7.2 libvirt

Libvirt je svobodná knihovna na správu virtuálních strojů podporující různé virtualizační programy (mezi nimiž jsou například Qemu, VMWare, Xen, Virtualbox, ...), za kterou převážně stojí firma Redhat. Slouží jako základ pro cloudovou platformu openstack, používá ji CI systém Jenkins a mnoho dalších[34]. Mezi podporovanou funkcionalitu, kterou využijeme patří:

- Správa diskových obrazů
- Snapshoty
- Podpora qcow2 formátu
- Správa sítě

[32] Ve zdrojových kódech je okomentovaná varianta této funkce.

[33] https://www.postgresql.org/docs/9.3/static/sql-select.html#SQL-FROM

[34] https://libvirt.org/apps.html

- Deklarace virtuálních strojů a jejich změny přes volání API

Knihovna je psána v jazyce C, s automaticky generovaným API pro Python. (verze 2 i 3) Rozhodl jsem se použít libvirt pro systém na správu virtuálních strojů.

7.2.1 Správa Disků

Libvirt umí spravovat obrazy disků, a jejich snapshotů. Umožňuje použít QCow2 koncept *backing store*[35] přímo pomocí generování XML definice disku. Postup pro využití *libvirt* pro správu disků je následující:

- Připravit počáteční disk
- Umístit počáteční disk do *storage pool* libvirt
- Najít počáteční disk přes API libvirt
- Vygenerovat XML pro nový disk
- Spustit virtuál s novým diskem

Výsledná XML definice disku:

```
<volume>
<name>debian-mips</name>
<target>
    <path>{VOL_PATH}</path>
    <format type='qcow2'/>
</target>
<backingStore>
    <path>{BACKING_PATH}</path>
    <format type='qcow2'/>
</backingStore>
</volume>
```

Podobným způsobem lze pomocí XML a libvirt spravovat verze jádra pro architektury, které potřebují specifikovat kernel externě (například ARM či MIPS).

7.2.2 Deklarace virtuálního stroje

Zde se zabývám tím, jak udělat XML specifikaci pro samotný stroj. Je potřeba specifikovat

- Architekturu

[35] Popsaný v části 4.3.2

- Typ procesoru
- Disk
- Umístění jádra (pokud to architektura vyžaduje)
- Parametry jádra (pokud to architektura vyžaduje)

7.2.2.1 Virtuální stroj

Tím, že libvirt podporuje několik virtualizačních systémů, je potřeba explicitně specifikovat, který má použít. V případě Qemu se v libvirt systém jmenuje *hvm*. V případě architektury s externě specifikovaným jádrem je potřeba ve správci svazků najít přesnou cestu požadovaného jádra, tuto zadat do specifikace virtuálního stroje. Nakonec je potřeba specifikovat parametry pro jádro. (Například výchozí sériová linka) Výsledná část XML vypadá následovně:

```
<os>
  <type arch='mips' machine='malta'>hvm</type>
  <kernel>/var/lib/libvirt/kernels/vmlinux-3.16.0-4-4
      kc-malta</kernel>
  <cmdline>root=/dev/sda1 console=/dev/ttyS0</cmdline>
</os>
```

7.2.2.2 Disk

Disk vytvořený pro tento virtuální stroj je potřeba přesně specifikovat v definici virtuálního stroje. Je potřeba název disku, jeho umístění, přes jaké rozhraní je disk připojen (SATA/IDE), a jak je daný řadič připojen do virtuálního stroje. Výsledné XML:

```
    <disk type='volume' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source pool='default' volume='test-01'/>
      <target dev='hda' bus='ide'/>
      <alias name='disk0'/>
      <address type='drive' controller='0' bus='0' unit
          ='0'/>
    </disk>
```

7.2.3 Sériová linka

Sériovou linku je třeba definovat ve dvou fázích, podobně jako se to dělá pomocí přepínačů Qemu. Jednak se pomocí XML tagu *serial* musí definovat

samotná sériová linka, a jednak se musí pomocí XML tagů *channel* a *serial* definovat jakým způsobem se připojí do hostitelského stroje..

```
<channel type='pty'>
    <target type='virtio' />
</channel>
<serial type='pty'>
    <target port='0' />
</serial>
```

Z XML popisující běžící stroj se následně musí vyčíst kde sídlí sériová linka v hostitelském stroji.

