VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

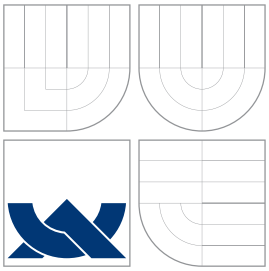
NÁSTROJ PRO SNAZŠÍ ZABEZPEČENÍ POČÍTAČŮ S OS LINUX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

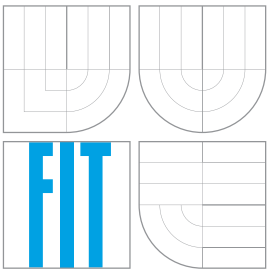
AUTOR PRÁCE
AUTHOR

Bc. MAROŠ BARABAS

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO SNAZŠÍ ZABEZPEČENÍ POČÍTAČŮ S OS LINUX

A TOOL TO EASILY LOCK DOWN LINUX COMPUTERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAROŠ BARABAS

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. TOMÁŠ VOJNAR, Ph.D.

BRNO 2009

Abstrakt

Účelem práce je příblížit čtenáři nové přístupy při zjišťování a odstraňování zranitelností v oblasti počítačové bezpečnosti a navrhnout nový systém na zlepšení bezpečnosti v počítačích s operačním systémem Linux. Tento systém má za úkol analyzovat vzdálené operační systémy a odhalit tím zranitelná místa, které můžou být následně odstraněné aplikováním existujících bezpečnostních standardů.

Abstract

The purpose of this thesis is to explain new approaches to scanning and locking vulnerabilities in computer security and to design a new system to improve security of computers running the Linux operating system. The purpose of this system is to analyze remote operating systems and detect and lock down vulnerabilities by existing security standards.

Klíčová slova

bezpečnost, správa, diagnostika, zranitelnost, Lock Down, NIST, DISA, STIG, MITRE, G2, Red Hat, SCAP, CVE, CCE, CVSS, OVAL, XCCDF

Keywords

security, administration, analysis, vulnerability, Lock Down, NIST, DISA, STIG, MITRE, G2, Red Hat, SCAP, CVE, CCE, CVSS, OVAL, XCCDF

Citace

Maroš Barabas: Nástroj pro snazší zabezpečení počítačů
s OS Linux, diplomová práce, Brno, FIT VUT v Brně, 2009

Nástroj pro snazší zabezpečení počítačů s OS Linux

Prohlášení

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením pána Doc. Ing. Tomáša Vojnara.

.....
Maroš Barabas
25. mája 2009

Poděkování

Chcem poďakovať Doc. Tomášovi Vojnarovi za odbornú pomoc a vedenie tejto práce. Taktiež ďakujem celému tímu firmy Red Hat za podporu a pomoc pri implementácii a taktiež vývojárom z firmy G2 za ich prínos v projekte open-scap.

Obsah

Kapitola 1

Úvod

Linux je spoľahlivý, relatívne jednoduchý operačný systém s bohatou históriou. Ponúka široké možnosti využitia a beží na skoro akomkoľvek hardweri. Jeho veľkou prednosťou je univerzálnosť a otvorené distribuovanie kódu[1], ktoré ponúka tisíce potenciálnych vývojárov a testerov pracujúcich na jeho zdokonaľovaní.

I napriek obrovskej vývojárskej komunite sú prieniky do linuxových systémov na dennom poriadku. Nie je to tým, že by tieto systémy boli z podstaty málo bezpečné, ale so zvyšujúcou sa univerzalitou a požiadavkami na funkčnosť sa stáva ich zabezpečenie stále ťažšie. Dnešný svet má vysoké nároky na poskytované služby a rozvoj systémov o nové súčasti a programy, spolu s pridanou funkčnosťou do systému zavádza aj nové chyby a zraniteľné miesta. Ak k nim pridáme faktory ako omylnosť človeka, či fenomén internetu, stáva sa toto prostredie pre akýkoľvek systém veľkou hrozbou. Preto sa snažíme nájsť optimálnu cestu medzi bezpečnosťou, jednoduchosťou a použiteľnosťou. Cesta, ktorá vedie k plnej funkčnosti, jednoduchosti a zároveň zaručuje 100% stabilitu a bezpečnosť si v dnešnej dobe dovolím nazvať utópiou.

1.1 Obsah a ciele práce

Prvým cieľom tejto práce je zoznámenie sa so základnými pojmami z oblasti počítačovej bezpečnosti, načrtnutie problematiky, ako aj nové pohľady na bezpečnosť a zraniteľnosť operačných systémov. V práci by som chcel objasniť nový pohľad na zraniteľnosti systému, ich detekciu, prípadne možnosti ich zabezpečenia a predstaviť novo vznikajúce štandardy, ktoré sa sústreďujú na čo najdetailnejší popis týchto zraniteľností a tým uľahčujú ich detekciu a nápravu.

Tak ako vznikajú stále nové štandardy a pribúdajú nové bezpečnostné riziká, musia aj spoločnosti zaoberajúce sa implementovaním bezpečnostných aplikácií držať krok s dobou a dané štandardy priniesť v podobe auditných, testovacích a zabezpečovacích mechanizmov svojim zákazníkom. Za touto snahou stojí aj hlavný cieľ tejto práce – navrhnúť a implementovať systém, ktorý by sa mal stať prvým open-source riešením na báze SCAP[2] štandardov poskytujúci bezpečnostné riešenia v oblasti operačných systémov. Projekt je zameraný hlavne na operačný systém Linux a opiera sa o využitie najmodernejšieho mechanizmu kontroly stavu systémov a ich náchylnosť na bezpečnostné incidenty, jazyk OVAL[3]. Systém bol pomenovaný podľa anglického sloveného spojenia „lock-down", čo je voľne preložiteľné ako „zabezpečiť".

[1] Myšlienka otvoreného kódu, `www.opensource.org`
[2] The Security Content Automation Protocol
[3] Open Vulnerability and Assessment Language

Projekt zadala firma Red Hat a na projekte sa zúčastňujú a aktívne prispievajú, hlavne početnými pravidelnými konzultáciami, inštitúcia NIST[4], skupiny MITRE[5] a G2[6] a rôzne materiály boli dodané agentúrou NSA[7] / CSA (národnou kryptologickou organizáciou vlády Spojených štátov amerických). Ich úloha v tomto projekte bude detailnejšie popísaná v kapitole 1.3.

Hlavné zameranie projektu sa počas celého obdobia od vzniku zadania až po súčasnosť niekoľkokrát zmenilo. Projekt bol najskôr zadaný ako sprievodca na konfiguráciu systému zabezpečovacími politikami, neskôr sa rozvinul na zabezpečovací mechanizmus, ktorý mal za úlohu narábať so štandardmi popisujúcimi bezpečnostné pravidlá a vo finálnej fáze aplikovať vybrané pravidlá na lokálny počítač. Po uskutočnení konferencie „4th Annual IT Security Automation Conference“ (ďalej len SAC konferencia) v septembri 2008, kde boli prezentované štandardy SCAP a ich možnosti využitia v manažmente zraniteľností a bezpečnostných konfigurácií, projekt začal získavať na význame a tým sa začal aj jeho účel rozširovať. V poslednej fáze ide o systém s diagnostickým a zabezpečovacím nástrojom, kde užívateľ po vybratí a upravení bezpečnostnej politiky dokáže v jednom okamihu zabezpečiť veľké množstvo počítačov a zároveň zbierať štatistické informácie o stave počítačov a tým detekovať prípadne narušenia v okruhu kontrolovaných systémov.

Projekt sa po určitom čase rodelil na dva samostatné projekty. Účelom prvého projektu je vytvorenie systému Lock-Down, ktorý slúži ako rozhranie ku knižnici open-scap[8], ktorej vytvorenie je účelom vzniku druhého projektu. V názvoch jednotlivých kapitol, je spomínaný iba jeden projekt z dôvodu, že táto práca popisuje pôvodný zámer a zadanie, ktoré popisuje obe časti ako spojenie jedného projektu. Ten sa ďalej delí na projekt Lock-Down a projekt open-scap ako bolo už spomenuté.

Ako je uvedené v ďalšom texte, systém Lock-Down je z pohľadu logického členenia rozdelený na dve samostatné časti a to časť, ktorá sa bude zaoberať diagnostikou systému a časť, ktorá bude aktívne zabezpečovať systém. V texte je projekt ponímaný ako celok a jednotlivé časti sú pre lepšiu zrozumiteľnosť pomenované ako diagnostická časť a zabezpečovacia časť. Rozdelenie z pohľadu architektúry na frontend a backend je z dôvodu predchádzajúcej verzie dokumentu [13] ponechaný, avšak počas vývoja bol backend rozšírený na spomínaný projekt open-scap, ktorý je naďalej súčasťou tejto práce. V konečnom dôsledku má toto rozdelenie za následok, že práca popisuje zároveň tvorbu dvoch projektov, open-scap a Lock-Down.

1.2 História a prínos projektu

Projekt Lock-Down vznikol ako samostatná myšlienka Steva Grubba, vedúceho tímu security engineering firmy Red Hat, na začiatku roku 2008. Avšak jeho príchod predznamenal viac ako ročný vývoj projektu sectool[9], ktorý slúži ako testovací nástroj na zisťovanie chýb alebo slabých miest v systéme, pomocou ručne písaných testov. Nástroj Lock-Down mal pôvodne bežať paralelne so sectoolom, ale prelom nastal na už spomínanej SAC konferencii v septembri 2008, kde boli predstavené automatické metódy (štandardy SCAP) na testovanie, meranie a vyhodnocovanie zraniteľností, ktorých odstránenie bolo cieľom práve projektu sectool, ale nami definovaným spôsobom, ktorý nepodporoval štandardy SCAP. Kvôli tomuto rozdielu, príliš odlišnej koncepcii projektov a tlakom spoločnosti na štandardizáciu

[4] National Institute of Standards and Technology, `www.nist.gov`
[5] Public-interest not-for-profit organization, `www.mitre.org`
[6] `http://www.g2-inc.com`
[7] National Security Agency
[8] `www.open-scap.org`
[9] `https://fedorahosted.org/sectool/`

bezpečnostných funkcií, boli práve štandardy SCAP príčinou ukončenia projektu sectool a vzniku nového bezpečnostného nástroja Lock-Down.

V rannom štádiu bol účel projektu zameraný na poskytnutie jednoduchého sprievodcu testovaním a zabezpečením linuxového systému, neskôr rozšírený o manažment vzdialených počítačov. Po uplynutí určitého času sa do projektu angažovala spoločnosť G2 a priniesla nový pohľad na štandardy SCAP a ich uplatnenie v komunite. Dnešná[10] predstava o projekte je úzko zviazaná s vytvorením knižnice, ktorá obsahuje všetky základné nízko-úrovňové funkcie poskytujúce API[11] k práci so SCAP štandardmi. Táto knižnica môže byť využitá na vývoj ďalších programov slúžiacich na automatický manažment zraniteľností, meranie a vyhodnocovanie bezpečnostných politík. Naopak, systém Lock-Down je aplikácia týchto myšlienok do praxe. Samotný program je ukážka využitia existujúcich štandardov a môže slúžiť ako predloha k vytvoreniu špecifických projektov zameraných na konkrétnu časť bezpečnostnej problematiky. Koncovému užívateľovi poskytuje jednoduchý nástroj na kontrolu a zabezpečenie systému.

1.3 Externá spolupráca

V tejto časti by som rád objasnil úlohu niektorých inštitúcií a spoločností priamo alebo nepriamo sa podieľajúcich na vývoji knižnice open-scap a dopad vzájomnej spolupráce na projekt. Prvým veľkým míľnikom v živote Lock-Downu bola určite konferencia SAC v septembri 2008, ktorá určila nový smer vývoja a pridala projektu určitú dôležitosť, aj vďaka ktorej bola následne nadviazaná spolupráca s firmou G2, ktorá stojí za vývojom projektu OVAL (viď kapitolu 3.1.5). Práve spoločnosť G2 venovala nemalé časové prostriedky na podporu projektu open-scap a v priebehu niekoľkých mesiacov dokázala implementovať základné funkcie systému OVAL do knižnice a prispôsobiť ho jej potrebám. Ďalej sa aktívne podieľali na návrhu knižnice a jazyka ORML[12] . Druhou nemenej významnou spoločnosťou je inštitúcia americkej vlády – National Institute of Standards and Technology (NIST). NIST je vládna organizácia, ktorá stojí za vznikom SAC konferencie a v prvom rade je zodpovedná za prijímanie a kontrolu nových štandardov. To sa priamo týka ďalšej neziskovej organizácie MITRE, ktorá popri práci pre americkú vládu (konkrétne pre ministerstvo obrany) vedie vlastný nezávislý výskum v rôznych oblastiach, ktorého úlohou je vývoj a nasadenie nových technológií. Tieto dve organizácie stoja za vznikom štandardov SCAP a spolu s Red Hatom sa podieľajú na vzniku nových štandardov vznikajúcich popri vývoji open-scap knižnice.

Prínos týchto organizácii je pre vývoj projektu veľmi dôležitý, ale prináša so sebou aj negatívne prvky ovplyvňujúce hlavne rýchlosť vývoja a zmeny v návrhu projektu. Veľkým problémom je veľmi zdĺhavá komunikácia a problematická synchronizácia tímov daná niekoľkohodinovými časovými rozdielmi a veľkou geografickou vzdialenosťou.

1.4 Konvencie a preklad

V tomto dokumente nájdete rôzne štýly textu, ktoré sú použité na odlíšenie rozdielnych typov informácií. V nasledujúcom texte sú niektoré príklady štýlov s vysvetlením ich významu.

[10] Slovo „dnešný" sa viaže k času písania tohto odstavca, čo je približne v prostriedku apríla roku 2009.

[11] API (application programming interface) je množina rutín (funkcií, metód,...), dátových štruktúr, tried a/alebo protokolov poskytovaná knižnicami a/alebo službami operačného systému za účelom podporiť vytváranie aplikácií (`http://en.wikipedia.org/wiki/Api`).

[12] ORML – OVAL Remediation Language, v čase písania tejto práce nebol ORML štandardom a jeho špecifikácia bola v štádiu návrhu.

Kurzívou sú väčšinou uvedené citácie alebo konkrétne príklady:

(APP3320.1: CAT II) The Designer will ensure the application has the capability to require account passwords having a minimum of 15 alphanumeric characters in length.

Časti kódu, príkazy, vstupy a výstupy príkazového riadku sú písané nasledovne:

```
$ python main.py
```

Nové pojmy a **dôležité slová** sú prvý krát napísané tučným písmom. Text, ktorý sa objaví na obrazovke, v ponukách alebo dialógoch bude v texte zvýraznený:
„po kliknutí na tlačidlo **Details** sa objaví ponuka zobrazujúca detaily konkrétneho klienta".

Pri písaní tejto práce som musel čerpať informácie z anglických textov, keďže literatúra z oblasti bezpečnosti v období posledných rokov je v slovenskom alebo českom jazyku nedostupná. Pri preklade takéhoto textu sa stáva, že preklad konkrétnych pojmov alebo definícií nie je jednoznačný alebo v slovenčine vôbec neexistuje. Preto by som rád upozornil na to, že pojmy technického charakteru použité v tomto texte sa nemusia presne zhodovať s pojmami v inej literatúre. Vo väčšine prípadov bude za takýmto slovom odkaz na komentár v spodnej časti stránky, kde bude dané slovo vysvetlené alebo bude uvedený jeho anglický ekvivalent.

1.5 Štruktúra práce

Štruktúra práce sa člení na sedem kapitol. V každej z nich sa zameriam na jednu problematiku týkajúcu sa projektov Lock-Down a open-scap. Pokiaľ nebude uvedené inak, pojem projekt bez bližšieho upresnenia budeme chápať ako celkový projekt, ktorý obsahuje nástroj Lock-Down a knižnicu open-scap.

Kapitola 2: Stručný úvod do problematiky zabezpečenia linuxových systémov a ich spojitosť s projektom. Vysvetlenie základných pojmov z oblasti bezpečnosti, teoretické prístupy pri riešení bezpečnostných problémov a spojitosť s projektom Lock-Down.

Kapitola 3: Kapitola začína uvedením štandardov implementovaných v projekte Lock-Down, pokračuje rozpravou o využiteľnosti nástrojov OVALDI, Augeas a štandardu STIG. Na záver je predstavený nový kandidát na SCAP, štandard ORML.

Kapitola 4: Podrobne spracované vstupné požiadavky projektu s následnou analýzou. V kapitole je uvedený postup od požiadaviek k prípadom použitia a analytickým triedam.

Kapitola 5: Návrh programu vychádzajúci z analýzy poskytuje čitateľovi celkový pohľad na štruktúru a architektúru projektu s patričným rozborom.

Kapitola 6: Zhrnutie implementácie, problémy a implementačné detaily na projekte.

Kapitola 7: Záver a popis projektu v poslednom štádiu so zhodnotením celkovej práce. V závere je taktiež podaná predstava o budúcnosti projektu.

Kapitola 2

Problematika zabezpečenia počítačov

Problematika počítačovej bezpečnosti je oblasť, ktorej sa už dlhý čas venujú rozličné bezpečnostné agentúry, skupiny, jednotlivci, či už ide o záujmovú činnosť, komerčný vývoj alebo národnú bezpečnosť . „Počítačová bezpečnosť“, je oblasť vedy, ktorá sa zaoberá odhaľovaním a eliminovaním rizík spojených s počítačmi [11]. O užšej špecifikácii alebo rozdelení nemá zmysel písať, pokiaľ si neobjasníme základné pojmy, ciele, úrovne a okolnosti, ktoré v základe viedli k vzniku činnosti, ktorú dnes môžeme označiť ako *počítačovú kriminalitu.*

V ďalšom texte sa zoznámime s ponímaním bezpečnosti na najzákladnejšej úrovni a uvedieme princípy bezpečnostných politík, ktorými sa hodlá moderný svet v budúcnosti riadiť k dosiahnutiu čo najväčšej bezpečnosti a tým aj dôveryhodnosti modernej technológie zvanej IT.

2.1 Úvod do bezpečnosti

Prvá myšlienka, ktorá sa nám vybavuje pri spomenutí slova „bezpečnosť“ je určite bezpečnosť pri hrozbe fyzickej ujmy, či už našej, alebo ujmy na materiálnych statkoch, ktoré považujeme za cenné. Pokiaľ si odmyslíme samotný ľudský život, zdravie a všetky duševné cennosti, ktoré si človek váži, tak to, čo pre nás ostane cenné a to, čo sa oplatí zabezpečiť z materiálnej časti života, je naše súkromné vlastníctvo. Pre prostých ľudí to môže predstavovať dom či auto, pre firmy to môže skôr znamenať technológiu, ktorá danú firmu stavia pred konkurenciu a pre národ je to strategická informácia zabezpečujúca prosperitu a niekedy aj vojenskú a ekonomickú výhodu nad ostatnými. V každom prípade, keď budeme hovoriť o týchto cennostiach a statkoch, budeme ich nazývať **aktívami** daného subjektu a odmyslíme si ich podstatu v zmysle materiálnej alebo fyzickej podoby a budeme uvažovať iba o ich hodnote a to hodnote ako pre majiteľa týchto aktív, tak aj pre útočníka.

Bezpečnosť ako pojem nám naznačuje, že pred niečím alebo niekým chránime naše aktíva. Je celá škála činností, ľudí a vlastností pred ktorými sa môžeme chrániť. V prípade domu to môže byť povodeň, v prípade firmy konkurencia a v prípade národnej bezpečnosti napríklad špióni. Všetky prípady, ktoré súvisia s ohrozením bezpečnosti a ktoré môžu narušiť bezpečnosť súhrnne nazveme **hrozby**. Tie môžeme ďalej rozdeliť na hrozby neúmyselné – poruchy, chyby, živelné pohromy, atď a úmyselné – krádež, úmyselné poškodenie, neoprávnená manipulácia a podobne.