7.2.4 Specificity architektury MIPS

Pro účel této práce je potřeba aby se mohly připravit virtuální stroje téměř jakékoliv architektury. Prvotní testy s libvirt jsem dělal na architektuře x86 s předpokladem, že změna architektury je jenom změna jednoho parametru v XML. Po provedení změny architektury se ale virtuální stroj odmítal zapnout s chybovou hláškou

```
error: XML error: No PCI buses available
```

Po rozsáhlé analýze problému jsem zjistil, že pro architekturu MIPS libvirt nespecifikuje virtuální stroj s PCI řadičem a že je potřeba ho specifikovat explicitně.[36] Problém vyřešilo přidání definice PCI řadiče přímo do XML.

```
<controller type='pci' index='0' model='pci-root'>
  <alias name='pci.0'/>
</controller>
```

Tím se problém sice vyřešil, ale vznikla potřeba specifikovat PCI řadič na kterém sídlí diskový řadič. To zavedlo další komplexnost do generace definujícího XML, a to v závislosti IDE řadiče na PCI řadiči. Při analýze jsem také zjistil, že libvirt staví celý virtuální stroj přes Qemu argumenty a že to zřejmě není zcela funkční pro všechny architektury. Zároveň jsem zjistil, že si Qemu umí sám potřebná zařízení vytvořit, pokud mu to není zakázáno pomocí přepínače

```
-nodefaults
```

což libvirt činí.

Vzhledem k problémům na architektuře MIPS se dají předpokládat problémy i na dalších architekturách. Ve spojení s potíží aplikovat dokumentaci na

[36] `https://www.redhat.com/archives/libvir-list/2016-May/msg00588.html`

API v jazyce Python jsme se rozhodli řešení pomocí libvirt nakonec nepoužít, a zkusit vlastní řešení.

7.3 Vlastní řešení

Pro celkovou automatizaci celého procesu jsou potřeba 3 části:

- Spuštění virtuálního stroje
- Ovládání virtuálního stroje
- Vyhodnocování stavu virtuálního stroje a rozhodování o dalším postupu.

Implementace je postačující pro spuštění virtuálního stroje vhodného k realizaci postupu popsaného v této práci. Soubor `shim.sh` spustí kontrolující skript se správnými parametry.

7.3.1 Správa virtuálního stroje

Tato část napsaná v jazyce *bash* se stará o vytvoření snapshotu disku a o programatické generování argumentů pro Qemu. Aktuálně vygeneruje argumenty pro Qemu pro připojení disku, sdílení adresáře a siťová zařízení. Dále spustí virtuální stroj a spustí část na ovládání virtuálního stroje, která je napsaná v jazyce Python. Skript nakonec čeká až všechno skončí.

7.3.2 Ovládání virtuálniho stroje

Ovládání virtuálního stroje je napsané v jazyce Python. Hlavním důvodem je existující implementace protokolu QMP. Přes protokol QMP najde sériovou linku, a připojí se. K samotnému ovládání stroje je použita knihovna pexpect[37], která umí otevřít soubor zařízení pseudoterminálu, a provádět na něm I/O operace. Tím může na základě výstupu virtuálního stroje posílat různé príkazy. V tuto chvíli se umí přihlásit a připojit sdílené adresáře.

7.3.3 Další nutný vývoj

Celkově je toto řešení ve fázi ověřování zda přístup může fungovat. Při pokusech se osvědčilo: virtuální stroj skripty připravily rychle a spolehlivě.

Část s pravidly je ještě potřeba navrhnout kompletně. Odvážím si tvrdit, že na to statická pravidla stačit nebudou, a že bude vhodné použít kombinaci statických pravidel a nějaký typ strojového učení.

[37] `https://pexpect.readthedocs.io/en/stable/`

Závěr

Cíl práce spustit webový server z routeru ve virtuálním prostředí byl dosažen. Současně se prozkoumalo několik slepých uliček, potvrdit předpoklady, a diskutovat o možnosti automatizace procesu.

Webový server routeru DHP-1565, běžící ve virtuálním prostředí, je funkční do té míry, že webové rozhraní lze používat nejen manuálně, ale i pro automatické detekce. Experiment na tomto konkrétním příkladu ukázal, jak spustit webový server routeru mimo své nativní prostředí a vytvořil základní návod pro použití podobného přístupu na další modely.

V laboratorních podmínkách se nepodařilo údajně známé zranitelnosti routeru DHP-1565 zneužít ani na reálném, ani na virtualizovaném webovém serveru. Detekované vlastnosti virtualizovaného webového serveru byly totožné se serverem reálného routeru, což informacím obdržených z virtualizovaného serveru dává veliký potenciál. Pro potvrzení úspěšnosti tohoto přístupu by bylo potřeba experiment s virtualizací služby opakovat na jiným modelu, případně i jiné službě. (například upnp) a také porovnat s fyzickým zařízením.

Při potvrzení předpokladů o podobnosti mezi jednotlivými modely lze očekávat, že automatický systém na hledání zranitelností umožní odhalování zranitelností a získání většího souboru srovnávacích dat.

Využití *libvirt* pro automatizaci správy virtuálních strojů se neosvědčilo. Vlastní řešení přináší základ, na kterém se dá budovat zmíněná automatizace. Vzhledem k tomu, že jsem nakonec dospěl se svým základem k řešení podobnému tomu, které používají pro systém Firmadyne[10], by asi bylo záhodné zkusit tuto část systému převzít.

Vedlejším výsledkem této práce bylo zjištění, které informace jsou potřeba pro automatizaci, vytvoření databáze routerů a základních nástrojů pro její použití. Data obsažené v databázi jsou rozsáhlá, ale s jejich rozšířením se do budoucna samozřejmě počítá.

Literatura

[1] Nordrum, A.: Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated [online]. Srpen 2016, [cit. 2017-02-10]. Dostupné z: http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[2] Craig, S.: Beware of older cyber attacks [online]. Leden 2016, [cit. 2017-02-10]. Dostupné z: http://safewayconsultoria.com/wp-content/uploads/2016/05/Beware-of-older-cyber-attacks_2016-1.pdf

[3] Ragan, S.: Here are the 61 passwords that powered the Mirai IoT botnet [online]. Říjen 2016, [cit. 2017-02-10]. Dostupné z: http://www.csoonline.com/article/3126924/security/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html

[4] Paganini, P.: The NSA wants to exploit IoT devices for surveillance and sabotage [online]. Červen 2016, [cit. 2017-02-10]. Dostupné z: http://securityaffairs.co/wordpress/48319/iot/nsa-iot-devices.html

[5] McLaughlin, J.: NSA Looking to Exploit Internet of Things, Including Biomedical Devices, Official Says [online]. Červen, [cit. 2017-02-15].

[6] Ullrich, J. B.: Port 7547 SOAP Remote Code Execution Attack Against DSL Modems [online]. Listopad 2016, [cit. 2017-02-10]. Dostupné z: https://isc.sans.edu/forums/diary/Port+7547+SOAP+Remote+Code+Execution+Attack+Against+DSL+Modems/21759

[7] Biggs, J.: Hackers release source code for a powerful DDoS app called Mirai [online]. Říjen 2016, [cit. 2017-02-10]. Dostupné z: https://techcrunch.com/2016/10/10/hackers-release-source-code-for-a-powerful-ddos-app-called-mirai/

[8] Townsend, K.: New Leet Botnet Shows IoT Device Security Regulation May Become Necessary [online]. Prosinec 2016, [cit. 2017-02-10]. Dostupné z: http://www.securityweek.com/massive-attack-new-leet-botnet-reaches-650-gbps

[9] Arntz, P.: DNS Hijacks: Routers [online]. Prosinec 2015, [cit. 2017-02-10]. Dostupné z: https://blog.malwarebytes.com/cybercrime/2015/12/dns-hijacks-routers/

[10] Chen, D. D.; Egele, M.; Woo, M.; aj.: Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. Únor 2016.