„Hrozba je vlastnosť prostredia, ktorá môže narušiť bezpečnosť, pokiaľ k tomu dostane príležitosť.“

Keď hovoríme o hrozbách a aktívach, musíme spomenúť aj posledný základný pojem používaný v spojitosti s bezpečnosťou a tým je **zraniteľné miesto**. Zraniteľné miesto je chyba, medzera, nedodržanie stanoveného postupu alebo slabina, ktorá môže byť využitá na narušenie bezpečnosti [1]. To všetko má za následok zmenu hrozby na bezpečnostný incident. Avšak i v prípade, že došlo k incidentu, bezpečnostná politika nám pomáha k odhaleniu a následnému zneškodneniu problému, prípadne k minimalizácii škôd spôsobených týmto narušením.

V ďalšom texte s pomocou vysvetlených pojmov si definujeme oblasť vedy, ktorá sa venuje bezpečnosti v informačných technológiách a zavedieme nové pojmy týkajúce sa počítačovej bezpečnosti.

2.2 Počítačová bezpečnosť

Každá oblasť vedy zaoberajúca sa bezpečnosťou si kladie za cieľ ochrániť aktíva a to na rôznych úrovniach a rôznymi metódami.

S príchodom 21. storočia a s nasadením technológií ako je internet do bežného života, si človek nevie predstaviť život bez počítača. Cez internet sa dnes dá nakupovať, komunikovať s bankami, podávať daňové priznania, efektívne riadiť firmu bez nutnosti cestovať. Ale všetky vymoženosti doby internetu so sebou prinášajú veľký priestor pre vznik nových hrozieb plynúcich väčšinou z kriminálnej činnosti. Tým sa ale otvára aj nové pole pôsobnosti pre oblasť počítačovej bezpečnosti a to bezpečnosť spojenú s prenosom informácií, prístupom k systémom, prípadne overovaním identity a podobne.

Pojem „počítačová bezpečnosť“ by sme mohli rozdeliť do nasledujúcich dvoch hlavných skupín:

1. **Sieťová bezpečnosť** – bezpečnosť prístupu k systému, komunikácia a prenos dát.
2. **Bezpečnosť systému** – integrita systému a dát, dostupnosť a dôveryhodnosť.

Tieto dve hlavné skupiny sú navzájom prepojené a skoro neoddeliteľné, hlavne v reálnom svete, kde musíme počítať s faktom, že každý systém je obsiahnutý v nejakej sieti (nemusí to byť internet). V ideálnom prípade by sme dokázali zabezpečiť počítač tým, že by sme ho umiestnili tisíce metrov pod zem do izolovanej miestnosti bez spojenia s vonkajším svetom a bez možnosti fyzického dosahu, ale tento prístup nemá v dnešnom svete skoro žiadne opodstatnenie.

Keď hovoríme o bezpečnosti, musíme si uviesť základné ciele bezpečnosti a určiť prístupy, ako zaručiť alebo zvýšiť bezpečnosť v systémoch.

Počítačová bezpečnosť v informačných systémoch si kladie za cieľ zaručiť nasledujúce tri základné body [20]:

1. **Dôveryhodnosť** – ochrana proti neoprávnenému prezradeniu informácie.
2. **Integrita** – ochrana proti neoprávnenej modifikácii informácie.
3. **Dostupnosť** – ochrana proti neoprávnenému odopreniu prístupu k dátam alebo službám.

Zvýšenie bezpečnosti môžeme zabezpečiť troma základnými prístupmi:

1. **Prevencia** – ochrana proti hrozbám.
2. **Detekcia** – odhalenie neoprávnenej činnosti a slabého miesta v systéme.
3. **Náprava** – odstránenie slabého miesta v systéme.

2.3 Bezpečnosť systému Linux

O bezpečnosti linuxových systémov existuje veľa literatúry a bolo napísaných veľa článkov a táto práca je zameraná na zraniteľnosti systému pred vznikom bezpečnostného incidentu, preto sa v tomto texte budem venovať novému prístupu k definovaniu bezpečnostných rizík a zraniteľností systému a to konfigurácii systému[1].

Linuxový systém je viacužívateľský systém, ktorý ponúka veľké množstvo štandardných služieb a funkcií. Čím väčší počet funkcií systému, tým väčšia pravdepodobnosť výskytu zraniteľnosti. Bežný užívateľ si neuvedomuje dôvod, prečo by mohol byť práve jeho systém napadnutý a považuje ho za málo atraktívny pre útočníkov. Pravdou je, že na každom systéme sa nachádzajú aktíva, ktoré sú pre určitú skupinu útočníkov dôležité. Medzi tieto aktíva patrí hlavne:

- **Konektivita** - útočník použije počítač ako základňu pre „skutočný" útok.
- **Procesor** - útočník potrebuje procesorový čas na náročné operácie a získa ho spojením viacerých počítačov.
- **Diskový priestor** - voľný diskový priestor v napadnutom počítači môže slúžiť na uloženie a prípadnú distribúciu nelegálne získaných dát.
- **Dáta** - útočník sa zaujíma o dáta z napadnutého systému. Môžu to byť informácie o kreditných kartách, účtoch, či heslách.

Posledným dôvodom útočníka na napadnutie systému môže byť túžba dobyť a prípadne zničiť systém a získať tak slávu, uznanie alebo potešenie.

Operačný systém Linux je postavený na modeli open-source, čo znamená, že zdrojové kódy systému sú voľne dostupné a ktokoľvek ich môže prezerať a modifikovať. Tento model má svojich zastáncov aj kritikov. Hlavnou nevýhodou, ktorá je open-source modelu vytýkaná je práve nevýhoda otvorenosti kódu, kde útočník môže analyzovať kód a prípadnú chybu okamžite zneužiť. Preto je z proaktívneho hľadiska (pred útokom) nutné dbať na dôkladnú konfiguráiu systému a odstránenie všetkých známych zraniteľností.

Hlavnými problémami v linuxových systémoch sú obyčajne zlé nastavenia kľúčových služieb systému, ako serverové alebo bezpečnostné aplikácie, zlé práva súborov a zastaralé verzie programov. My potrebujeme rôzne problémy, ktoré sa môžu v systéme objaviť, popísať uceleným spôsobom. Preto definujeme pojem **konfigurácia**, ktorý obsahuje informácie o nastavení určitej časti systému. Konfiguráciou môžeme nazvať informáciu o verzi určitého programu, o spustených službách, rôznych nastaveniach programov a podobne. Príkladom jednoduchej konfigurácie je informácia o nastavení minimálnej požadovanej dĺžky hesla užívateľa. Pojmom **konfigurácia systému** budeme nazývať množinu konfigurácií daného systému.

Ak dokážeme efektívne získať dosť obsiahlu konfiguráciu sysému, môžme definovať pravidlá (tzv. konfiguračné pravidlá, často označované ako „politika zabezpečenia" alebo „bezpečnostné zásady"), aké by mali mať jednotlivé konfigurácie hodnoty a nastavenia tak, aby

[1]Pojem „konfigurácia systému" je preložená z často používaných pojmov *configuration issue* alebo *configuration statement.*

bol z nášho pohľadu systém bezpečný. Ak je spomínanou konfiguráciou minimálna požadovaná dĺžka hesla užívateľa, môžme definovať, že hodnota tejto dĺžky nižšia ako päť znakov je zraniteľnosť, pretože útočník disponuje prostriedkami na zistenie hesla a dokáže touto cestou preniknúť do systému. Definície týchto hodnôt boli vytvorené a popisujú ich štandardy ako STIG (viď kapitolu 3.6).

Pokiaľ získame konfiguráciu systému a máme definíciu vyžadovaných hodnôt, ponúka sa nám možnosť tento systém zabezpečiť automatizovaným spôsobom, bez nutnosti nápravy a zásahu užívateľom. Doteraz neznámy prístup k automatizovanému zabezpečeniu systému definujú štandardy SCAP (viď kapitolu 3.2), ktoré popisujú metódy na detekciu a odstránenie zraniteľných miest.

Jedným z problémov pri tomto prístupe sa javí samotné zistenie konfigurácie systému. Tento problém bol vyriešený po vzniku štandardu OVAL (viď kapitolu 3.1.5).

Tento proaktívny prístup sa dá využiť aj v prípade, že útočník už do systému prenikol a neustála diagnóza a porovnanie zmien v konfiguráciach systému pred a po útoku dokáže zistiť prítomnosť útočníka, ktorý pozmenil niektorú konfiguráciu systému. Samozrejme tento prístup funguje iba dovtedy, kým útočník priamo nezmení nástroj, ktorý s týmito konfiguráciami pracuje.

2.3.1 Diagnóza a zabezpečenie

Celková konfigurácia systému je špecifikovaná pomocou štandardu OVAL. OVAL definuje jazyk na popis konfiguráciií a interpret tohto jazyka dokáže podľa určitých pravidiel zistiť aké programy a služby sú nainštalované na danom počítači, všetky dôležité parametre a konfiguračné možnosti týchto programov. Konfigurácia systému sa následne porovná s konfiguráciou štandardu, ktorá je považovaná za najbezpečnejšiu vzhľadom k danému systému a štandardu, ktorý ju definuje a celý proces sa vyhodnotí výsledkom porovnania. Celá diagnóza systému je postavená na tomto princípe. Zabezpečenie systému sa odlišuje v tom, že nezisťujeme ani neporovnávame konfiguráciu systému, ale podľa daného štandardu túto konfiguráciu systému vnútime. Snahou projektov Lock-Down a open-scap je implementácia týchto procesov.

2.4 Úloha projektov Lock-Down a open-scap v bezpečnosti

Myšlienka vzniku nového nástroja na zabezpečenie bola podporená existujúcimi bezpečnostnými hrozbami. Nástroj Lock-Down má priniesť nový spôsob testovania a zabezpečenia systému podľa daného štandardu, ktorý obsahuje konfiguračné pravidlá (pravidlá aké hodnoty by mali mať jednotlivé konfigurácie).

V tejto kapitole boli vysvetlené princípy a základné pojmy týkajúce sa bezpečnosti. Obsahom tejto práce je vývoj nového nástroja, ktorý bude riešiť problémy spojené s bezpečnosťou a zlepšovať podmienky pre úspešné odvrátenie bezpečnostných hrozieb v počítačovej bezpečnosti. Projekt je z väčšej časti zameraný na operačný systém Linux, konkrétne na distribúciu Red Hat rovnomennej spoločnosti.

Pokiaľ by sme chceli zaradiť projekt Lock-Down do kategórií z predchádzajúcich podkapitol, tak by to bol nástroj na zlepšenie bezpečnosti systému zabezpečením lepšej dôveryhodnosti a integrity programov a dát, zväčša zameraný na prevenciu a nápravu potenciálnych slabých miest v systéme.

Z určitej menšej časti sa dá program využiť ako IDS (Intrusion detection system) a to hlavne diagnostickou časťou projektu, porovnávaním konfigurácie systému pred a po zásahu útočníkom do systému (ktorý z pravidla zmení niektorú konfiguráciu na zaistenie prístupu

do systému i po odstránení zraniteľnosti, ktorú využil v prvom prístupe). Zabezpečovacia časť projektu slúži na prevenciu proti hrozbám nápravou známych zraniteľných miest.

Na nasledujúcom príklade si uvedieme možnosť rôzneho prístupu k zabezpečaniu rôznych počítačov na demonštráciu využitia nástroja Lock-Down v počítačovej bezpečnosti.

Užívateľ má počítače na ktorých beží webový server a VPN[2] server, oba prístupné zo sveta a skupinu zamestnaneckých desktopov v sieti NAT[3] s prístupom do internetu. Máme tri druhy konfigurácií: webový server bude mať zabezpečenie hlavne na zaistenie dostupnosti a integrity, VPN server bude mať najvyššie zabezpečenie zamerané na dôveryhodnosť a integritu a zamestnanecké počítače, ktoré môžu mať nižšiu úroveň zabezpečenia. Administrátor pre každý počítač zaistí úroveň zabezpečenia vhodným výberom štandardu a výberom zásad bezpečnostnej politiky z tohto štandardu, tak ako mu to bude vyhovovať na danom počítači (záleží na službách, ktoré budú poskytovať). Následne zabezpečí tieto počítače programom Lock-Down a zaistí, že sa na daných počítačoch nenachádzajú žiadne známe zraniteľné miesta, voči danému štandardu. Samozrejme by sa dal každý počítač zaistiť najvyšším možným stupňom ochrany, ale musíme si uvedomiť, že to v určitom prípade môže napríklad znamenať vypnutie služby apache[4] a teda nezmyselnú konfiguráciu v prípade webového serveru. Preto je dôležité najprv pripraviť najvhodnejšiu konfiguráciu pre dané počítače tak, aby neobmedzovali funkčnosť a hlavne účel systému.

Druhým krokom je následné pravidelné testovanie počítačov na bezpečnostné incidenty. Tie sa môžu napríklad prejaviť tým, že na našom webovom serveri je zapnutá služba, ktorá by podľa prítomnej konfigurácie vybraného štandardu mala byť vypnutá a my sme odhalili neoprávnený prístup k serveru a môžeme včas zasiahnuť.

[2]Virtual Private Network
[3]Network Address Translation
[4]HTTP server

Kapitola 3

Súčasti projektu

V tejto kapitole budú vysvetlené súčasti projektu[1], štandardy a nástroje, ktoré využíva. Projekt musí používať a spĺňať rozličné štandardy, korých väčšinu implementuje knižnica open-scap. Hlavnou súčasťou je štandard SCAP inštitútu NIST, ktorý stojí ako základ a jeden z dôvodov vzniku projektu. Nasledujúce kapitoly čerpajú z publikácií [2, 3, 4, 5, 6, 7, 8, 10, 9].

Okrem štandardov SCAP vznikajú štandardy, ktoré s ním blízko súvisia:

- **CWE** – Common Weakness Enumeration – popis slabín v programoch (zdrojových súboroch) v spojitosti s rôznymi faktormi, ako je napríklad platforma.
- **CEE** – Common Event Expression – uvádza presný formát na vytváranie logovacích správ.
- **CRF** – Common Result Format – špecifikuje formát na poskytovanie výsledkov medzi programami hlavne kvôli vzájomnej kompatibilite medzi systémami.

Tieto štandardy nie sú implementované, ale môžu byť v budúcnosti pre rôzne projekty, ktoré vzniknú nad knižnicou open-scap dôležité. Štandard CEE bude v knižnici open-scap implementovaný, momentálne ale má veľmi nízku prioritu, aby bol zahrnutý do plánu projektu.

3.1 Štandardy SCAP

SCAP – The Security Content Automation Protocol. Štandard SCAP je metóda na použitie špecifických štandardov, ktoré umožňujú automatický manažment zraniteľnosti, meranie tejto zraniteľnosti a vyhodnotenie vyhovenia bezpečnostnej politiky (FISMA compliance) [6]. Špecifickejsšie, SCAP kombinuje viacero otvorených štandardov a definuje ako tieto štandardy kombinovať a spôsob ako s nimi pracovať. Jednotlivé štandardy číselne popisujú softwarové chyby, bezpečnostné konfigurácie, produktové názvy, merajú systém na zistenie prítomnosti zraniteľných miest a poskyujú mechanizmy na ohodnotenie výsledkov meraní na vyhodnotenie dopadu zistených bezpečnostných rizík.

SCAP obsahuje tieto špecifické štandardy :

- **CVE** – Common Vulnerabilities and Exposures – verejný zoznam známych zraniteľných miest a chýb.

[1] Projem projekt je nutné chápať ako spojenie projektov Lock-Down a open-scap.

- **CCE** – Common Configuration Enumeration – poskytuje unikátne identifikátory v zozname známych konfigurácií systému.
- **CPE** – Common Platform Enumeration – definuje schému popisu pre identifikáciu systému, platformy a programového balíka.
- **CVSS** – Common Vulnerability Scoring System – poskytuje „kalkulačku“ alebo systém na výpočet skóre určitej zraniteľnosti na základe charakteristík systému a prostredia.
- **XCCDF** – The Extensible Configuration Checklist Description Format – štandard, ktorý definuje XML formát špecifikujúci bezpečnostné kontrolné zoznamy, zrovnávacie testy a konfiguračné dokumenty.
- **OVAL** – Open Vulnerability and Assessment Language – štandardizuje kroky na zistenie rôznych bezpečnostne relevantných informácií z počítača.

Tieto otvorené štandardy boli vytvorené a sú naďalej vyvíjané rôznymi inštitútmi ako je MITRE, NSA a špeciálna záujmová skupina v rámci FIRST – Forum of Incident Response and Security Teams. Nižšie sú tieto štandardy popísané detailnejšie.

3.1.1 Štandard CCE

Keď hovoríme o CCE, musíme si najprv objasniť, čo pre tento štandard znamená „zoznam konfigurácií systému“. Ak je napríklad v štandarde XYZ zásada, že každý užívateľ musí mať dĺžku hesla väčšiu ako x znakov, tak ide o konfiguráciu systému a je tejto zásade priradený unikátny identifikátor CCE. Na obr. 3.1 je vidieť záznam položky CCE-3416-5, obsahujúca odporúčané hodnoty (v poslednom stĺpci), kde príslušný štandard definuje, či má byť služba rhnsd podľa tohto štandardu zapnutá alebo vypnutá. Položka ďalej obsahuje popis mechanizmu pomocou ktorého je možné toto kritérium zistiť (aplikovať) a parametre CCE záznamu.

CCE priraďuje unikátny identifikátor ku konfiguráciam za účelom uľahčiť rýchlu a presnú koreláciu medzi konfiguráciami v rozdielnych doménach. CCE je momentálne v správe MITRE corporation.

CCE ID	CCE Description	CCE Parameters	CCE Technical Mechanisms		NSA "Guide to the Secure Configuration of Red Hat Enterprise Linux 5" (Section)	NSA "Guide to the Secure Configuration of Red Hat Enterprise Linux 5" (Recommended Value)
CCE-3416-5	The rhnsd service should be enabled or disabled as appropriate.	enabled / disabled	via chkconfig		2.1.2.2	disabled

Obrázok 3.1: Položka CCEv5 Red Hat Enterprise Linux 5

Všeobecne je cieľ CCE identifikátorov určený schopnosťou automatického procesu (myslí sa proces bez zásahu človeka) overiť parameter určitej časti konfigurácie. Ak daná konfi-

gurácia môže byť verifikovaná ohodnocujúcim nástrojom alebo aplikovaná systémom na manažment konfigurácií, mal by k nej byť priradený CCE identifikátor.

Štandard má za úlohu poskytnúť jedinečný zoznam konfigurácii, ktorý poskytuje referencie na zraniteľnosti pomocou unikátnych identifikátorov. Ako príklad môžeme uviesť príklad použitý v dokumente [10] definície CCE: Anglická veta „Use strong password", ktorá je prinajmenšom vágna a neurčitá (dalo by sa povedať „fuzzy"). Pri použití CCE by danú vetu nahradili nasledujúce tvrdenia:

- The „minimum password age" setting should meet minimum requirements (CCE-2439-8)
- The „minimum password length" setting should meet minimum requirements. (CCE-2981-9)
- The „password must meet complexity requirements" setting should be configured correctly. (CCE-2735-9)
- The „enforce password history" setting should meet minimum requirements. (CCE-2994-2)
- The „store password using reversible encryption for all users in the domain" setting should be configured correctly. (CCE-2889-4)

CCE identifikátory sú definované rôzne pre rozličné bezpečnostné modely a pre rôzne platformy. Podrobnejšie vysvetlenie by bolo nad rozsah tejto práce.

Štruktúra CCE

- ID: univerzálny identifikátor pre CCE (CCE-####-#).
- Definícia: popis konfigurácie v ľudsky zrozumiteľnej forme.
- Konceptuálne parametre: parametre, ktoré musia byť špecifikované pri implementácii CCE systému.
- Technické mechanizmy: technické prostriedky k dosiahnutiu požadovaného výsledku.
- Citácie: ukazatele na časti dokumentov alebo nástrojov, kde je daná konfigurácia detailnejšie popísaná.

Parametre

V nasledujúcom príklade je na dvoch veľmi podobných konfiguráciach vysvetlený dôvod vzniku parametrov v CCE:

- Zdroj: DISA Gold Disk Tool for Windows 2000
- Konfigurácia: Hranica pokusov o prihlásenie k účtu = 3
- Zdroj: CIS Level 1 Benchmark for Windows 2000
- Konfigurácia: Hranica pokusov o prihlásenie k účtu = 50

Tieto dve konfigurácie je možné považovať za rovnaké nastavenie „maximálneho počtu pokusov o prihlásenie". Líšia sa iba rozdielnou hodnotou nastavenia, čo je spôsobené rozdielnymi autormi. CCE rieši túto nezhodu pridaním hodnoty k CCE ako parameter:

CCE-3229-2
Definícia: Maximálny počet pokusov o prihlásenie by mal spĺňať minimálne požiadavky
Parametre: Počet pokusov

Technické mechanizmy

Technické mechanizmy popisujú kontrolné činitele použité na ovplyvnenie zmien v systéme indikované konfiguráciou. Technické mechanizmy sú ďalším aspektom konfigurácií, ktorý bol pridaný do štandardu CCE.

Nasledujúci príklad ukazuje tri rôzne mechanizmy v jednom zázname CCE [3]:

CCE-3260-7
Definition: The "Log Dropped Packets" option for the Windows Firewall should be configured correctly for the Domain Profile.
Parameter: Enabled or Disabled.
Technical Mechanisms:

- *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\DomainProfile\Logging\LogDroppedPackets*
- *Computer Configuration\Administrative Templates\Network\Network Connections\Windows Firewall\Domain Profile\Windows Firewall: Allow Logging - Log Dropped Packets*
- *Control Panel\Windows Firewall\Advanced\Security Logging\Logging Options\Log dropped packets*

Tieto mechanizmy definujú cestu, akou sa dá pre daný CCE záznam zmeniť jeho hodnota.

Citácie

Posledným prvkom CCE štandardu sú citácie, ktoré obsahujú ukazovatele na konkrétne časti dokumentov, nástrojov, alebo iných zdrojov, kde je daná konfigurácia detailnejšie popísaná.

3.1.2 Štandard CVSS

CVSS - Common Vulnerability Scoring System. CVSS je štandard na vyhodnocovanie závažnosti danej zraniteľnosti. Definuje mieru hodnotenia, ako je daná zraniteľnosť vážná v porovnaní s ostatnými definovanými zraniteľnosťami tak, aby reakcia mohla byť priorizovaná. Na toto vyhodnotenie používa rôzne metriky, ktoré sú rozdelené do troch skupín (viď obr. 3.2):

1. Základné metriky: odvíjajúce sa od vnútorných vlastností zraniteľnosti.
2. Časové metriky: pre vlastnosti, ktoré sa vyvíjajú počas životného cyklu zraniteľnosti.

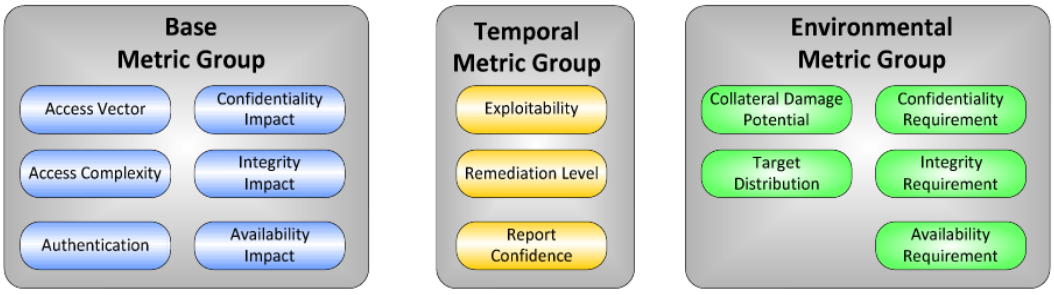
3. Metriky prostredia: pre vlastnosti, ktoré závisia od určitej implementácie alebo prostredia.

Účel množiny základných metrík je definovanie základných charakteristík zraniteľnosti, ako je napríklad dosiahnuteľnosť zraniteľnosti zo siete, aké dodatočné podmienky sú nutné na využitie zraniteľnosti a dopad zneužitia na dôveryhodnosť, integritu a dostupnosť systému. Časové metriky a metriky prostredia poskytujú kontextové informácie, ktoré presnejšie reflektujú riziká týkajúce sa daného prostredia. Pri stanovení celkovej závažnosti zraniteľnosti sa postupuje jednotlivým vyhodnotením vektorov skupín metrík. Každý vektor je textový reťazec obsahujúci hodnoty priradené ku každej metrike a používa sa na informovanie, ako presne bolo dané skóre pre každú zraniteľnosť odvodené. Jednotlivé vektory sú zobrazené vtabuľke 3.1, kde sa každá metrika skladá z vektorov špecifikovaných skratkami a možnými hodnotami vektora v hranatých zátvorkách. Názov je od hodnôt oddelený dvojbodkou a jednotlivé vektory lomítkom. Základná metrika obsahuje napríklad „Access vektor" špecifikovaný ako *AV:[L,A,N]*, kde L, A a N sú hodnoty vektora. Definícia týchto hodnôt je uvedená v príklade nižšie.

Skupina metrík	Vektor
Základná	AV:[L,A,N]/AC:[H,M,L]/Au:[M,S,N]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C]
Časová	E:[U,POC,F,H,ND]/RL:[OF,TF,W,U,ND]/RC:[UC,UR,C,ND]
Prostredia	CDP:[N,L,LM,MH,H,ND]/TD:[N,L,M,H,ND]/CR:[L,M,H,ND]/ IR:[L,M,H,ND]/AR:[L,M,H,ND]

Tabuľka 3.1: Základné vektory, časové vektory a vektory prostredia

Každá metrika má stanovené hodnoty, ktoré môže daná zraniteľnosť po ohodnotení obsahovať. Pomocou týchto hodnôt sa tvorí vektor danej zraniteľnosti.



Obrázok 3.2: CVSS skupiny metrík (prevzaté z literatúry[9])

V nasledujúcom príklade detailnejšie popíšem Access vektor v skupine základných metrík. Vzhľadom na rozsah práce je špecifikovanie ostatných vektorov príliš obsiahle a podrobné informácie je možné nájsť v literatúre [9].

Metrika Access Vecor reflektuje spôsob, akým môže byť daná metrika využitá vzhľadom na prístup k napadnutému systému. Čím môže byť útočník „vzdialenejší v sieti" k počítaču s danou zraniteľnosťou, tým je skóre vyššie.

Access vektor môže nadobúdať nasledujúce hodnoty:

- Local (L): zraniteľnosť môže byť využitá iba v prípade, že útočník má lokálny prístup k systému a to buď fyzický prístup alebo prístup k lokálnemu účtu (shellu).

- Adjacent Network (A): zraniteľnosť využiteľná prístupom z priľahlej siete vyžaduje od útočníka prístup k broadcastu alebo kolíznej doméne[2].
- Network (N): zraniteľnosť využiteľná so sieťovým prístupom znamená, že zraniteľnosť sa viaže na sieťový prístup a útočník ju môže využiť bez lokálneho prístupu.

Ku príkladu zraniteľnosť so základnými hodnotami metrík: „Access Vector: Local, Access Complexity: Medium, Authentication: None, Confidentiality Impact: None, Integrity Impact: Partial, Availability Impact: Complete", má vektor:

AV:L/AC:M/Au:N/C:N/I:P/A:C

Konečná hodnota má formu čísla, ktoré označuje celkovú závažnosť alebo prioritu skúmanej zraniteľnosti. Na stránkach inštitútu NIST je možné nájsť kalkulačku na výpočet skóre zraniteľností (viď `http://nvd.nist.gov/cvss.cfm?calculator&version=2`).

3.1.3 Štandard CVE

Common Vulnerabilities and Exposures je, ako už bolo spomenuté, verejný zoznam, ktorý poskytuje pohľad na rôzne známe problémy a zraniteľné miesta v systéme. Každá položka tohto zoznamu obsahuje unikátne číslo „CVE-ID", stav položky, ktorý indikuje, či ide o kandidáta, popis problému, a relevantný odkaz.

CVE-ID

CVE-2004-0042

Description

vsftpd 1.1.3 generates different error messages depending on whether or not a valid username exists, which allows remote attackers to identify valid usernames.

Impact

CVSS v2 Base Score: 5.0 (MEDIUM) (AV:N/AC:L/Au:N/C:P/I:N/A:N)

References

- SECTRACK:1008628
- URL:http://securitytracker.com/id?1008628

Status

Candidate This CVE Identifier has "Candidate" status and must be reviewed and accepted by the CVE Editorial Board before it can be updated to official "Entry" status on the CVE List. It may be modified or even rejected in the future.

Phase

Modified (20050526)

Votes

ACCEPT(2) Baker, Armstrong
NOOP(3) Williams, Wall, Cole
REJECT(1) Cox

Comments

```
Williams> insufficient data.
CHANGE> [Cox changed vote from REVIEWING to REJECT]
Cox> Expected behaviour.  By source code analysis the difference in
  behaviour mentioned in the report only occurs when an administrator has
  configured the server with an explicit userlist - either to allow or deny
  all users in the userlist.  The vsftpd manual page states that if a
  userlist is used then the user will be denied access before they are asked
  for a password to help prevent cleartext passwords being transmitted.
  Administrators who don't want this behaviour do not need to configure an
  optional userlist.
```

Obrázok 3.3: Príklad položky CVE

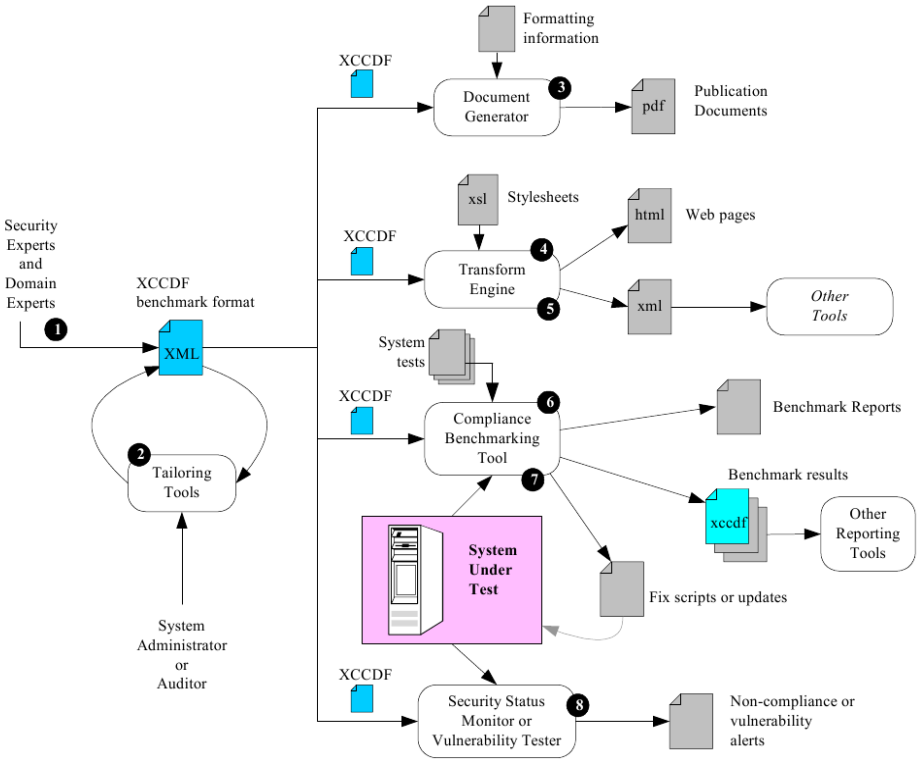
[2]Kolízna doména je segment siete, kde dátové pakety prenášané na zdieľanom médiu môžu navzájom „kolidovať".

Po prijatí novej položky sa pridelí problému indikácia kandidáta a nechá sa o probléme diskutovať. Po odsúhlasení je položka prijatá a príznak kandidáta sa zmení na tzv. „entry".

Na obr. 3.3 je vidieť príklad jednej položky CVE, ktorá je stále v stave kandidáta. CVE poukazuje na implementačnú chybu v programe vsftpd[3], ktorú môže útočník využiť.

3.1.4 Štandard XCCDF

The Extensible Configuration Checklist Description Format je štandard, ktorý definuje jazyk na písanie dokumentov, ktoré obsahujú množinu konfiguračných pravidiel použiteľných na prenos informácií, automatické testovanie a vyhodnocovanie zraniteľností. Štandard taktiež špecifikuje formát na uchovávanie výsledkov z testovania. XCCDF dokumenty sú špecifikované vo formáte XML. Vývojom tohto formátu sa zaoberá agentúra NSA v spolupráci s inými organizáciami a agentúrami.



Obrázok 3.4: Prípady použitia XCCDF (prevzaté z literatúry [2])

Na obr. 3.4 je zobrazený diagram použitia XCCDF súborov. V čiernych krúžkoch sú očíslované jednotlivé kroky pri jeho vytváraní a používaní. V prvom a druhom kroku dochádza k tvorbe súboru na ktorom sa podieľajú bezpečnostní experti a systémoví administrátori. Hlavnou časťou je využitie tohto súboru. Ako je vidieť z tretieho až ôsmeho prípadu, informácie z XCCDF dokumentu môžeme publikovať ako PDF[4], internetové stránky alebo XML súbor. XCCDF súbor ďalej slúži ako vstup pre rôzne testovacie nástroje výkonnosti a zraniteľností. Hlavným využitím XCCDF v projektoch open-scap a Lock-Down je sprostredkovanie informácií užívateľovi o jednotlivých zraniteľnostiach.

[3]Very Secure FTP server

[4]PDF (Portable Document Format) je súborový formát vytvorený firmou Adobe a slúžiaci na distribúciu dokumentov nezávislý na zdrojovom alebo cieľovom operačnom systéme.

```
<cdf:Rule id="pwd-perm" selected="1" weight="6.5" severity="high">
  <cdf:title>Password File Permission</cdf:title>
  <cdf:description>Check the access control on the password
     file.  Normal users should not be able to write to it.
  </cdf:description>
  <cdf:requires idref="passwd-exists"/>
  <cdf:fixtext>
    Set permissions on the passwd file to owner-write, world-read
  </cdf:fixtext>
  <cdf:fix strategy="restrict" reboot="0" disruption="low">
        chmod 644 /etc/passwd
  </cdf:fix>
  <cdf:check system="http://www.mitre.org/XMLSchema/oval">
    <cdf:check-content-ref href="ovaldefs.xml" name="OVAL123"/>
  </cdf:check>
</cdf:Rule>
```

Obrázok 3.5: Jednoduché XCCDF pravidlo

Príklad pravidla `<Rule>` písaného vo formáte XCCDF dokumentu je na obr. 3.5. Toto pravidlo špecifikuje konfiguráciu práv na súbore s heslami užívateľov (*/etc/passwd*) a definuje spôsob ako dosiahnuť jeho zabezpečenie. Konkrétne pravidlo zabezpečenia je definované pomocou príkazu `chmod 644 /etc/passwd`, ktoré nastavuje práva na zápis vlastníkovi súboru a skupine a ostatným užívateľom prideľuje iba práva na čítanie.

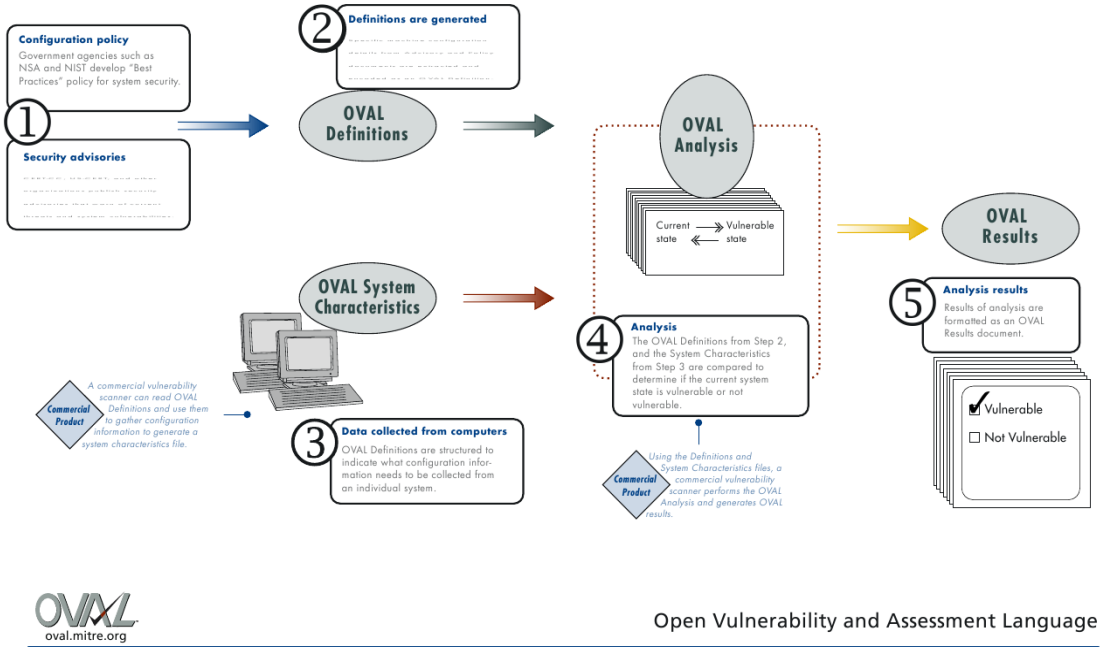
V projektoch open-scap a Lock-Down budú XCCDF dokumenty použité pri definícií pravidiel rozličných štandardov. Kompletná množina pravidiel podobných hore uvedenému príkladu so súborom s heslami bude uložená v jednom súbore, ktorý bude definovať štandard. Príkladom takéhoto štandardu je STIG (viď 3.6). V konečnej podobe nebude dokument použitý v projekte Lock-Down obsahovať záznamy s pravidlami `<Rule>`, ale iba definície a odkazy do definičného súbory OVALu alebo ORML (viď nižšie).

3.1.5 Štandard OVAL

Open Vulnerability and Assessment Language je štandard, ktorý umožňuje popis otvorených, verejne dostupných informácií z oblasti bezpečnosti a štandardizuje spôsob doručenia týchto informácií do celého spektra bezpečnostných nástrojov a služieb. OVAL špecifikuje tri štandardizované kroky tzv. ohodnocujúceho procesu: reprezentáciu informácií o konfigurácií testovaných počítačov, analyzovanie systému na zistenie konkrétneho stavu počítača (slabého miesta, určitej konfigurácie, záplaty) a zobrazenie výsledkov tohto ohodnotenia. OVAL vyvinul tri schémy napísané v XML, ktoré korešpondujú s tromi spomenutým krokmi:

1. **OVAL System Characteristics schema:** reprezentuje informácie o systéme,
2. **OVAL Definition schema:** slúži na vyjadrenie konkrétneho stavu,
3. **OVAL Results schema:** na zobrazenie výsledkov hodnotenia.

Schéma na obr. 3.6 ukazuje klasický prípad použitia OVALu. V prvom kroku sa získa (z NSA, NIST) popis najlepšej možnej skupiny pravidiel z pohľadu bezpečnosti systému. V druhom kroku sa z tohto popisu vytvorí OVAL definícia, ktorá obsahuje definované stavy objektov, ktoré budeme testovať. V treťom kroku importujeme túto definíciu do programu, ktorý získa charakteristiku systému. Tá je veľmi podobná súboru s definíciami, akurát obsahuje stavy objektov, ktoré zodpovedajú aktuálnej konfigurácii systému. Vo štvrtom kroku



Obrázok 3.6: Schéma procesu OVAL

sa táto definícia porovná s charakteristikou systému. Výstupom je OVAL Result dokument, ktorý poskytuje výsledky hodnotenia ukazujúce na bezpečnostné problémy a stručný popis systému. Tieto informácie sa dajú následne využiť rôznymi spôsobmi. Náš zámer je podať výsledok testovania užívateľovi, poukázať na závažné chyby a nedostatky v systéme, a pre každú chybu spočítať jej CVSS (viď kapitolu 3.1.2). Ďalším zámerom je poskytnutie užívateľovi zabezpečenie týchto zistených chýb.