[11] Shick, S.: Linux.Wifatch: The Router Virus That May Be Secretly Defending You From Other Malware [online]. Říjen 2015, [cit. 2017-02-10]. Dostupné z: https://securityintelligence.com/news/linux-wifatch-the-router-virus-that-may-be-secretly-defending-you-from-other-malware/

[12] Comparison of platform virtualization software [online]. Únor 2017, [cit. 2017-02-10]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software

[13] Simons, G.: QEMU User Emulation [online]. Říjen 2013, [cit. 2017-02-10]. Dostupné z: https://wiki.debian.org/QemuUserEmulation

[14] Qemu: User mode emulation and Full system emulation [online]. Duben 2015, [cit. 2017-02-10]. Dostupné z: http://www.cnblogs.com/pengdonglin137/p/5020143.html

[15] Lovett, K.: Multiple D-Link Routers - Multiple Vulnerabilities [online]. Květen 2014, [cit. 2017-02-10]. Dostupné z: https://www.exploit-db.com/exploits/33520/

[16] Craig: Mucking About With SquashFS [online]. Srpen 2014, [cit. 2017-02-10]. Dostupné z: http://www.devttys0.com/2014/08/mucking-about-with-squashfs/

[17] Nguyen, B.: Linux Filesystem Hierarchy [online]. Červenec 2004, [cit. 2017-02-10]. Dostupné z: http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/

[18] ldd - print shared object dependencies. Červenec 2016, [cit. 2017-02-10].

[19] Granlund, T.: System emulation using QEMU [online]. Únor 2017, [cit. 2017-02-10]. Dostupné z: https://gmplib.org/~tege/qemu.html

[20] Garimella, N.: Snapshot technology overview [online]. Duben 2006, [cit. 2017-02-10]. Dostupné z: http://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html

[21] ld - The GNU linker. Červen 2016.

[22] Yadav, A.: Exploiting by Information Disclosure[online]. Březen 2014, [cit. 2017-02-15]. Dostupné z: `http://resources.infosecinstitute.com/exploiting-information-disclosure-part-1/`