Dokument štandardu OVAL obsahuje konfiguračné pravidlá, ktoré majú zložitú XML štruktúru prispôsobenú na definíciu konfigurácie tak, aby bolo zistenie jej stavu čo najjednoduchšie. Keďže interpretáciou jazyku OVAL dokážeme zo systému získať informácie o konfiguráciach, bolo by pre projekt Lock-Down zaujímavým riešením použiť tento systém na zabezpečenie konfigurácie tak, ako je použitá na spomenutú diagnózu. Jazyk OVAL žiaľ nie je natoľko komplexný, aby dokázal popísať spôsob, akým zabezpečiť nejakú chybu a preto musí byť v projekte použitý dokument písaný iným jazykom a to jazykom ORML, popísaným vnasledujúcej kapitole.

Nasledujúci príkald demonštruje štruktúru časti definičného súboru OVAL, ktorý popisuje doporučovené konfigurácie systému. Časť XML súboru je vytiahnutá pre názornú ukážku definovania a odkazovanie jednotlivých elementov v rámci definícií zraniteľností. Príklad sa týka zistenia operačného systému testovaného počítača a v každom jednotlivom konfiguračnom pravidle v tomto súbore je odkaz na toto pravidlo, aby sa zaistilo, že každý test, ktorý sa bude následne vykonávať (nemusí sa testovať počítač voči celému definičnému súboru, ale iba voči jednotlivým pravidlám) je spustený na určenom operačnom systéme. Kód tohto pravidla je vidieť v nasledujúcom texte:

```
<definition class="inventory" version="1" id="oval:gov.irs.rhel5:def:10000">
   <metadata>
      <title>Red Hat Enterprise Linux 5</title>
      <affected family="unix">
         <platform>Red Hat Enterprise Linux 5</platform>
      </affected>
      <reference ref_id="cpe:/o:redhat:enterprise_linux:5" source="CPE"/>
      <description>
         The operating system installed on the system is Red Hat Enterprise Linux 5
      </description>
   </metadata>
   <criteria>
      <criterion test_ref="oval:gov.irs.rhel5:tst:10000"
                 comment="Red Hat Enterprise Linux 5 is installed"/>
      <criterion test_ref="oval:gov.irs.rhel5:tst:10001"
                 comment="Installed operating system is part of the unix family"/>
   </criteria>
</definition>
```

Z tohto popisu je vidieť, že konfiguračné pravidlo jasne určuje, aký typ systému je pre dané pravidlo vyžadovaný. V tomto prípade je to Red Hat Enterprise Linux 5. Ďalej pravidlo obsahuje referenciu na záznam CPE *„cpe:/o:redhat:enterprise_linux:5”* a kritéria, ktorý musí konfiguračné pravidlo spĺňať, aby bolo vyhodnotené ako splnené. Pre jednoduchosť budeme uvažovať iba prvé kritérium, ktoré hodnotou `test_ref` odkazuje na nasledujúci XML element:

```
<rpminfo_test check_existence="at_least_one_exists" comment="redhat-release is version
              5" version="1" id="oval:gov.irs.rhel5:tst:10000" check="at least one"
              xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
   <object object_ref="oval:gov.irs.rhel5:obj:10000"/>
   <state state_ref="oval:gov.irs.rhel5:ste:10000"/>
</rpminfo_test>
```

Z hore uvedeného kódu je vidieť, že jazyk OVAL popisuje splnenie kritérií testami, ktoré sú vyhodnotené porovnaním testovaného elementu `object` a jeho požadovaným stavom `state`. Tieto dva elementy obsahujú informácie o objektoch, ktoré sú testované a spôsobe ako ich testovať. Nasledujúci kód obsahuje oba elementy, ktoré boli hore odkazované.

```
<rpminfo_object version="1" id="oval:gov.irs.rhel5:obj:10000"
                xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
   <name>redhat-release</name>
</rpminfo_object>
<rpminfo_state version="1" id="oval:gov.irs.rhel5:ste:10000"
               xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
   <version operation="pattern match">^5[^[:digit:]]</version>
</rpminfo_state>
```

V tomto prípade sa jedná iba o jeden testovaný objekt a to „redhat-release”, ktorý musí spĺňať regulárny výraz „^5[^[:digit:]]”. V prípade, že ho spĺňa, je kritérium zo začiatku príkladu splnené. Jazyk OVAL ďalej obsahuje referencie na získanie hodnoty týchto objektov a týmto spôsobom testuje systém na dané pravidlá.

3.2 ORML

OVAL Remediation Language je adept na nový štandard SCAP, ktorý je v projekte potrebný pri zabezpečovaní systému. Názov podkapitoly nie je schválne nazvaný „Štandard ORML", pretože do dnešného dňa, kedy píšem túto prácu, nebol inštitúciou NIST prijatý a momentálne existuje iba návrh (draft) prvej verzie. ORML by mal byť veľmi podobným jazykom ako OVAL, založeným na rovnakých princípoch s podobnou sémantikou. Rozdiel je len v tom, že kým jazyk OVAL popisuje, aké stavy by mali mať rôzne objekty popisujúce konfiguráciu systému a ako tento stav zistiť, tak jazyk ORML popisuje, ako tento stav dosiahnuť. Je totiž veľmi rozdielne, keď napríklad zisťujeme, či v systéme existuje nejaký balík určitej verzie a keď potrebujeme zabezpečiť, aby daný balík na systéme v tej verzii bol. V prvom prípade ide o jednoduché zavolanie príkazu na zistenie verzie programu a porovnanie so žiadúcim stavom, kým v druhom prípade ide o nainštalovanie tohto balíka a v prípade, že už existuje, jeho aktualizáciu. Doteraz nie je úplne jasné, či tento jazyk bude mať úspech alebo budeme musieť použiť inú cestu na zabezpečenie systému.

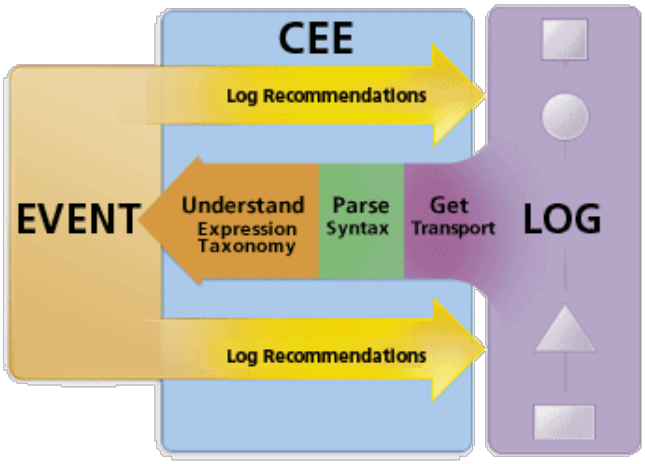
3.3 Projekt OVALDI

Interpret štandardu OVAL [8] implementovaný firmou MITRE. Ovaldi je open-source nástroj, ktorý bol vytvorený za účelom prezentovať spôsob, akým môžu byť z jedného počítača zozbierané informácie za účelom testovania a vyhodnotenia OVAL definícií pre danú platformu a zobrazenia výsledkov testov. Nástroj OVALDI mal byť použitý ako nástroj na interpretáciu štandardov pri diagnóze klientov. Ďalej sa javilo ako výhodné použiť interpret OVALDI pri zabezpečovaní klientov určitým štandardom, ale dlhší čas nebolo známe, či existuje aspoň jedna zásada, ktorú je jednoduché kontrolovať systémom OVALDI, ale ktorá je definovaná tak, že nie je možné usúdiť, či daná zásada je ľahko zabezpečiteľná. Dnes vieme, že takáto zásada existuje a OVALDI nemôže byť použitý na zabezpečovaciu časť systému Lock-Down.

Ako už bolo spomenuté, OVALDI mal byť v projekte Lock-Down použitý na diagnózu, teda testovanie klientov. Vo fáze návrhu systému Lock-Down bol však tento prístp zamietnutý, ako nevýhodne riešenie. Toto rozhodnutie je podložené skutočnosťami, že OVALDI používa zastaralú verziu štandardu OVAL a po diskusii s vývojarmi inštitúcie G2 sme sa rozhodli znovu preimplementovať OVAL do knižnice open-scap. Tá v konečnom dôsledku nahradí OVALDI v plnom rozsahu a poskytne zázemie na prácu so štandardom OVAL.

3.4 Projekt Augeas

Projekt Augeas [7] sa zaoberá vývojom nástroja, ktorého účelom je vytvorenie vrstvy medzi konfiguračnými súbormi a programami, ktoré pristupujú k týmto konfiguračným súborom. Augeas prichádza s abstrakciou formy, akou je daný konfiguračný súbor písaný a programátor pri riešení problému ako pracovať s konfiguračným súborom môže použiť Augeas, ktorý mu namiesto súboru poskytne obsah súboru rozložený do prehľadného stromu. Ako som spomenul v semestrálnej práci [13], dlho sa nevedelo, či bude Augeas v projekte Lock-Down použitý kvôli výkonnostným charakteristikám. V momentálnom štádiu je program Augeas z projektu úplne vylúčený a ako náhrada slúži knižnica open-scap, ktorá ponúka možnosť parsovania OVAL definičných súborov do dátových štruktúr. Postupne bude implementovaná funkcia na parsovanie XCCDF súborov a na ostatné súbory, ako konfiguračné súbory Lock-Downu bude použitá niektorá zo štandardných knižníc pre jazyk python.



Obrázok 3.7: CEE architektúra

3.5 Štandard CEE

Common Event Expression [10] je štandard, ktorý nespadá pod štandardy SCAP, ale pre projekt open-scap je veľmi dôležitý. Prínosom štandardu CEE je ucelené definovanie syntaxe zápisu udalostí, ktoré nastanú v systéme. Vznik tohto štandardu podnietila hlavne neucelenosť syntaxe správ programov, ktoré generujú systémové logy. CEE sa snaží popísať reprezentáciu udalostí, ich interpretáciu a vzájomnú komunikáciu v logoch.

Na obr. 3.7 je zobrazená architektúra CEE zobrazujúca proces od vzniku udalosti („Event"), kedy sa s odpovedajúcimi detailami vygeneruje CCE a ten sa pošle prijímateľovi („Log"). Po obdržaní („Get Transport") správy, prijímateľ správu sparsuje podľa syntaktických pravidiel a posledným krokom je porozumenie problému („Understand") podľa „Expression taxonomy", ktorá môže byť definovaná tak, aby jej porozumel napríklad človek (vygenerovanie reprezentácie informácie do čitateľnej podoby). Pre lepšie pochopenie uvediem príklad: Pri prenose logov protokolom založeným na SOAP[5], je na prenos použitý HTTP protokol, ako syntax je použitá XML schéma a taxonómia[6] je použitá „Common taxonomy", čo je bežné členenie objektov do stromu (klasické XML, kde elementy tvoria strom). Ak odosieľateľom označíme knižnicu open-scap, ktorá vygeneruje správu obsahujúcu informácie o zmene konfigurácie systému, tá sa následne pošle prijímateľovi, ktorým môže byť nejaký sofistikovaný logovací nástroj. Tento nástroj môže ďalej podľa pravidiel určiť typ a závažnosť správy a vykonať príslušnú operáciu (pochopenie problému).

Účel, ktorý má spĺňať CEE v tomto projekte je práve reprezentácia udalostí generovaných knižnicou open-scap a komunikácia medzi knižnicou a externými logovacími nástrojmi. Na tento účel existujú štandardy ako *Intrusion Detection Message Exchange Format (IDMEF)* a *WebTrends Enhanced Log File (WELF)*, avšak bol zvolený práve štandard CEE pre jeho rozsah, pokrývajúci kompletnú podporu narábania s udalosťami a fakt, že štandard zapadá k SCAP štandardom a o jeho vývoj sa stará MITRE Corporation.

Avšak pre malú prioritu bude implementácia tohto štandardu odložená na neurčito.

[5]Simple Object Access Protocol - nástupca XML-RPC, tvorí základnú vrstvu komunikácie medzi webovými službami a definuje syntax a sémantiku uloženia objektov na bázi XML

[6]Taxonómia je reprezentácia kopmonent a ich vzájomné väzby. Najbežnejšou taxonómiou je hierarchické rozdelenie organizmov do stromu. V operačných systémoch to môže byť napríklad organizácia súborového systému

3.6 Štandard STIG

Štandard STIG (*Security Technical Implementation Guides*) obsahuje zoznam zásad bezpečnostnej politiky. Obsahuje napríklad politiku zabezpečenia v registrácii programov, kryptografii, autentifikácii, vlastnostiach hesiel a auditu. Jeho plné znenie je možné nájsť na internetovom portáli `http://iase.sida.mil`.

Príklad konfigurácie (politiky zabezpečenia) minimálnej dĺžky hesla 15 znakov:

(APP3320.1: CAT II) The Designer will ensure the application has the capability to require account passwords having a minimum of 15 alphanumeric characters in length.

Takto vytvorený štandard (ako vidíme, ide skôr o „slovné" definovanie konfiguračných pravidiel) je nutné previesť do definičného súboru, ako je napríklad súbor definícií OVALu. Prevodom získame strojom spracovateľnú špecifikáciu – odporúčané hodnoty konfigurácie systému a to je presne vstup programu Lock-Down.

Tento štandard nespadá priamo pod vývoj projektu Lock-Down a do práce som ho zahrnul, aby som poukázal na fakt, že už existujú štandardy, ktoré definujú politiky zabezpečenia systému pomocou konfigurácií. Prevodom týchto politík je možné štandardy spracovávať strojovo a vyhnúť sa nutnosti zásahu človeka.

Kapitola 4

Analýza

V tejto kapitole bude priblížená analýza projektu, jej proces, neštandardné kroky a jednotlivé iterácie. Uvediem a vysvetlím hlavné požiadavky, ktoré boli kladené na výsledný systém, popíšem analytické triedy, architektúru projektu a v závere bude zhrnutý analytický model s diagramom.

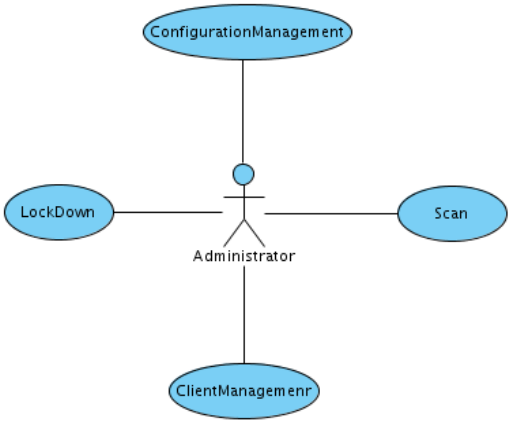
Analýza je rozdelená na dve základné časti. Prvou z nich je backend, ktorý sa skladá z knižnice open-scap a príslušného programu na jej obsluhu a druhou časťou je frontend, ktorý sa skladá z grafického užívateľského rozhrania. Analýza musí s týmito dvoma časťami samozrejme pracovať ako s celkom, pretože frontend je priamo závislý na backende a naopak backend by sa dal chápať ako časť frontendu. Analýza prebiehala navrhnutím kompletného systému, bez oddelenia frontendu a backendu a až v neskoršom štádiu bolo navrhnuté rozdelenie, ktoré zlepšuje bezpečnosť systému ako celku a pridáva možnosť zabezpečovať viac systémov centralizovaným prístupom bez nutnosti mať na každom klientovi nainštalovaný frontend.

Následné podkapitoly analýzy budú v nadväznosti na postup analýzy projektu podľa metodiky UP[1] a to počnúc špecifikáciou požiadaviek a končiac analytickým modelom. Ďalšia kapitola, ktorá priamo nadväzuje na analýzu vychádza z tohto modelu a venuje sa návrhu nástroja Lock-Down a knižnice open-scap.

4.1 Požiadavky

Prvou podkapitolou ako aj prvým krokom analýzy je špecifikácia požiadaviek na systém. Táto špecifikácia prichádza vždy zo strany zákazníka. Keďže zadanie práce nebolo vytvorené zákazníkom, túto úlohu splnilo vedenie tímu Security standards firmy Red Hat. Základnou požiadavkou na systém bolo vytvorenie nástroja s grafickým užívateľským rozhraním, ktorý by dokázal podať užívateľovi informácie o existujúcich konfiguráciach systému a závažnosti ich porušenia. Medzi tieto konfigurácie by sa mali postupne zaradiť aj CVE zraniteľnosti, prípadne iné využiteľné databázy. Ďalej by mal systém ponúknuť základnu prácu s tymito informáciami ako je uloženie, načítanie a výber konfigurácií. Nástroj nebude užívateľovi poskytovať upravovanie konfiguračných pravidiel, pretože je to neželané správanie, ktoré by mohlo viesť k zníženiu bezpečnosti nástroja a systémov, ku ktorým nástroj pristupuje a zároveň existencia štandardov, ktoré popisujú túto konfiguráciu zamedzuje ich upravovanie. V ďalšej fáze by mal systém dokázať komunikovať s inými počítačmi, poskytnúť užívateľovi správu týchto počítačov, zber informácií z ich testovania a aplikovanie zabezpečovacích pravidiel.

[1] Unified Process alebo Unified Software Developement Process je iteratívny proces vývoja softvéru



Obrázok 4.1: Biznis model prípadov použitia

Nástroj by mal samozrejme spĺňať vysoké požiadavky na bezpečnosť, pretože program, ktorý nemá obmedzenie v prístupe k systému a zároveň pracuje ako bezpečnostná aplikácia, je atraktívnym cieľom potencionálnych útočníkov.

Zverejnenie kompletných detailov o podpore a funkčnosti programu pred jeho dokončením by mohlo mať za následok zvýšenie konkurencie v oblasti, ktorú chce projekt pokryť a preto ich v práci neuvádzam.

4.1.1 Aktéri

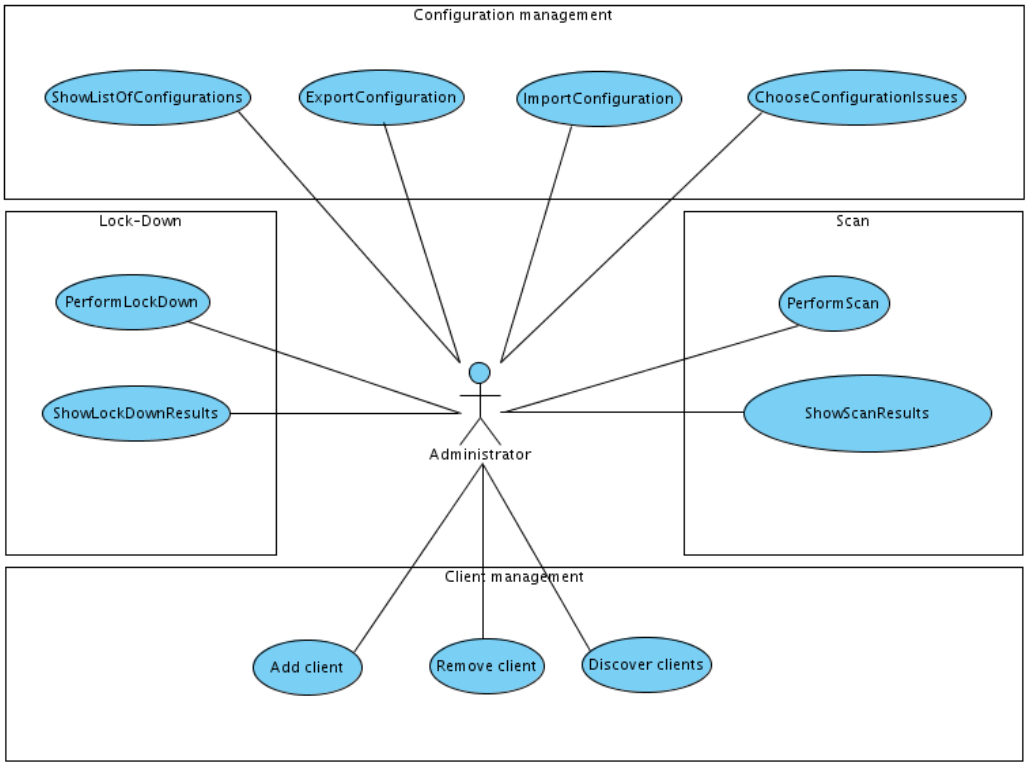
Podľa špecifikácie požiadaviek by malo ísť o systém, ktorý bude ovládaný len administrátorom a iní užívatelia nebudú mať k programu prístup. Systém ďalej nebude vykonávať žiadnu autonómnu činnosť, ani zasahovať do iného systému, kde by bolo nutné pridávať nezávislého aktéra. Pri vytváraní modelu prípadu použitia budeme uvažovať len o jednom aktérovi.

4.1.2 Model prípadov použitia

Pri analýze prípadov použitia som vychádzal z definovania tzv. „biznis modelu prípadov použitia", kde zákazník, ktorý zadal projekt, vidí len hrubý náčrt funkčnosti systému ako celok a následne sa pridávajú funkcie systému do takej miery, kým celý „use-case diagram" popisujúci model nie je komplentný. Ako posledný krok sa vyčlenia subjekty tohto modelu a celý diagram sa minimalizuje tak, aby stále pokrýval celú oblasť funkčnosti. Na obr. 4.1 je prvý use case diagram, ktorý určuje práve tú obchodnú časť, ktorá zaujíma zákazníka.

Posledný model prípadov použitia v rámci analýzy (obr. 4.2) pokrýva všetky požiadavky zákazníka na výsledný systém. Požiadavky sú zoskupené do subjektov, ktoré rámcujú oblasť pôsobnosti funkcií. V ďalšom kroku bude navrhnutá ukážka grafického užívateľského rozhrania.

V každej iterácii modelovania systému pomocou prípadov použitia bol model predvedený zákazníkovi a tým sa docielilo postupné dolaďovanie požiadavok pod vedením zákazníka. Tento prístup odstaňuje nedostatky s pochopením požiadavok, kedy sa pri návrhu, ktorý vychádza z analýzy zisťuje, že požiadavky sú nepresné alebo nevyhovujúce.



Obrázok 4.2: Model prípad použitia

4.1.3 Stanovenie funkčných, nefunkčných a kritických požiadaviek

Požiadavky rozdelím jednoduchou taxonómiou na funkčné a nefunkčné, z funkčných požiadaviek potom určím takzvaných „20% kritickej funkcionality"[2], ktorými sa označuje časť systému, ktorú zákazníci najviac využívajú. Správne definované a rozdelené požiadavky vytvárajú profil systému, ku ktorého cieľu sa snažíme dostať. Vymedzenie úzkeho okruhu kritickej funkcionality nám ponúkne cestu, na ktorej sa zameriame naozaj len na tú časť, ktorú užívateľ bude najviac využívať (tento spôsob myslenia sa osvedčil úspešným korporáciam ako je napr. Google).

Funkčné požiadavky

Funkčné požiadavky systému je formulácia toho, čo by mal daný systém vykonávať. V našom prípade je to nasledujúci zoznam základných funkčných požiadaviek.

1. Systém bude zobrazovať užívateľovi zoznam konfigurácií.
2. Systém bude poskytovať funkcie na ukladanie, načítanie, prezeranie konfigurácií, ich filtráciu.
3. Systém musí vedieť testovať vybraný počítač na dané konfigurácie.
4. Systém bude vedieť zabezpečiť vybraný počítač podľa pravidiel určitých konfigurácií.
5. Systém bude poskytovať spôsob manipulácie so vzdialenými počítačmi.

[2] V skutočnosti nemusí ísť o 20% funkcionality, toto číslo vychádza z myšlienky určiť kritickú časť aplikácie a je podložené štatistickými informáciami využívania funkcionality programu od výrobcu softvéru, ktorého meno sa mi nepodarilo zistiť.

6. Systém bude zobrazovať históriu akcií vykonaných na danom počítači.

7. Systém bude spolupracovať s inými systémami na manažment počítačov ako napr. IPA.

Nefunkčné požiadavky

Nefunkčné požiadavky sú obmedzujúce podmienky daného systému

1. Celý systém musí byť prenositeľný.

2. Komunikačná časť systému musí byť dostatočne zabezpečená.

3. K systému sa budú môcť dostať len vybrané osoby.

4. Grafická časť systému bude napísaná v jazyku Python.

5. Výkonná časť systému bude napísaná v jazyku C.

6. Systém musí byť postavený na aktuálnych verziách štandardov SCAP.

Oblasť kritickej funkcionality

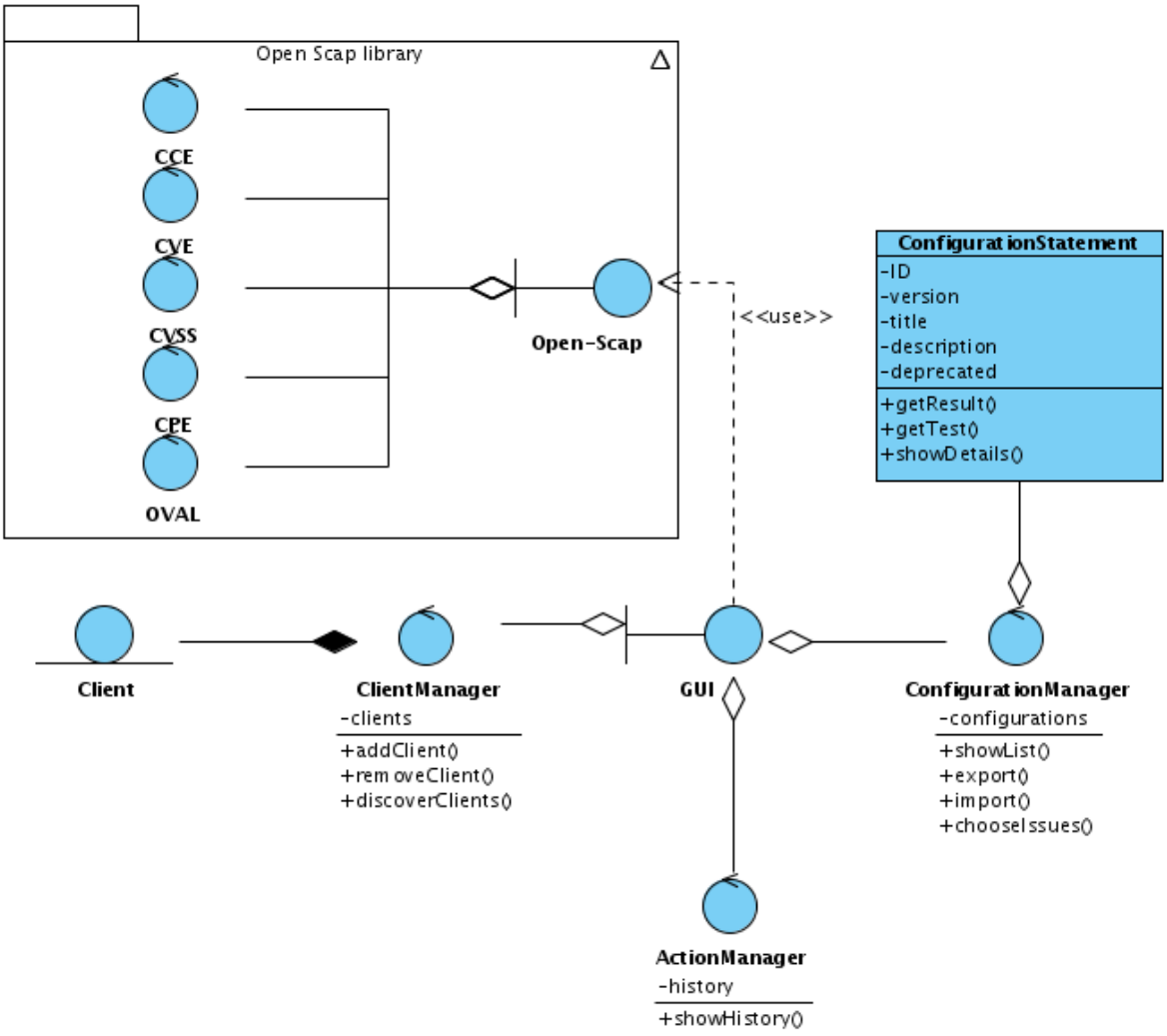
Kritická funkcionalita je okruh funkcií, ktoré užívateľ považuje za najdôležitejšie a ktoré bude na systéme najviac využívať. Princípom je určiť čo najmenšiu časť systému, ktorá je pre zákazníka najdôležitejšia. To znamená, že do tejto oblasti určite zahrnieme zabezpečenie a testovanie systémov, pretože je to časť, ktorú bude užívateľ využívať vo veľkej miere. Kým práca s počítačmi (klientami), ako ich pridávanie a odoberanie do tejto oblasti nespadá, pretože predpokladáme, že užívateľ si prvý krát vytvorí zoznam počítačov, ktoré bude neskôr sledovať a tento zoznam nebude meniť príliš často.

Kritické funkcie tohto projektu sa z hore uvedeného dôvodu zúžili na zabezpečenie a testovanie systémov a základnú prácu s konfiguráciami. V systéme Lock-Down je oblasť kritickej funkcionality zúžená na vyvolanie akcie nad určitými konfiguráciami a zobrazenie výsledkov tejto akcie. V skutočnosti je to menej ako spomínaných 20%. V knižnici open-scap je toto číslo omnoho vyššie a oblasť bude zasahovať hlavne časť implementujúcu štandard OVAL.

4.2 Analytické triedy

V tejto časti sa zameriam na tvorbu analytického modelu. Prvým krokom k jeho vytvoreniu je nájdenie analytických tried. Artefaktami pre túto činnosť sú biznis model, ktorý je reprezentovaný prvotnými požiadavkami od zákazníka, model požiadaviek, ktorý sme vytvorili v časti 4.1.3, model prípadov použitia vidíte na obrázku 4.2 a posledným artefaktom je popis architektúry (viď prílohu A).

Hľadanie analytických tried prebiehalo metódou analýzy podstatných mien a slovies a bola doplnená pomocou stereotypov metodiky RUP. Prvá zo spomínaných metód, analýza podstatných mien a slovies, je jednoduchou metódou analýzy textu, ktorá sa zameriava na podstatné mená a určuje ich význam ako samostatných tried a slovesá, ktoré určujú zodpovednosť týchto tried. Vstupným textom uvedenej metódy boli spomínané artefakty ako požiadavky, prípady použitia a špecifikácia architektúry. Výstupom analýzy sú triedy *GUI*, *ConfigurationManager, ConfigurationStatement, ActionManager, ClientManager* a *Client* (viď obr. 4.3).

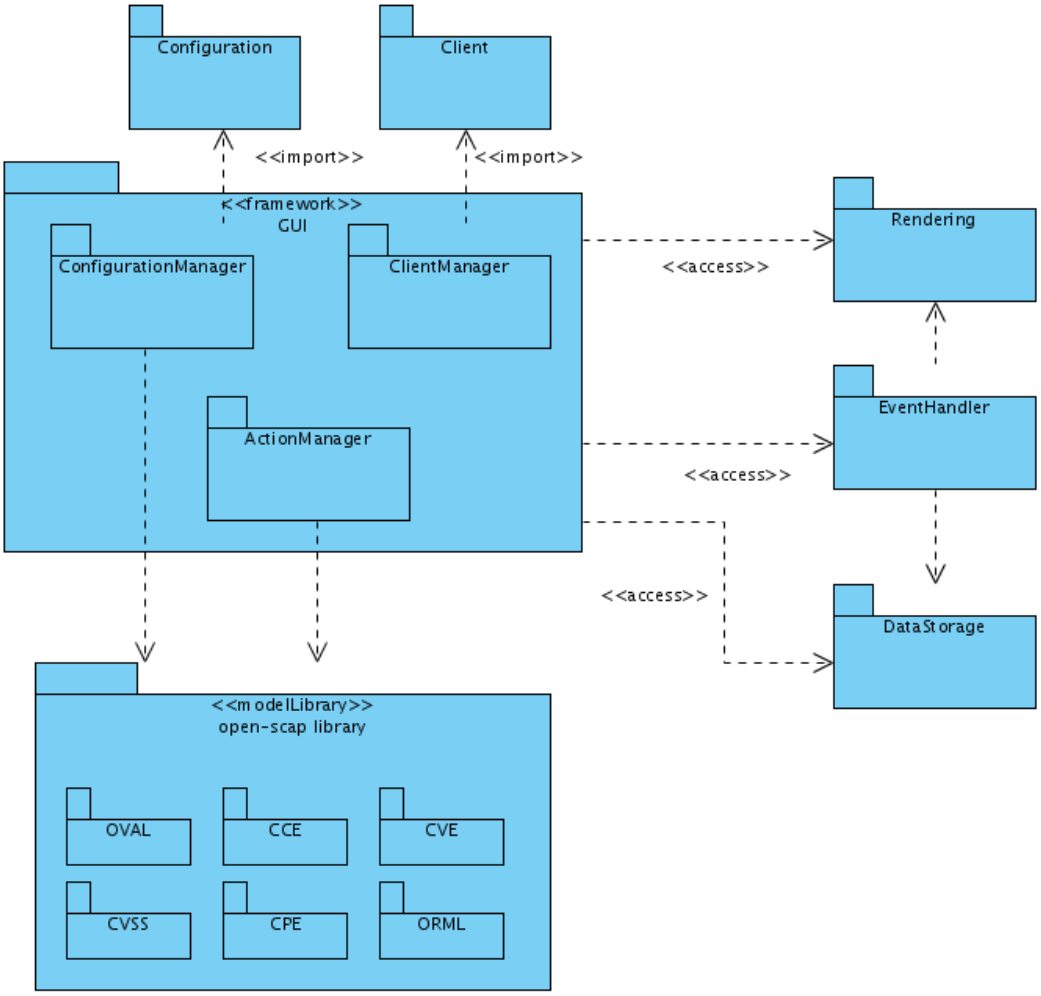


Obrázok 4.3: Diagram analytických tried

Druhou metódou na vyhľadávania analytických tried bola metodika vyhľadávania pomocou steretypov RUP[3].

Výsledný diagram je vidieť na obr. 4.3. Systém je rozdelený na tri časti, kde každá má určitý okruh zodpovedností. Tie sú mapované na prípad použitia. Na prvý pohľad sa môže zdať, že model je vytvorený funkčnou dekompozíciou než objektovou orientovanou analýzou. Tento jav je možné jednoducho vysvetliť tým, že analýza prebiehala neštandardnými krokmi, ktoré sú spomenuté v závere analýzy (viď kapitola 4.5). Druhá časť diagramu (na obr. 4.3), zobrazená ako podsystém s názvom „Open Scap library" je zapúzdrená knižnica, ktorá je súčasťou projektu a ktorá poskytuje vrstvu medzi cieľovým počítačom (klientom) a užívateľským rozhraním a vykonáva všetky operácie na strane klienta. Pre viac informácií viď kap. 5.3.2.

[3]Rational Unified Process



Obrázok 4.4: Diagram balíčkov

4.3 Analýza balíčkov

Analytické balíčky umožňujú tvorbu zrozumiteľných a jasne štrukturovaných modelov, keďže umožňujú zoskupovať predmety s tesnými semantickými väzbami. V modeli, v ktorom rôzne balíčky zaisťujú rôzne aspekty funkcie systému, môžu analytické balíčky tvoriť semantické hranice [16]. Analýza analytických balíčkov nám poskytne lepší pohľad na architektúru systému a naviazanosť jednotlivých objektov počas tvorby návrhu programu. Keďže hierarchia týchto balíčkov tvorí hierarchiu menných priestorov, vychádzajú z tohto modelu taktiež závislosti medzi triedami.

Pri hľadaní balíčkov bol opäť využitý model prípadov použitia, aby sme sa čo najviac priblížili pohľadu biznis modelu.

Ako je vidieť na diagrame 4.4, samotné jadro frontendu je tvorené troma oddelenými systémami na manažment akcií, klientských staníc a konfigurácií. Jadro ďalej spolupracuje s renderovacím modulom, ktorý je nezávislý na použití vykresľovacej technologie (GTK, QT, webové technológie). Celá logika užívateľského rozhrania je totiž implementovaná v module „ActionManager" a komunikácia cez jadro systému je naimplementovaná tak, aby funkcie, ktoré sú z jadra volané, mohli byť implementované ako pre GTK rozhranie, tak aj pre iné, napríklad webové rozhranie (samozrejme je možné použiť aj iné technológie).

4.4 Slovník pojmov

Slovník pojmov je veľmi dôležitý artefakt v riadení projektu. Slovník nám pomáha ustáliť terminológiu použitú v projekte a zlepšiť tým tímovú aj externú komunikáciu. V nasledujúcom texte vysvetlím niektoré dôležité pojmy, ktoré sa zdali byť v procese analýzy problematické:

1. Konfigurácia: tento pojem je prebraný z anglického „configuration statement", ktorý sa tiež niekedy uvádza ako „configuration issue". Ide o konkrétnu konfiguráciu daného systému a značí to jeden prípad nastavenia systému, ktorý je popísaný CCE identifikátorom. Ako príklad môžeme uviesť konfiguráciu: „Nastavenie minimálnej dĺžky hesla na 10 znakov".

2. Daemon: pojem opäť prebratý z anglickej terminológie, ktorý označuje nepretržite bežiaci proces, zvyčajne počúvajúci na vonkajšom porte. V terminológii tohto projektu pôjde o programovú časť, ktorá má za úlohu sprostredkovať komunikáciu medzi knižnicou na klientskom počítači a užívateľským rozhraním na vzdialenom počítači. Daemon neobsahuje knižnicu, iba volá funkcie knižnice.

3. Knižnica: množina funkcií programu s vlastným API, ktorý nie je samostatne spustiteľný. Knižnicou v tejto práci označujeme projekt open-scap. Knižnica poskytuje API k funkciám pre prácu so štandardmi CCE, CVE, CEE, CPE, CVSS, OVAL, ORML a XCCDF.

4. Backend: súhrnné pomenovanie knižnice a daemona je v práci použité hlavne z historických dôvodov používania slova v semestrálnom projekte. V praxi sa tento pojem objavuje len výnimočne, pretože spôsoboval v komunikácii značné problémy. Pre ujasnenie sa momentálne používajú hlavne slová „knižnica" a „daemon".

5. Frontend: grafické užívateľské rozhranie poskytujúce užívateľovi jednoduchý prístup k funkciám backendu. Označenie „frontend" sa v praxi vyskytuje len výnimočne. Pojem bol nahradený slovným spojením „projekt Lock-Down"

6. Sondy: z anglického slova „probs", pomenovanie jednoduchých binárie využívaných knižnicou na prístup k funkciám systému.

Slovník pojmov nie je kompletný, pretože jeho rozsah je nad rámec tejto práce. Počas analýzy projektu nebol nikdy úplne skompletizovaný, pretože ďalej nenastávali problémy v komunikácii a jeho dokončenie nebolo potrebné.