Příloha **A**

Seznam použitých zkratek

SOHO Small Office/Home Office

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

IoT Internet of Things

XML Extensible markup language

GPL GNU General Public License

LZMA Lempel–Ziv–Markov chain algorithm

DNS Domain Name System

MIPS Microprocessor without Interlocked Pipeline Stages

SSI Server Side Includes

qcow QEMU Copy On Write

SNMP Simple Network Management Protocol

NSA National Security Agency

eBPF extended Berkeley Packet Filter

QMP Qemu Machine Protocol

ARM Advanced RISC Machine

PLC Powerline communication

CI Continuous Integration

HTTP Hypertext tranfer protocol

HNAP Home Network Administration Protocol

Příloha B

Výstup diff

```
Files global.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    global.h are identical
5c5,11
< CFLAGS := -I$(INCLUDEDIR) -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -
    D_GNU_SOURCE -O2
---
> CFLAGS := -I$(INCLUDEDIR)  -I../lzmasdk/C -D_FILE_OFFSET_BITS=64 -
    D_LARGEFILE_SOURCE -D_GNU_SOURCE -O2
> USE_LZMA = 1
> ifdef USE_LZMA
>   LZMA_CFLAGS = -DUSE_LZMA
>   LZMA_LIB = -L../lzmasdk/C/LzmaUtil -llzma
>   CFLAGS += $(LZMA_CFLAGS)
> endif
9,10c15,16
< mksquashfs: mksquashfs.o read_fs.o sort.o swap.o pseudo.o
<       $(CC) mksquashfs.o read_fs.o sort.o swap.o pseudo.o -lz -lpthread -lm
     -o $@
---
> mksquashfs: lzmalib mksquashfs.o read_fs.o sort.o swap.o pseudo.o
    uncompress.o
>       $(CC) mksquashfs.o read_fs.o sort.o swap.o pseudo.o uncompress.o -lz
    -lpthread -lm $(LZMA_LIB) -o $@
12c18,19
< mksquashfs.o: mksquashfs.c squashfs_fs.h mksquashfs.h global.h sort.h
    squashfs_swap.h Makefile
---
> lzmalib:
>       make -C ../lzmasdk/C/LzmaUtil -f makefile.gcc
14c21,23
< read_fs.o: read_fs.c squashfs_fs.h read_fs.h global.h squashfs_swap.h
    Makefile
---
> mksquashfs.o: mksquashfs.c squashfs_fs.h mksquashfs.h global.h sort.h
    squashfs_swap.h uncompress.h Makefile
>
> read_fs.o: read_fs.c squashfs_fs.h read_fs.h global.h squashfs_swap.h
    uncompress.h Makefile
22,23c31
< unsquashfs: unsquashfs.o unsquash-1.o unsquash-2.o unsquash-3.o unsquash-4.
    o swap.o
<       $(CC) unsquashfs.o unsquash-1.o unsquash-2.o unsquash-3.o unsquash-4.
    o swap.o -lz -lpthread -lm -o $@
---
> uncompress.o: uncompress.c uncompress.h
25c33,34
< unsquashfs.o: unsquashfs.h unsquashfs.c squashfs_fs.h squashfs_swap.h
    squashfs_compat.h global.h Makefile
---
> unsquashfs: unsquashfs.o unsquash-1.o unsquash-2.o unsquash-3.o unsquash-4.
```

```
    o swap.o uncompress.o
>       $(CC) unsquashfs.o unsquash-1.o unsquash-2.o unsquash-3.o unsquash-4.
    o swap.o uncompress.o -lz -lpthread -lm $(LZMA_LIB) -o $@
27c36
< unsquash-1.o: unsquashfs.h unsquash-1.c squashfs_fs.h squashfs_compat.h
    global.h Makefile
---
> unsquashfs.o: unsquashfs.h unsquashfs.c squashfs_fs.h squashfs_swap.h
    squashfs_compat.h global.h uncompress.h Makefile
29c38
< unsquash-2.o: unsquashfs.h unsquash-2.c unsquashfs.h squashfs_fs.h
    squashfs_compat.h global.h Makefile
---
> unsquash-1.o: unsquashfs.h unsquash-1.c squashfs_fs.h squashfs_compat.h
    global.h uncompress.h Makefile
31c40
< unsquash-3.o: unsquashfs.h unsquash-3.c squashfs_fs.h squashfs_compat.h
    global.h Makefile
---
> unsquash-2.o: unsquashfs.h unsquash-2.c unsquashfs.h squashfs_fs.h
    squashfs_compat.h global.h uncompress.h Makefile
33c42,44
< unsquash-4.o: unsquashfs.h unsquash-4.c squashfs_fs.h squashfs_swap.h
    global.h Makefile
---
> unsquash-3.o: unsquashfs.h unsquash-3.c squashfs_fs.h squashfs_compat.h
    global.h uncompress.h Makefile
>
> unsquash-4.o: unsquashfs.h unsquash-4.c squashfs_fs.h squashfs_swap.h
    global.h uncompress.h Makefile
36c47,48
<       -rm -f *.o mksquashfs unsquashfs
---
>       -rm -f *.o mksquashfs unsquashfs
>       make -C ../lzmasdk/C/LzmaUtil -f makefile.gcc clean
66a67,78
> #include "uncompress.h"
>
> #ifdef USE_LZMA
> #include <LzmaEnc.h>
> #include <LzmaDec.h>
> #define LZMA_DEFAULT_LEVEL    5
> #define LZMA_DEFAULT_DICT     0
> #define LZMA_DEFAULT_LC       1
> #define LZMA_DEFAULT_LP       2
> #define LZMA_DEFAULT_PB 2
> #define LZMA_DEFAULT_FB 32
> #endif
832a845,857
> #ifdef USE_LZMA
> static void *lzma_malloc(void *p, size_t size)
> {
>       (void)p;
>       return malloc(size);
> }
> static void lzma_free(void *p, void *addr)
> {
>       (void)p;
>       free(addr);
> }
> static ISzAlloc lzma_alloc = { lzma_malloc, lzma_free };
> #endif
843a869,917
> #ifdef USE_LZMA
>       if (compression == LZMA_COMPRESSION) {
>               size_t outsize = block_size - LZMA_PROPS_SIZE;
>               size_t propsize = LZMA_PROPS_SIZE;
>               CLzmaEncProps props;
>
>               LzmaEncProps_Init(&props);
>               props.level = LZMA_DEFAULT_LEVEL;
>               props.dictSize = LZMA_DEFAULT_DICT;
>               props.lc = LZMA_DEFAULT_LC;
>               props.lp = LZMA_DEFAULT_LP;
```

```
>                 props.pb = LZMA_DEFAULT_PB;
>                 props.fb = LZMA_DEFAULT_FB;
>                 props.numThreads = 1;
>
>                 res = LzmaEncode((unsigned char *) d + LZMA_PROPS_SIZE, &
    outsize,
>                         (unsigned char *) s, size,
>                         &props, (unsigned char *) d, &propsize,
>                         1, NULL, &lzma_alloc, &lzma_alloc);
>                 switch(res) {
>                 case SZ_OK:
>                         outsize += LZMA_PROPS_SIZE;
>                         break;
>                 case SZ_ERROR_DATA:
>                         BAD_ERROR("lzma::compress failed, data error\n");
>                         break;
>                 case SZ_ERROR_MEM:
>                         BAD_ERROR("lzma::compress failed, memory allocation
    error\n");
>                         break;
>                 case SZ_ERROR_PARAM:
>                         BAD_ERROR("lzma::compress failed, invalid parameters\
    n");
>                         break;
>         case SZ_ERROR_OUTPUT_EOF:
>             memcpy(d, s, size);
>             return size | (data_block ? SQUASHFS_COMPRESSED_BIT_BLOCK :
>                     SQUASHFS_COMPRESSED_BIT);