4.5 Záver analýzy

Počas analýzy boli najdôležitejšie procesy dôkladne prediskutované s vedením projektu na strane firmy Red Hat. Projektové konzultácie ohľadom analýzy sa konali v pravidelných intervaloch každé dva týždne počas troch mesiacov, kedy sa projekt dostal do štádia implementácie. Neštandardným krokom bolo hlboké prelínanie analýzy s návrhom. Vedenie projektu si počas analýzy vyžiadalo návrh GUI a svoje požiadavky smerovalo hlavne na jeho úpravu a nie na funkčnosť. Jedným z príkladov by som mohol uviesť nasledujúcu situáciu.

Po prezentácii v poradí tretieho návrhu GUI vedeniu bola nastolená otázka, či tlačítko na vykonanie akcie je dôležité mať v okne pre správu konfigurácií, ako aj v okne pre správu klientov, alebo iba v detailoch daného klienta. Odpoveďou v danej chvíli bolo rozhodnutie, že tlačítko bude viditeľné len v dialógovom okne s detailami klienta, čo neskôr

zásadne ovplyvnilo definíciu požiadaviek, kedy sa zmenila požiadavka z „systém musí vedieť zabezpečiť daného klienta podľa vybranej konfigurácie" na „systém musí vedieť zabezpečiť vybraného klienta podľa určitých konfigurácií" a výber konfigurácie posunulo až za krok výberu klienta (musíme si uvedomiť, že v aktuálnej verzii neexistuje výber konfigurácie). Tento rozdiel sa síce môže zdať z prvého pohľadu banálny, ale pokiaľ sa lepšie pozrieme na diagram balíčkov (obr. 4.4), tak medzi balíčkom „ClientManager" a balíčkom „ConfigurationManager" nie je žiadna väzba, čo prináša zjednodušenie systému. V neskoršej verzii táto situácia nenastane, pretože užívateľ si bude môcť vybrať medzi viacerými množinami konfigurácií, čo nie je v aktuálnej verzii prípustné, ale jasne vidíme určitú výhodu špecifikácie požiadaviek na základe grafického výzoru aplikácie.

4.5.1 Ďalšie kroky

V tomto okamihu by sa dalo povedať, že analýzu projektu máme hotovú. Pravdou je, že v životnom cykle projektu nastávajú zmeny v špecifikácii požiadaviek a určitá časť analýzy sa vykonáva pravidelne, tak ako sú pravidelné konzultácie so zákazníkom.

Výstupom analýzy je špecifikácia požiadaviek, diagram prípadov použitia, diagram analytických tried a diagram balíčkov. Z týchto artefaktov bude v nasledujúcej kapitole čerpať návrh projektu, kde sa vytvorí diagram tried, architektonický model aplikácie a návrh GUI.

Kapitola 5

Návrh projektu

Kapitola popisujúca proces návrhu programu vychádza z predchádzajúcej kapitoly 4 – Analýza. V tejto časti práce rozoberiem štruktúru a architektúru navrhovaného projektu a niektoré zaujímavé časti návrhu. Oproti predchádzajúcemu návrhu zo semestrálnej práce [13] sa zmenil návrh backendu, čiže vrstvy medzi užívateľským rozhraním a samotným počítačom. Oproti analýze projektu, ktorá sa sústreďuje hlavne na požiadavky a vôbec neprihliada na zvolenú platformu, v návrhu je kladený dôraz na poskytnutie kompletného zázemia pre implementačnú fázu, zohľadnenie parametrov cieľovej architektúry a výber najefektívnejších technológií. V ďalšom texte rozdelím projekt do základných komponentov z logického a fyzického pohľadu a sústredím sa na návrh jednotlivých častí. Pri návrhu projektu zastávam osobný názor, že čo najdetailnejšie rozdelenie na komponenty (hlavne funkčnou dekompozíciou) a určenie ich závislostí je najlepšou cestou k návrhu kvalitného softwaru.

5.1 Logické rozdelenie

Celý systém na zabezpečenie operačného systému sa dá rozdeliť podľa dvoch pohľadov na aplikáciu. Prvým z nich je z pohľadu logického rozdelenia.

1. časť, **scanner**, bude slúžiť ako bezpečnostný nástroj na diagnózu systému. Bude schopný podľa vybraného štandardu alebo kombinácii jeho pravidiel vykonať diagnózu jedného alebo viacerých počítačov. Ďalším možným rozšírením je grafické zobrazenie informácií z daných počítačov v určitom časovom rozmedzí a zobrazenie štatistík.

2. časť, **Lock-Down** je systém, ktorý výlučne slúži na zabezpečenie počítača na princípe odstránenia potenciálnych zraniteľných miest. Čo a ako sa zabezpečí určuje súbor pravidiel daného štandardu importovaného v programe pomocou definičného súboru.

Tieto dve logicky oddeliteľné časti budú vo výslednom systéme úzko spolupracovať. Jednou možnou spoluprácou sa javí zistenie zraniteľného miesta tzv. scannerom a následne jeho odstránenie, resp. zabezpečenie systému (ang. lock-down) pomocou programu Lock-Down.

Druhým pohľadom na projekt je rozdelenie na rozhrania, čo tvorí architektúru systému a vo väčšine prípadov sa jedná o dekompozíciu podľa funkčnosti jednotlivých komponent. Tomuto rozdeleniu je venovaná väčšina textu tejto kapitoly.

5.2 Proces návrhu

Proces návrhu projektu vo svojich začiatkoch vznikal dosť neštandardným spôsobom a preto pokladám za dôležité ho spomenúť. Celý proces je možné rozdeliť do troch krokov, ktoré

je samozrejme možné ďalej deliť podľa času priebežných kontrol zo strany vedúceho tímu Security standards, Steva Grubba a manažéra Karla Wirtha, oboch z firmy Red Hat Inc. v ktorej bol projekt zadaný.

Návrh vychádza z analýzy, kde boli určené a spracované požiadavky, ktoré má projekt v konečnom dôsledku spĺňať. Tento zoznam požiadavok nie je momentálne zverejnený a po určitom čase bude umiestnený na serveri `http://open-scap.org`, ktorá sa stala verejnou a komunitnou základňou pre tento projekt.

Druhým krokom práce na vývoji systému bolo namiesto vytvorenia UML[1] návrhu zvolené iba vytvorenie diagramu prípadov použitia a následné vytvorenie návrhu užívateľského prostredia. Tento krok bol zaujímavým ťahom a myslím, že jeho silnou stránkou bolo ujasnenie, ktoré časti budú musieť byť zviditeľnené užívateľovi a logické rozdelenie aplikácie. Taktiež tento krok pomohol k dokončeniu a spresneniu podmienok z kroku analýzy. Nevýhodou tohto postupu, ktoré bolo doposiaľ postrehnuteľné, je predĺženie času stráveného na návrhu, ale táto časť sa len veľmi ťažko posudzuje bez možnosti porovnania s iným rovnako veľkým projektom. Tento postup zapríčinil, že návrh a analýza sa skoro úplne prekrývali.

Tretím krokom je klasický UML návrh vyplývajúci zo štandardného postupu pri návrhu projektu. Tieto diagramy budú ukázané v nasledujúcich kapitolách návrhu.

Ďalším krokom je implementácia knižníc a podporných nástrojov súbežne s implementáciou užívateľského rozhrania. Tieto dve časti sú logicky rozdelené do viacerých pracovných blokov. Proces rozdelenia na implementačné kroky a ich kontrola sú uvedené v kapitole 6.

5.3 Architektúra projektu

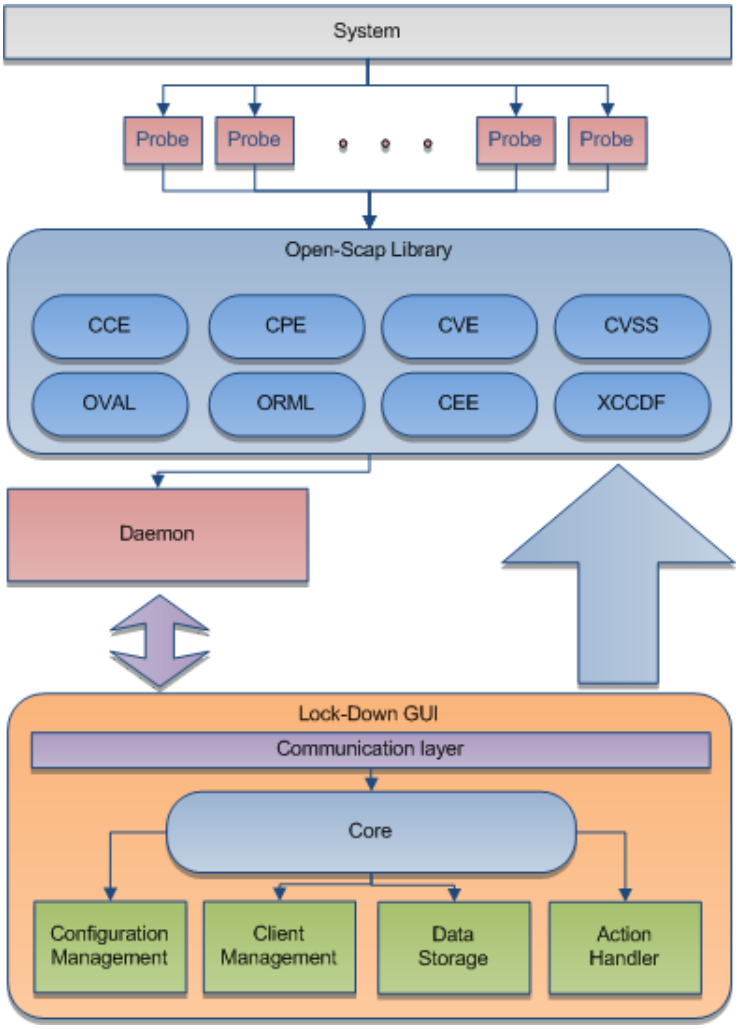
Architektúra projektu ponúka pohľad na fyzické rozčlenenie systémov do celkov a ich vzájomné prepojenie.

Architektúra je rozdelená do dvoch hlavných častí. Prvou z nich je frontend – grafické užívateľské rozhranie, ktoré komunikuje pomocou komunikačného rozhrania, ktoré je taktiež vo výsledku nezávislé od ostatných častí. Na narábanie s externými zdrojmi (ako sú napríklad definičné súbory) používa knižnicu open-scap, ktorá je zároveň druhou hlavnou časťou, backendom. Backend je systém inštalovaný na klientov, koncové zariadenia, na ktorých bude prebiehať zabezpečenie alebo zbieranie informácií. Knižnica taktiež ponúka zdroje na prácu s definičnými súbormi, ukladanie stavových informácií a prístup k systému. Backend a frontend spolu komunikujú cez už spomenuté komunikačné rozhranie. Ďalšou nezanedbateľnou časťou celého projektu je rozhranie na manažment počítačov. Táto časť nie je v prvej fáze projektu žiadaná, keďže systém má pracovať iba lokálne, avšak návrh grafického rozhrania musí toto rozšírenie zohľadniť. V prvej fáze bude práca s klientami implementovaná ako jednoduchá databáza s možnosťou tzv. „discovery“ alebo objavovania nových klientov na sieti. Neskôr môže byť celá časť nahradená pokročilým systémom manažmentu, akým je napríklad IPA[2] (viď prílohu A).

Na obr. 5.1 je zobrazená základná architektúra projektu Lock-Down a knižnice open-scap. Jej jednotlivé časti sú popísané v nasledujúcich kapitolách. Pre jednoznačnosť popisovaných súčasti len uvediem definíciu farebnej schémy obrázka:

- Prvý šedý pruh na hornej strane obrázka abstrahuje systém, na ktorom sa nachádza nainštalovaná knižnica.
- Červenou farbou sú označené binárne programy. Konkrétne ide o jednoduché súbory ako sú tzv. *sondy* a program daemon, ktorého funkčnosť je uvedená nižšie.

[1] Unified Modeling Language
[2] Identity management for Linux and Unix systems



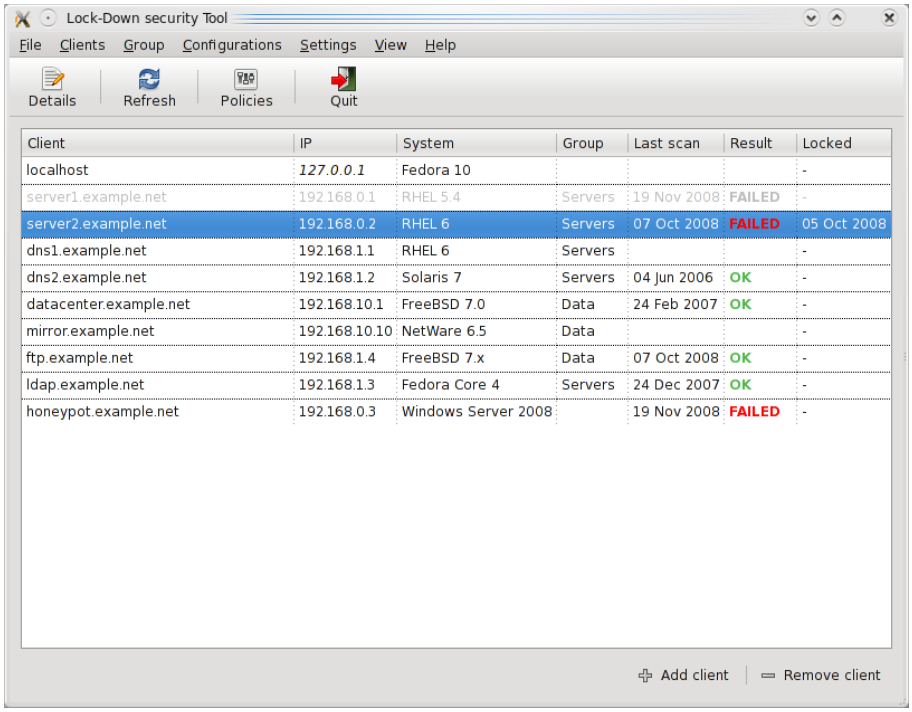
Obrázok 5.1: Pohľad na architektúru systému Lock-Down a knižnice open-scap.

- Modrá farba určuje spojitosť s knižnicou open-scap. Tá sa nachádza hneď pod binárnymi súbormi – probami a modrá šípka značí volania API knižnice.
- Fialová farba komunikačného rozhrania taktiež symbolizuje sieťovú komunikáciu.
- Oranžovou farbou je označené grafické užívateľské rozhranie projektu Lock-Down a zelenou farbou jeho riadiace súčasti.
- Modrá farba použitá na časť Core symbolizuje jadro systému.

5.3.1 FrontEnd

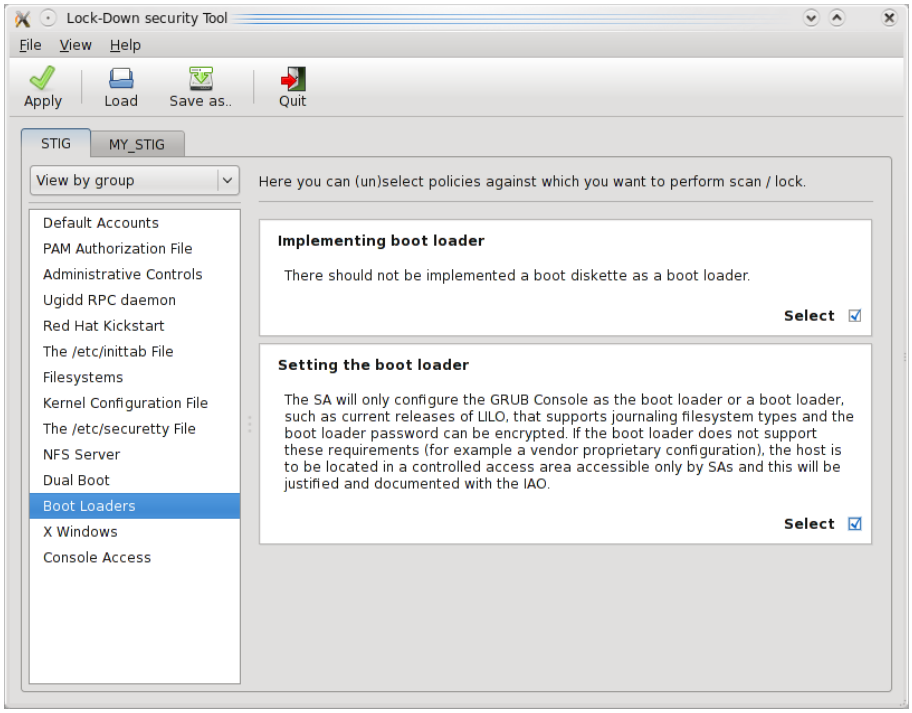
Programovo samostatná jednotka, základ pozostáva z jadra, ktoré je akousi riadiacou jednotkou spájajúcou užívateľské rozhranie, manažment klientov, manažment konfigurácií a rozhranie na komunikáciu s backendom, prípadne s inými nástrojmi. Užívateľské rozhranie je implementované použitím technológií Python a GTK+. Jednou z požiadaviek na jeho implementáciu je nezávislosť renderovacej technológie a preto bola logika programu úplne oddelená od zobrazovacej časti (viac v kapitole 6).

Grafický návrh



Obrázok 5.2: Grafický návrh hlavného okna systému Lock-Down

Grafický návrh je v prvej fáze najdôležitejšou časťou návrhu (viď proces návrhu). Hlavným pilierom je hlavné okno (obr. 5.2), ktoré obsahuje zoznam klientov so základnými informáciami, ako je IP, dátumy poslednej diagnostiky a zabezpečenia klienta a farbou písma je označený status, či je klient dostupný alebo nie. Užívateľ môže z tohto okna pokračovať buď vyvolaním okna na konfiguráciu štandardov (obr. 5.3), kde je možné do určitej miery určiť, ktoré zásady politiky daného štandardu sa budú brať do úvahy alebo kliknutím na



Obrázok 5.3: Grafický návrh okna na editáciu konfigurácií štandardov

klienta zobraziť detailnejšie informácie s možnosťou vybratia akcie pre daného klienta. Z tohto dialógového okna sa po vybratí akcie vyvolá krátky sprievodca, kde po poskytnutí informácií ako štandard, typ akcie a čas vyvolania akcie (okamžitý, alebo naplánovaný), užívateľ zavolá metódu na vykonanie tejto procedúry.

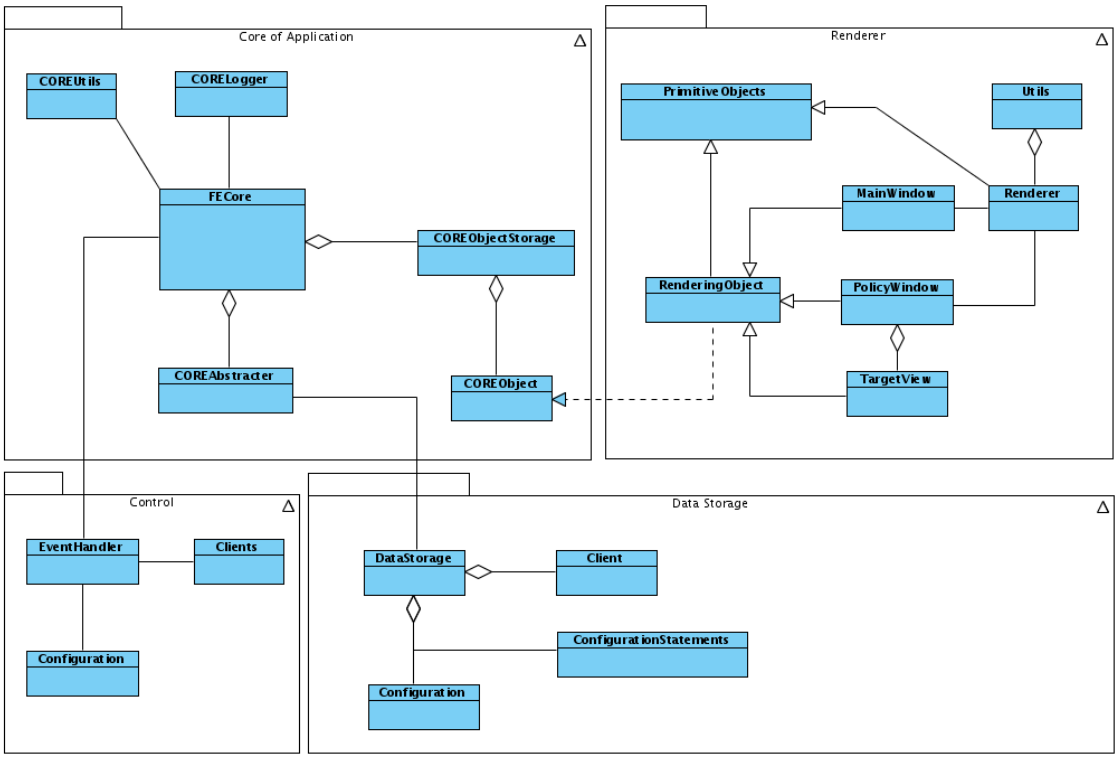
Okno so štandardami, ktoré obsahujú jednotlivé konfigurácie (obr. 5.3) sa skladá z ľavej časti, kde je daný zoznam a z pravej časti sú detaily vybraného záznamu (konfigurácie), ktoré sa zobrazia po kliknutí na niektorú položku zo zoznamu. V hornej lište je roletové menu, ktoré slúži ako prepínač medzi rôznymi spôsobmi zobrazenia zoznamu. Buď je zobrazený popis položky alebo jeho referencia, čo môže byť CCE, CVE alebo CPE.