>         case SZ_ERROR_THREAD:
>                         BAD_ERROR("lzma::compress failed, errors in
    multithreading functions\n");
>                         break;
>                 /* should not happen */
>                 default:
>                         BAD_ERROR("lzma::compress failed, unknown error\n");
>                         break;
>                 }
>
>                 return outsize;
>         }
> #endif
>
1672c1746
<                 res = uncompress((unsigned char *) buffer->data, &bytes,
---
>                 res = uncompress_wrapper((unsigned char *) buffer->data, &
    bytes,
1676c1750
<                                 BAD_ERROR("zlib::uncompress failed, not
    enough "
---
>                                 BAD_ERROR("uncompress failed, not enough "
1679c1753
<                                 BAD_ERROR("zlib::uncompress failed, not
    enough "
---
>                                 BAD_ERROR("uncompress failed, not enough "
1682c1756
<                                 BAD_ERROR("zlib::uncompress failed,"
---
>                                 BAD_ERROR("uncompress failed,"
4162a4237,4238
>         compression=LZMA_COMPRESSION;
>
4284a4361,4364
> #ifdef USE_LZMA
>                 } else if(strcmp(argv[i], "-nolzma") == 0) {
>                         compression = ZLIB_COMPRESSION;
> #endif
4412a4493,4495
> #ifdef USE_LZMA
>                         ERROR("-lzma Enable LZMA compression\n");
> #endif
4494c4577
```

```
<                                   S_IRWXU);
---
>                                   S_IRUSR |S_IWUSR|S_IRGRP|S_IROTH);
4807c4890
<       sBlk.compression = ZLIB_COMPRESSION;
---
>       sBlk.compression = compression;
43a44,48
>
> extern int uncompress_wrapper(unsigned char *dest, unsigned long *dest_len,
>     const unsigned char *src, unsigned long src_len);
>
>
Files pseudo.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    pseudo.c are identical
Files pseudo.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    pseudo.h are identical
53a54
> #include "uncompress.h"
86,87c87,88
<               res = uncompress(block, &bytes, (const unsigned char *)
    buffer,
<                       c_byte);
---
>               res = uncompress_wrapper(block, &bytes,
>                       (const unsigned char *) buffer, c_byte);
90c91
<                               ERROR("zlib::uncompress failed, not enough "
---
>                               ERROR("uncompress failed, not enough "
93c94
<                               ERROR("zlib::uncompress failed, not enough "
---
>                               ERROR("uncompress failed, not enough "
96c97
<                               ERROR("zlib::uncompress failed, unknown error
    "
---
>                               ERROR("uncompress failed, unknown error "
Files read_fs.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    read_fs.h are identical
Files sort.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/sort.c
     are identical
Files sort.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/sort.h
     are identical
Files squashfs_compat.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-
    tools/squashfs_compat.h are identical
231a232
> #define LZMA_COMPRESSION       2
Files squashfs_swap.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-
    tools/squashfs_swap.h are identical
Files swap.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/swap.c
     are identical
Files unsquash-1.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    unsquash-1.c are identical
Files unsquash-2.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    unsquash-2.c are identical
Files unsquash-3.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    unsquash-3.c are identical
Files unsquash-4.c and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    unsquash-4.c are identical
26a27
> #include "uncompress.h"
600c601
<               res = uncompress((unsigned char *) block, &bytes,
---
>               res = uncompress_wrapper((unsigned char *) block, &bytes,
602d602
<
605c605
<                               ERROR("zlib::uncompress failed, not enough "
---
>                               ERROR("uncompress failed, not enough "
608c608
```

```
<                                ERROR("zlib::uncompress failed, not enough "
---
>                                ERROR("uncompress failed, not enough "
611c611
<                                ERROR("zlib::uncompress failed, unknown error
     "
---
>                                ERROR("uncompress failed, unknown error "
648c648
<                res = uncompress((unsigned char *) block, &bytes,
---
>                res = uncompress_wrapper((unsigned char *) block, &bytes,
650d649
<
653c652
<                                ERROR("zlib::uncompress failed, not enough "
---
>                                ERROR("uncompress failed, not enough "
656c655
<                                ERROR("zlib::uncompress failed, not enough "
---
>                                ERROR("uncompress failed, not enough "
659c658
<                                ERROR("zlib::uncompress failed, unknown error
     "
---
>                                ERROR("uncompress failed, unknown error "
1462c1461
<                return TRUE;
---
>                goto done;
1550a1550,1552
> done:
>        compression = sBlk.compression;
>
1713c1715
<                res = uncompress((unsigned char *) tmp, &bytes,
---
>                res = uncompress_wrapper((unsigned char *) tmp, &bytes,
1719c1721
<                                ERROR("zlib::uncompress failed, not enough"
---
>                                ERROR("uncompress failed, not enough"
1722c1724
<                                ERROR("zlib::uncompress failed, not enough "
---
>                                ERROR("uncompress failed, not enough "
1725c1727
<                                ERROR("zlib::uncompress failed, unknown error
     "
---
>                                ERROR("uncompress failed, unknown error "
Files unsquashfs.h and ../../../src/AthSDK/tools/squashfs-4.0/squashfs-tools/
    unsquashfs.h are identical
```