Architektúra systému Lock-Down

Architektúra projektu Lock-Down (frontendu) je z časti zobrazená na obr. 5.1. Základ programu tvorí modul **Core** (ďalej len jadro aplikácie), ktorý stojí za behom aplikácie, zabezpečuje komunikáciu medzi ostatnými modulmi. Skladá sa z databázy všetkých prítomných objektov, ktoré obsahujú metódy spojené s udalosťami v aplikácii, riadiaci mechanizmus aplikácie a jednoduchého logovacieho algoritmu. Implementačné detaily tohto systému sú popísané v nasledujúcej kapitole.

Druhým komponentom je **Action Handler** (modul Logic), ktorý obsahuje logiku programu a na jeho jednotlivé metódy sú napojené objekty z modulu Core. Keďže logika programu je neoddeliteľne spojená so zobrazovacím systémom a jedna z požiadaviek bola práve nezávislosť použitej technológie, boli všetky objekty, ktoré sú nutne prítomné pri renderovaní a komunikácii abstrahované do jadra aplikácie a logika pristupuje k týmto objektom bez závislosti na ich implementácii.

Pri tomto prístupe vznikol problém volania metód týchto objektov a ich nejednoznačnosti. Preto v module **Renderer** (zastúpený modulom CoreRendererGTK, keďže implicitnou technológiou je GTK+) musí rendrovací mechanizmus implementovať jednotlivé abstraktné



Obrázok 5.4: Diagram tried systému Lock-Down

metódy objektov a tým poskytnúť rozhranie tak, aby nedošlo k spomínaným problémom. Priame implementačné detaily sú opäť spomenuté v nasledujúcej kapitole.

Pri návrhu frontendu bola kladená snaha na úplné oddelenie programovej časti aplikácie a dátovej časti. Postupne boli všetky dáta abstrahované a zoskupené do jedného komponentu **Data Storage**, ktorý obsahuje rôzne objekty ako dáta s vlastnosťami aplikácie, konfiguračné dáta, dáta o klientoch, riadiace dáta a podobne. Tento komponent sa nachádza ako privátny objekt v jadre aplikácie, ktoré ho sprístupňuje implementovanými metódami.

Ďalšími dvoma komponentami sú **Client Management** a **Configuration Management**, ktoré zastávajú úlohu riadiacich modulov pri operovaní s dátami klientských staníc a konfigurácií.

Nad jadrom systému je vybudovaná vrstva **Communication Layer** starajúca sa o komunikáciu medzi užívateľským rozhraním a daemonmi. K tejto vrstve je zo zjavných bezpečnostných dôvodov povolený prístup len cez jadro aplikácie.

Diagram tried

Diagram tried odpovedá architektúre frontendu, ale pohľad na aplikáciu je detailnejší a zameriava sa hlavne na návrhové triedy. V nasledujúcom texte rozdelím systém na päť základných podsystémov, ktoré sa skladajú z tried. Nižšie sú tieto triedy rozpísané spolu so stručnou charakteristikou. Tomuto modelu zodpovedá diagram návrhových tried na obr. 5.4.

- **Jadro aplikácie** - obsahuje jadro, ktoré bude mať na starosti beh aplikácie.
  - `FECore`: trieda obsahujúca základné objekty aplikácie a zabezpečujúca ich vzájomnú komunikáciu.

  - `COORELogger`: trieda, ktorá ma na starosti zobrazovanie ladiacich výpisov užívateľovi. Dá sa nastaviť v konfigurácii aplikácie, kam sa budú tieto výpisy zobrazovať.
  - `COREUtils`: trieda poskytujúca doplnkové statické metódy pre ostatné objekty ako zázemie pre ladiace výpisy a podobne.
  - `COREObjectStorage`: sklad všetkých objektov jadra (viď nižšie)
  - `COREObject`: objekt jadra, ktorý zapúzdruje každý objekt aplikácie a ponúka abstrakciu metód týchto objektov a ľahké ladenie aplikácie.
  - `COREAbstracter`: trieda, ktorý zapúzdruje ostatné triedy mimo jadra a riadi prístup k týmto triedam

- **Riadiaca časť aplikácie** - obsahuje triedu na ovládanie konfigurácií, klientov a na obsluhu signálov v aplikácii, riadi narábanie s dátami a signálmi, aplikáciu riadi jadro aplikácie.

  - `EventHandler`: trieda implementujúca metódy, ktoré sú volané pri odchytení zaregistrovaného signálu. Tento signál je odchytený jadrom aplikácie a to volá objekt triedy EventHandler.
  - `Clients`: trieda obsahujúca metódy na manažment klientov (poznámka: táto funkcionalita bude implementovaná až pri požiadavke na sieťovú komunikáciu).
  - `Configuration`: trieda obsahujúca mety na manažment konfigurácií. Táto trieda momentálne jediná pristupuje ku knižnici open-scap.

- **Dátova časť** - skladá sa z hlavného dátového skladu, pozostavajúceho z jednotlivých klientov a configurácií.

  - `DataStorage`: dátový sklad aplikácie.
  - `Client`*:* dátová položka obsahujúca informácie o danom klientovi.
  - `ConfigurationStatements`*:* dátová položka obsahujúca informácie o danej konfigurácii.
  - `Configuration`: trieda zapúzdrujúca konfiguráciu aplikácie.

- **Renderovacia časť** - ma za úlohu všetky akcie pri vykresľovaní aplikácie. Tieto triedy sú implementované v aktuálnej podobe pre renderovací systém GTK+.

  - `PrimitiveObjects`: obsahuje funkcie pre vykreslovacie primitíva, pri rozdielnych implementáciach táto trieda nemusí existovať.
  - `RenderingObject`: abstraktná trieda ponúkajúca metódy na komunikáciu s jadrom aplikácie.
  - `TargetView`: časť vykresľovania hlavného okna, ktoré má na starosti prácu s konfiguráciami.
  - `PolicyWindow`: dialógové okno aplikácie s prezeraním a prácou s konfiguráciami.
  - `Utils`: trieda poskytujúca doplnkové statické metódy potrebné pri vykresľovaní niektorých objektov.
  - `Renderer`: základná trieda, ktorá je volaná z jadra pri vykresľovaní.
  - `MainWindow`: hlavné okno aplikácie.

Ako je vidieť v časti, ktorá popisuje jadro aplikácie, tak objekty `COREObject` zapúzdrujú všetky objekty aplikácie a tým akýkoľvek prístup k týmto objektom je kontrolovaný a v základnom nastavení aplikácie sa na štandardnom výstupe zobrazujú výpisy o týchto prístupoch. Táto abstrakcia ponúka ďalšie možnosti, ako je napríklad úplná podpora zásuvných modulov a nezávislosť na ostatných častiach aplikácie (je stanovené pevné API).

Ďalšou zaujímavosťou návrhu tohto systému je trieda `COREAbstracter`, ktorá ma za úlohu zapúzdriť triedy a tým prebrať plnú kontrolu od ich vzniku až po ich zánik. Tento prístup ponúkol napríklad možnosť explicitnej notifikácie objektov, ktoré majú na starosti zobrazovať dáta z určitého objektu, pri akejkoľvek zmene týchto dát. To znamená, že ak akákoľvek akcia vyvolá zmenu dát jedného formuláru, je vyvolaná akcia na notifikáciu tohto formuláru, aby obnovil svoj obsah alebo na zmenu dát patrične reagoval. Povinnosťou programátora je len určiť zviazanosť dát a implementovať akciu, ktorá bude vyvolaná pri zmene týchto dát.

5.3.2 Backend

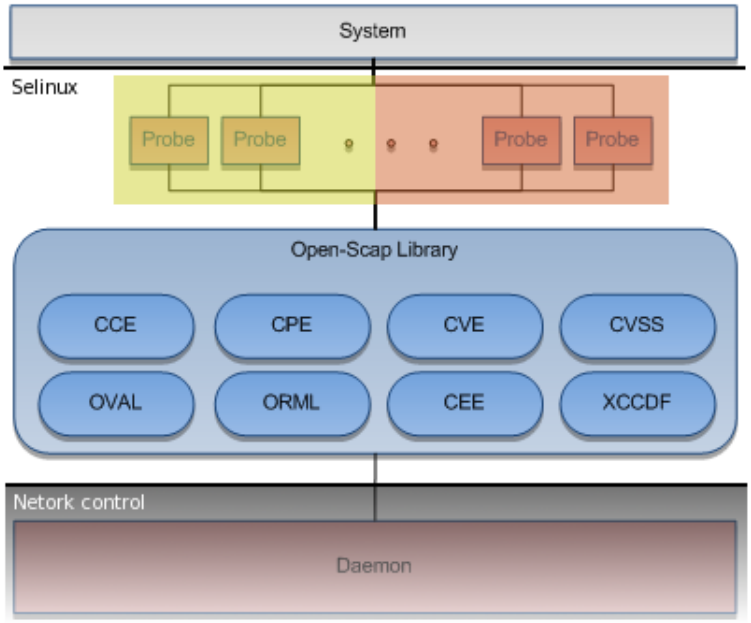
Backend je samostatný program spúšťaný na vyžiadanie spracovania úlohy a nezáleží akým spôsobom je daná úloha zadávaná. Podporným prostriedkom pre backend je taktiež už spomenuté komunikačné rozhranie, ktoré sa v tomto prípade musí starať aj o spustenie programu na vykonanie požadovaných operácií. Backend sa dá ďalej logicky rozdeliť na časť, ktorá je dedikovaná na komunikáciu backendu cez sieť a časť, ktorá volá systémové funkcie.

Bezpečnosť

Samotný backend je veľmi zraniteľná časť systému. Keď si uvedomíme, že jeden program, ktorý komunikuje cez sieť má zároveň superužívateľský prístup k celému systému a to ako práva čítania tak aj zápisu, tak sa samotný projekt na zabezpečenie systémov môže stať ich slabým miestom. Preto počas vytvárania návrhu architektúry projektu bola prijatá architektúra z obr. 5.1. Na obrázku je jasne vidieť fyzické odčlenenie medzivrstvy medzi systémom a knižnicou a komunikačné rozhranie medzi knižnicou a sieťou.

Na obrázku 5.5 sú vyznačené miesta s rozdielnou bezpečnostnou politikou. Pod šedým prechodom prekrytá časť s nápisom „Network control" sa nachádza časť backendu, ktorá je prístupná zo siete a v ktorej hrozí najväčšie bezpečnostné riziko (Daemon). Preto je sieťový daemon fyzicky oddelený od knižnice a správca ho môže kedykoľvek vypnúť. Pokiaľ získa útočník kontrolu na backendom, tak získa iba prístup k API knižnici, čo nemá pre útočníka skoro žiadnu hodnotu, keďže cez knižnicu môže iba testovať a zabezpečovať systém oproti definičným súborom už prítomným na systéme. Taktiež útočník nemôže zaviesť nižšie zabezpečenie na systém, než je nastavené pred jeho útokom. Pokiaľ chce užívateľ napríklad povoliť službu, ktorú zakázal systém Lock-Down na základe bezpečnostnej zásady, musí to urobiť ručne. Pokiaľ by chcel daný systém cez funkcie knižnice útočník napadnúť, musel by buď zaviesť do systému modifikované definičné ORML súbory a keďže je backendu zamedzené zapisovať na disk, musel by napadnúť inú službu.

Druhým bezpečnostným rizikom je prístup knižnice k celému disku bez obmedzenia. Toto riziko vzniklo kvôli zjednoteniu API knižnice pre zabezpečenie a pre skenovanie. Jedným z riešení sa ponúka použitie bezpečnostnej technológie **selinux**. Ako je vyznačené na obrázku 5.5, jednotlivé sondy, ktoré pristupujú k systému sú rozdelené na dva celky. Prvý z nich získava informácie zo systému a nepotrebuje mať povolený zápis a druhá časť, ktorá tento zápis vyžaduje. Každá zo sónd bude mať nastavenú inú selinuxovú politiku a bude



Obrázok 5.5: Náhľad na bezpečnosť backendu

môcť pristupovať iba k zdrojom, ktoré nutne potrebuje. Týmto sa samozrejme bezpečnostné riziko neodstránilo, ale dopad prípadného incidentu sa zmenšil na minimum.

API

API, inak povedané rozhranie knižnice, je jedna z najdôležitejších častí, pretože knižnica by mala podporovať čo najviac programovacích jazykov, minimálne jazyky C, C++, Java, Python a Perl.

API knižnice je tvorené funkciami na prístup k štandardom CCE, CVE, CVSS, CPE a OVAL.

Zaznamenávanie do logu

Zaznamenávanie práce backendu na klientovi je nutné kvôli spätnému overeniu, či nedošlo v systéme k neoprávnenému vyvolaniu akcií a tým k porušeniu bezpečnosti. Možností ako zabezpečiť zbieranie týchto informácií a následnú archiváciu je viacero. Prvým a najpravdepodobnejším použitým nástrojom je syslog – unixový nástroj na logovanie s možnosťou ich archivácie na vzdialenom počítači. Ďalšou dôležitou vecou, ktorú nemožno opomenúť je CEE (viď kapitolu 3.5) – štandard formátu logovacích správ, ktorý by mal backend podporovať. Pomocou tohto štandardu je možné zaznamenávať kompletnú prácu knižnice a jednotlivé udalosti sú zaznamenané v ucelenom a štandardnom formáte.

Vykonanie úloh

Ako už bolo spomenuté v predchádzjúcej kapitole, vykonanie požadovaných úloh sa deje prostredníctvom API knižnice. Na výber sú dve možnosti: prístup ku knižnici vzdialene cez protokol daemona, ktorý volá funkcie knižnice a druhým prístupom je spustenie nástroja,

ktorý importuje knižnicu a volá jej funkcie lokálne. Na diskusiu sa pridáva ešte tretia možnosť a to lokálnym prístupom cez textové užívateľské rozhranie, ktorého implementácia sa v momentálnom štádiu neuvažuje. Medzi textovým rozhraním a knižnicou by musel byť definovaný protokol na komunikáciu alebo by textové rozhranie používalo knižnicu ako spomenutý nástroj. Ďalším krokom pri vykonaní úlohy sa deje priamo v daných funkciách, ktoré okrem práce s dátami poskytujú aj prístup k operačnému systému. Tento prístup je realizovaný cez fyzicky oddelené sondy (viď kapitolu 5.3.2). Sondy zavolajú funkcie systému a vrátia výsledok do knižnice. Tá ponúka ďalšie funkcie na prácu s týmto výsledkom.

Výstup

V tomto prípade sa nám naskytlo niekoľko ciest, ako zaznamenávať výsledok vykonanej úlohy. V prípade zabezpečovacej úlohy ide len o status, ktorým úloha skončila, prípadne správy o možných problémoch. Postačujúcim mechanizmom je trojstavový informatívny typ správ v logovacom formáte, ale akonáhle bude do knižnice implementovaný štandard CEE (viď kapitolu 3.5), bude použitý ako náhrada protokolu o výsledku operácie. Každé jednotlivé pravidlo bude špecifikovať, či sa vykonalo správne, nevykonalo sa správne, prečo sa nevykonalo správne, alebo či sa vyskytli nejaké komplikácie.

V prípade diagnostickej úlohy je viac možností výstupu. Najjednoduchším prístupom je použitie ORF (OVAL Result File) súboru, ktorý dostaneme po použití OVALdi programu. Momentálne je výstupom knižnice charakteristika systému, ktorá sa ďalej použije na testovanie systému (viď kapitolu 3.1.5).

5.3.3 Komunikačné rozhranie

Komunikačné rozhranie je rozhranie slúžiace na spomínanú komunikáciu medzi frontendom a backendom, prípadne inými externými nástrojmi a je programovo nezávislé na implementácii ostatných častí. Dôležité je, aby sa toto komunikačné rozhranie dalo vymeniť za iné, v prípade, že sa zmenia podmienky, ako napríklad znefunkčnenie TCP/IP komunikácie kvôli zakázaným portom. Túto situáciu môžeme vyriešiť nahradením komunikácie tunelovaním cez SSH.

Keďže implementácia projektu bola rozdelená do dvoch hlavných častí, kde v prvej časti sa vôbec neuvažuje o komunikačnom rozhraní, nebola táto časť detailne navrhnutá, ani implementovaná (pre viac informácií viď kapitolu 6). I napriek tomu sa viedli vo fáze návrhu diskusie o možnostiach implementácie a preto uvádzam nasledujce možnosti, ktoré boli zvažované:

1. SSH + SCP[3] – cez príkazy pomocou šifrovaných príkazov a posielaním súborov cez program SCP. Jednoznačnou výhodou tohto prístupu je jednoduchosť implementácie, pretože všetky nástroje už existujú. Nevýhodou je uchovávanie hesiel pre superužívateľské účty pre každého klienta, kde musíme implementovať zabezpečenie lokálne uloženej databázy.

2. SSH + SCP + Kerberos – v tomto prístupe nám odpadá problém uchovania lokálnej databázy s heslami, ale nevýhodou je nutnosť, aby všetci klienti používali systém Kerberos.

3. Puppet – nástroj na vzdialené vykonávanie systémových úloh. Použitie je intuitívne a nie je náročné na implementáciu. V skutočnosti existujú len dve podmienky pou-

[3]Program na bezpečné prenášanie súborov cez sieť

žitia a to mať nainštalovaný puppet na každom klientovi a implementovať plugin na spúšťanie nášho programu.

4. Daemon – na klientovi je nainštalovaný špeciálny komunikačný program, ktorý načúva na porte ako server. V tomto prípade autentifikácia môže byť riešená jedným z mnohých spôsobov, ako je napríklad SSL a pod. Implementovaný autentifikačný algoritmus nie je nijak závislý na technológii tretej strany, takže z klientov odpadá nutnosť poskytovať mechanizmus SSH, Kerberos a iné. Riešenie daemonom ďalej poskytuje vlastný protokol a možnosť rozširovania so zachovaním spätnej kompatibility. Jedinou nevýhodou je náročnosť na implementáciu.

5. IPA – projekt vo vývoji firmy Red Hat sa postupne stane systémom poskytujúcim možnosť autentifikácie, komunikácie, vzdialené spúšťanie programov na klientoch ako aj doručovanie súborov cez systém LDAP[4]. Toto riešenie je momentálne vo vývoji a vyžaduje používanie tohto systému všetkými klientmi. IPA ďalej vyžaduje mechanizmy LDAP a rozšírenie aj Kerberos.