Příloha C

Výstup nikto

C.1 Virtualizovaný router

```
root@kali2:~# nikto -h 10.7.241.48
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.7.241.48
+ Target Hostname:    10.7.241.48
+ Target Port:        80
+ Start Time:         2017-02-04 13:57:05 (GMT-5)
---------------------------------------------------------------------------
+ Server: lighttpd/1.4.28-devel-4927
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the
    user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user
    agent to render the content of the site in a different fashion to the
    MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ lighttpd/1.4.28 devel appears to be outdated (current is at least 2.0.0)
+ Multiple index files found (note, these may not all be unique):
/index.do, /index.jhtml, /index.cfm, /index.asp, /index.aspx, /index.xml,
/default.asp, /default.htm, /index.pl, /index.php, /index.html,
/index.shtml, /default.aspx, /index.php3, /index.htm, /index.cgi
+ OSVDB-1193: /cfdocs.map: Cold Fusion CFCACHE tag places temporary cache
    files within the web document root, allowing remote attackers to obtain
    sensitive system information. http://cve.mitre.org/cgi-bin/cvename.cgi?
    name=CVE-2000-0057.
+ OSVDB-4314: /texis.exe/?-dump: Texis installation may reveal sensitive
    information.
+ 7557 requests: 12 error(s) and 7 item(s) reported on remote host
+ End Time:           2017-02-04 15:21:15 (GMT-5) (5050 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

C.2 Reálný router

```
 root@kali2:~# nikto -h 192.168.0.1
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.0.1
+ Target Hostname:    192.168.0.1
+ Target Port:        80
+ Start Time:         2017-02-04 13:35:43 (GMT-5)
---------------------------------------------------------------------------
+ Server: lighttpd/1.4.28-devel-4927
+ The anti-clickjacking X-Frame-Options header is not present.
```

```
+ The X-XSS-Protection header is not defined. This header can hint to theuser
     agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the
    useragent to render the content of the site in a different fashion to
    the MIMEtype
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ lighttpd/1.4.28 devel appears to be outdated (current is at least 2.0.0)
+ OSVDB-1193: /cfdocs.map: Cold Fusion CFCACHE tag places temporary
    cachefiles within the web document root, allowing remote attackers to
    obtainsensitive system information.http://cve.mitre.org/cgi-bin/cvename.
    cgi?name=CVE-2000-0057.
+ 7540 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2017-02-04 13:43:17 (GMT-5) (454 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Příloha D

Výsledky Nessus

D.1 Virtualizovaný router

Vulnerabilities By Plugin

- 85582 (1) - Web Application Potentially Vulnerable to Clickjacking
- 26194 (1) - Web Server Transmits Cleartext Credentials
- 11219 (3) - Nessus SYN scanner
- 10107 (1) - HTTP Server Type and Version
- 10386 (1) - Web Server No 404 Error Code Check
- 10662 (1) - Web mirroring
- 11032 (1) - Web Server Directory Enumeration
- 42057 (1) - Web Server Allows Password Auto-Completion
- 43111 (1) - HTTP Methods Allowed (per directory)
- 49704 (1) - External URLs
- 50344 (1) - Missing or Permissive Content-Security-Policy HTTP Response Header
- 50345 (1) - Missing or Permissive X-Frame-Options HTTP Response Header
- 91815 (1) - Web Application Sitemap

D.2 Reálný router

Vulnerabilities By Plugin

- 85582 (1) - Web Application Potentially Vulnerable to Clickjacking
- 26194 (1) - Web Server Transmits Cleartext Credentials
- 10107 (1) - HTTP Server Type and Version
- 10386 (1) - Web Server No 404 Error Code Check
- 10662 (1) - Web mirroring
- 11032 (1) - Web Server Directory Enumeration
- 11219 (1) - Nessus SYN scanner
- 24260 (1) - HyperText Transfer Protocol (HTTP) Information
- 42057 (1) - Web Server Allows Password Auto-Completion
- 43111 (1) - HTTP Methods Allowed (per directory)
- 49704 (1) - External URLs
- 50344 (1) - Missing or Permissive Content-Security-Policy HTTP Response Header
- 50345 (1) - Missing or Permissive X-Frame-Options HTTP Response Header
- 91815 (1) - Web Application Sitemap

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `BP_Bednar_Martin_2017.pdf` ..... text práce ve formátu PDF
- `BP` ..... LaTeX zdroje
- `Firmware` ..... DHP-1565 rozbalený firmware
  - `DHP-1565_FIRMWARE_1.01.ZIP` ..... Zabalený firmware
  - `DHP-1565_reva_GPL_code.rar` ..... Zdrojové kódy DHP-1565
  - `squashfs4.0.tar.gz` ..... zdrojové kódy squashfs
- `nvram` ..... zdrojové kódy libnvram
  - `nvram.default` ..... nvram defaults vytažené z firmware
  - `nvram.awk` ..... awk generátor defaults.h
- `Stana` ..... Framework Stana
- `VMs` ..... Virtuální stroje a jádra
  - `vmlinux-3.16.0-4-4kc-malta` ..... jádro Linux pro architekturu Mips-Malta
  - `virt2.tar.gz` ..... archiv s obrazy disků virtuálnch strojů
- `outputs` ..... výstupy příkazů
  - `kernel_strings` ..... Výstup příkazu strings na jádro
  - `Real router` ..... data z reálného routeru
  - `Virtual router` data z routeru s virtualizovaným webovým rozhraním
  - `router_diff` ..... rozdíly mezi virtuálním a reálné webui
  - `out.strace` ..... výstup strace, slouží jako vstup pro Stana
  - `DHP-web_h77lwq.html` ..... Nessus report DHP-1565
  - `virt-web_xupyon.html` ..... Nessus report virtual
  - `out.diff` ..... rozdíly mezi D-link squashfs a oficiálním squashfs
- `virt` ..... libvirt správce virtuálních strojů
- `virt2` ..... bash správce virtuálních strojů
- `router_test` ..... skripty na testování rotueru
- `db.sql.zip` ..... pg_dump databáze