5.3.4 Manažment klientov

Ako bolo spomenuté v predchádzajúcej podkapitole, v prvej iterácii projektu sa o komunikácii cez sieť vôbec neuvažuje. V ďalšej iterácii je to nepovinná časť zastúpená jednoduchým nástrojom na pridávanie a odoberanie klientov z užívateľskej databázy. V konečnej podobe bude táto časť projektu prepracovaná tak, aby dokázala pokryť manažment počítačov nezávisle na prítomnej technológii, ale zároveň podporujúca najnovšie mechanizmy, ako je napríklad IPA alebo RHN[5].

5.3.5 Manažment konfigurácií

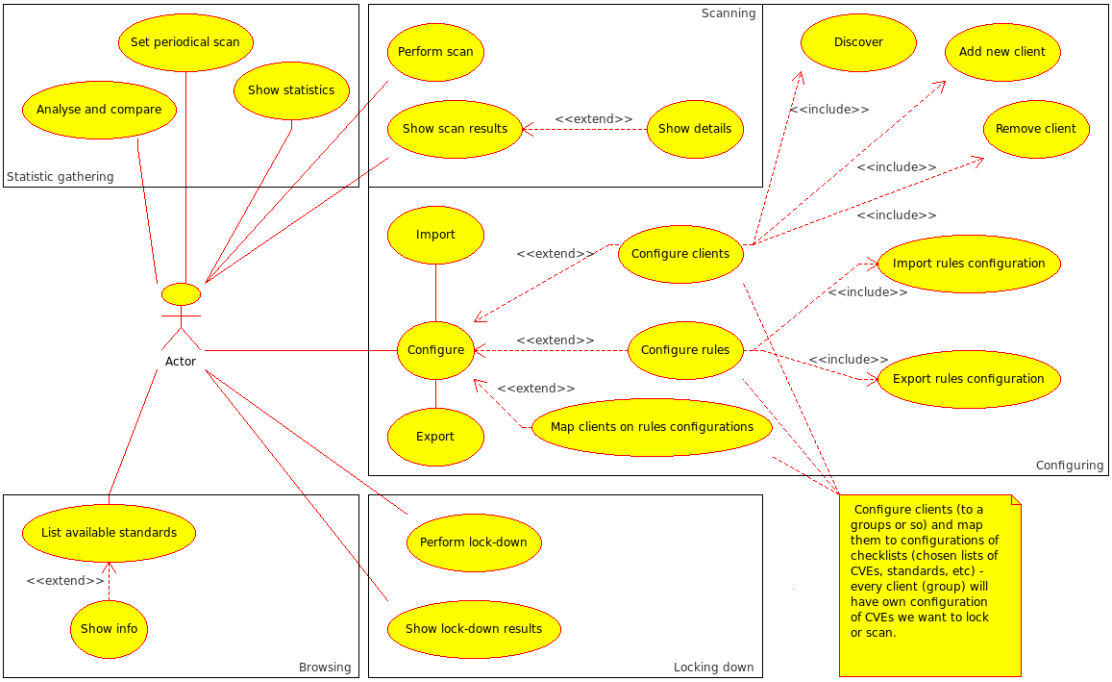
Jedna z častí frontendu, ktorá je celá implementovaná v backende. Na manažment konfigurácii je použitý štandard OVAL, ktorého API je volané metódami na správu týchto konfigurácií vo frontende. Konfigurácie sa nachádzajú v externom definičnom súbore a sú z tohto súboru získané knižnicou, ktorá ich zoskupí do jednotlivých štruktúr s ktorými sa ďalej pracuje. Ukladanie týchto definičných súborov nie je momentálne žiadané ani využiteľné.

5.4 Funkcionalita

V tejto časti by som rád načrtol základné princípy fungovania projektu ako celku. Na obr. 5.6 sú zobrazené prípady použitia pre spomínanú časť frontendu (viď kapitola 5.3.1) a na príklade si ukážeme jeden prípad použitia, ktorý bude aj slovne popísaný.

Prípady použitia môžeme rozdeliť do piatich skupín. Najdôležitejšou skupinou je „Configuring", kde užívateľ môže konfigurovať zásady štandardu a klientov, poprípade určiť, ktorý štandard sa aplikuje na daných klientov. Ďalšími dvoma skupinami sú „Scanning" a „Locking down", kde užívateľ vykonáva akcie súvisiace s diagnostikou a zabezpečením. Poslednými skupinami sú „Statistic gathering" na zber a zobrazenie štatistík a „Browsing" na zobrazenie podrobností o vybranom štandarde.

[4]Lightweight Directory Access Protocol
[5]Red Hat Network



Obrázok 5.6: Prípady použitia

5.4.1 Príklad prípadu použitia

Ako príklad na vysvetlenie princípu fungovania projektu ako celku ukážem jeden prípad užitia (viď obr. 5.7), kedy užívateľ zvolil na vybraných klientoch akciu zabezpečenia podľa vybraného štandardu.

Slovný popis

Užívateľ po zapnutí užívateľského rozhrania pred sebou vidí zoznam klientov momentálne zaradených v databáze. Po vybraní daného klienta sa zobrazí pred užívateľom dialógové okno so všetkými informáciami od klienta, aktuálnym štandardom podľa ktorého je vybraný klient zabezpečený, s históriou testov, na ktoré bol klient testovaný, ich výsledkami a možnosťami ďalšej akcie. Po tom, čo užívateľ zvolí akciu zabezpečenia na klientovi, ktorého si predtým zvolil, zobrazí sa sprievodca, kde podľa inštrukcií vyplní požadované informácie a spustí procedúru na samotnom klientovi.

Frontend v tomto okamihu začne komunikáciu s klientom cez komunikačné rozhranie. Pre jednoduchosť uvažujme, že ako komunikačné rozhranie sa použije TCP/IP daemon nainštalovaný na klientovi. Ten jednoduchým protokolom získa CCE čísla jednotlivých pravidiel daného štandardu a všetky potrebné informácie. Po overení bezpečnosti tohto príkazu daemon spustí na klientovi program, ktorý následne úlohu v už spracovanom formáte vykoná a výsledky vráti na komunikačné rozhranie. Tento proces sa ukončí spätným poslaním výsledku operácie na frontend, ktorý zabezpečí jeho zobrazenie užívateľovi. Frontend výsledky uloží do perzistentnej cache databázy, kde výsledky o danom klientovi zostanú až do chvíle, kedy sa znova nezosynchronizujú s klientom.

Ako je vidieť celý proces je veľmi náchylný na bezpečnostné hrozby. V celom procese je päť bezpečnostne problematických úsekov, ktoré je nutné pred implementáciou vyriešiť. Problématické je hlavne zabezpečenie komunikácie medzi frontendom a backendom.

Prípad použitia:	Vykonanie zabezpečovacej procedúry (Perform lock-down)
ID:	20
Stručný popis:	Systém zabezpečí vybraných klientov
Primárni aktéri:	Užívateľ
Sekundárni aktéri:	Žiadni
Predpoklady:	1. Prítomný aspoň jeden štandard 2. Vybraný aspoň jeden klient
Hlavný tok:	1. Prípad sa spustí po stlačení tlačidla na spustenie procedúry 2. Ak overenie predpokladov dopadlo dobre, pokračuj v toku 3. Vytvor nové spojenia na klientov 4. Pokiaľ sa nepodarí nadviazať spojenie na niektorého klienta, vytvor chybu a pokračuj pre ostatných klientov 5. Každému klientov vytvor novú úlohu a počkaj na jej výsledok 6. Zobraz výsledky a chyby užívateľovi
Nasledovné podmienky:	Komunikácia s klientmi bola nadviazaná a klienti vrátili výsledok procedúry
Alternatívne toky:	Zobrazenie problému užívateľovi a návrat na hlavné okno aplikácie

Obrázok 5.7: Prípad použitia

Momentálne je navrhnutá komunikácia na základe autentifikácie systémom Kerberos (viď vyššie).

5.5 Záver návrhu

V tejto kapitole bol popísaný proces návrhu systému Lock-Down a knižnice Open-Scap. Návrh sa počas obdobia začiatku práce na projekte až po súčasnosť niekoľkokrát zmenil a prispôsoboval sa hlavne rýchlemu rastu dôležitosti a veľkosti projektu. Jedným z problémov v procese návrhu boli meniace sa artefakty výstupu analýzy, ktoré sa taktiež menili s narastajúcim rozsahom projektu. Požiadavky na štandardy, ktoré sú v projekte implementované, neboli špecifikované na začiatku vývoja, ale pridávali sa postupne, čo malo tiež za následok niekoľkonásobné opakovanie procesu návrhu.

Spomenuté problémy, neustále upravovanie požiadaviek a nárast zložitosti stáli veľa práce a času a v konečnom dôsledku posunuli predpokladaný termín celkového dokončenia projektu v rámci Red Hatu z januára 2009 na december 2009. O týchto problémoch a ďalších súvislostiach budem hovoriť v nasledujúcej kapitole.

Kapitola 6

Implementácia

V tejto kapitole bude objasnený proces implementácie a špecifické detaily riešenia rôznych problémov, ktoré boli načrtnuté v predchádzajúcich fázach. Budú podané zmeny oproti pôvodnému návrhu a premietnem aktuálny stav projektu k dátumu písania tejto správy.

Implementácia projektu začala v septembri 2008 a pokračuje až doteraz[1]. Predpokladaný dátum ukončenia je stanovený na september 2009. Do implementácie projektu sa zapojili piati ľudia z firmy Red Hat a traja ľudia zo spoločnosti G2. Ich funkcia na projekte s oblasťou na ktorej sa podieľali je uvedený v podkapitole 6.1.1.

6.1 Úvod do implementácie

Implementácia projektu prebiehala v troch fázach. Prvá z nich bola v rámci analýzy projektu, kedy sa implementovali iba časti kódu poskytujúce pohľad na vizuálnu predstavu autora o projekte (išlo teda skôr o návrh ako o implementáciu, ale zdá sa mi vhodné spomenúť túto fázu). Táto predstava bola následne niekoľko krát predstavená vedeniu projektu vo firme Red Hat a upravená podľa pozmenených požiadaviek. Dôvod k tomuto postupu je vysvetlený v kapitole 4.5.

Druhou fázou projektu bola implementácia frontendu – grafického užívateľského rozhrania a treťou fázou implementácia knižnice open-scap. Obe fázy prebiehali paralelne, keďže implementácia užívateľského rozhrania musela čakať na ukončenie fáz implementácie knižnice. Tento postup nebol zvolený náhodne, ani nebol ponechaný na prirodzený vývoj projektu. Bol zvolený zámerne, pretože jednou zo základných požiadaviek na tím pracujúci na tomto projekte bola rýchlosť jeho uvedenia k zákazníkovi (tým teraz myslím zákazníka Red Hatu) a do komunity. Paralelny vývoj implementácie štandardov spolu s aplikáciou, ktorá použitie tejto knižnice uvádza do praxe bol strategickým cieľom firmy Red Hat. I keď sa môže zdať táto rýchlosť absurdná, ako je vidieť z predchádzajúcej kapitoly, kde je spomenuté predpokladané dokončenie na september 2009, dĺžka času stráveného na projekte je viac než optimistická (ako vidíme je to skoro jeden rok).

6.1.1 Projektový tím

Keďže jedným zo strategických cieľov tohto projektu sa stal čas uvedenia projektu do komunity, bolo nutné zväčšiť projektový tím. Vedenie sa rozhodlo prideliť na projekt tím skúsených programátorov, ktorí by po kvalitnej analýze a návrhu dokázali projekt implementovať v čo najkratšom čase. V tom čase začali na projekte spolupracovať ďalší štyria programátori

[1] Ako prítomnosť môžme brať do úvahy začiatok mája 2009.

z tímu **Security standards** firmy Red Hat – traja programátori v jazyku C a jeden programátor, ktorý mal lokálne nový tím riadiť. Moja úloha v projekte bola kompletná analýza a návrh projektov Lock-Down a open-scap a implementácia systému Lock-Down. Keďže sa samotná implementácia nástroja posunula, pracoval som ako pomocný programátor na knižnici open-scap.

Po určitom čase sa do projektu zapojila spoločnosť G2, ktorá sa angažuje vo vládnych projektoch Spojených štátov amerických v oblasti bezpečnosti. Tá priniesla do projektu nový pohľad na možnosti jeho rastu a ponúkla spoluprácu a ľudské zdroje na pomoc pri jeho vývoji. V projekte sa začalo uvažovať o preimplementácii projektu OVALDi do knižnice, ktorá poslúži ako základ projektu open-scap. V aktuálnom stave pracujú na projekte open-scap traja programátori z firmy G2 a ich implementačná časť je zameraná na spomínanú implementáciu OVAL štandardu do knižnice open-scap.

6.1.2 Použité technológie

Systém Lock-Down je celý implementovaný použitím programovacieho jazyka Python a grafického toolkitu GTK+ pre python (čiže knižnica pygtk). Aplikácia je otestovaná pre verzie pythonu 2.4 až 2.5. Samotnou podstatou pythonu je aplikácia prenositeľná medzi rôznymi operačnými systémami i keď táto skutočnosť nikdy nebola požadovaná. Na implementáciu nástroja Lock-Down bolo využité vývojové prostredie *Netbeans*.

Knižnica open-scap je implementovaná z väčšej časti v jazyku C. Počas návrhu bolo prijaté rozhodnutie použiť práve jazyk C, kvôli skutoťnosti, že knižnica by mala byť čo najviac prenositeľná a čo najviac využiteľná. Táto využiteľnosť sa viaže k jazykom, ktoré podporujú API knižnice. V momentálnom štádiu je to okrem implementačného jazyka C, jazyk Python a Perl. API týchto jazykov je automaticky generované cez systém SWIG [19].

Počas návrhu boli prediskutované rôzne technológie zmienené v procese analýzy, ale pristúpilo sa na rozhodnutia, že väčšinu funkcionality, ktoré nám ponúkajú navrhované technológie a programy budeme implementovať sami. Toto rozhodnutie prinesie do projektu určitú komplexnosť, ktorá nie je v linuxových systémoch až tak bežná, ale pri bezpečnostných systémoch podobnej úrovne by mala byť dodržiavaná.

Nástroj na vytváranie UML diagramov pre túto prácu bol použitý *Visual Paradigm for UML*. Na vytvárane UML diagramov vo firme Red Hat, boli použité nástroje *Gaphor* a *Umbrello*.

6.2 Proces implementácie

Implementácia bola rozdelená do dvoch hlavných častí. V prvej časti je prioritná implementácia väčšiny štandardov do knižnice open-scap a základ nástroja Lock-Down. Neuvažuje sa o sieťovej komunikácií, projekt je určený na lokálne použitie na jednom počítači. Hlavným pilierom druhej časti implementácie je sieťová komunikácia a manažment klientov. Aktuálne sa systém Lock-Down nachádza v prvej fáze implementácie. Čo presne sa podarilo implementovať nájdete v nasledujúcej kapitole.

6.3 Aktuálny stav projektu

Projekt sa momentálne nachádza približne v polovici navrhovanej funkcionality a taktiež v polovici predpokladaného projektového plánu. V knižnici open-scap boli úspešne implementované a otestované štandardy CCE, CPE, CVE, CVSS a z časti OVAL boli implementované funkcie na získavanie štruktúr definícií. Aktuálne sa pracuje na dokončení funkcionality

štandardu OVAL a implementácia štandardu XCCDF. Taktiež sa uvažuje o naimplementovaní API knižnice pre jazyky C++ a Java.

V projekte Lock-Down mohli byť implementované iba funkcie, ktoré v danom momente ponúka knižnica a okolitá funkcionalita, ktorá bola dopredu schválená manažmentom. V aktuálnom stave nástroj Lock-Down ponúka zobrazenie definícií konfigurácií v prehľadnom formáte a testuje lokálny systém na vybranú konfiguráciu. Na toto testovanie žiaľ nemohla byť použitá knižnica open-scap, keďže táto funkcionalita nie je dokončená. Namiesto knižnice pracuje nástroj so systémom OVALDI, ktorý bol pôvodne na tento účel navrhnutý. Táto zmena v implementácii je dočasným riešením a vznikla iba kvôli predstaveniu funkčnosti a možností nástroja v rámci tejto práce. Tak isto zobrazovanie definícií je momentálne implementované z definičného súboru OVALu, neskôr bude na tento účel použitý XCCDF súbor. V ďalšom kroku, po dokončení implementácie funkcionality časti OVAL v knižnici open-scap, bude OVALDI z projektu vylúčený a nástroj bude na testovanie využívať funkcie knižnice a po dokončení implementácie parsovania XCCDF súborov bude nástroj Lock-Down zobrazovať konfigurácie pomocou týchto dokumentov.

V aplikácii bola ďalej implementovaná celá vnútorná funkcionalita ako jadro aplikácie a zobrazovanie informácií. Momentálne sa čaká na doimplementovanie už spomínaného získavania charakteristík z lokálneho počítača (časť OVAL).

Na knižnici open-scap sa aktuálne pracuje na získavaní charakteristík počítača, ktoré momentálne implementujú vývojari spoločnosti G2.

6.3.1 Zmeny oproti návrhu

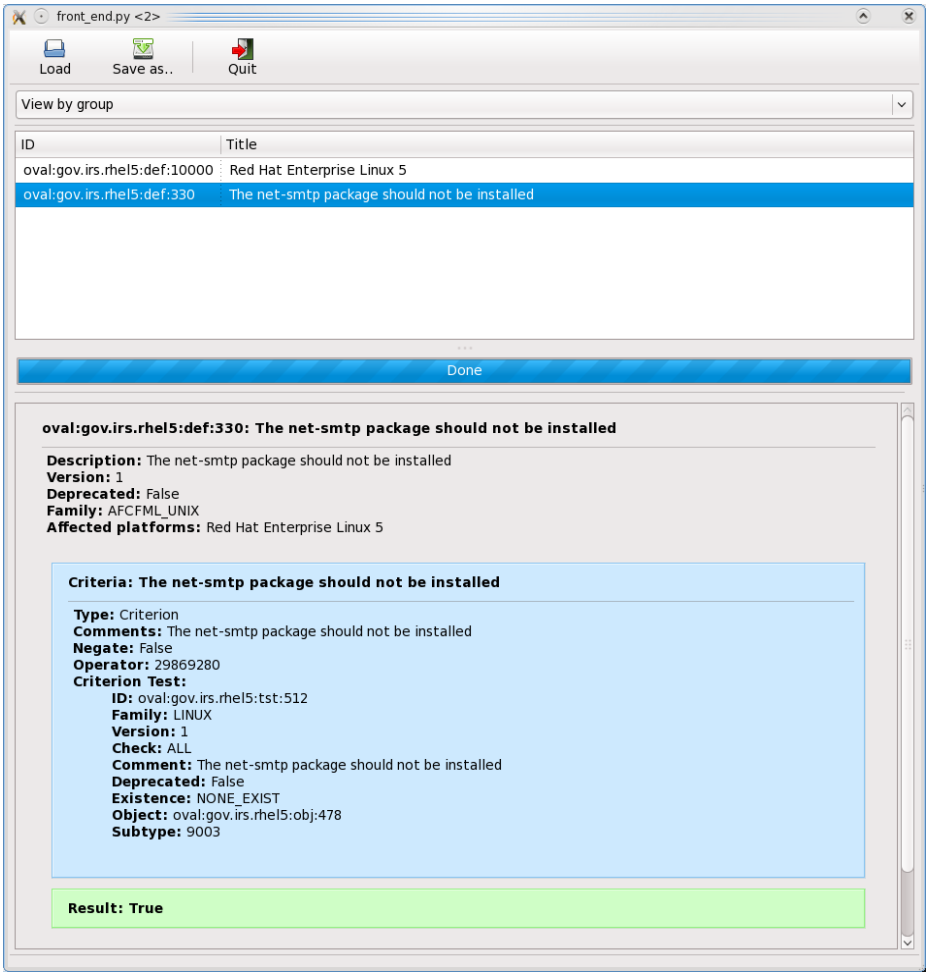
Keďže museli nastať zmeny v implementácii systému Lock-Down, kvôli chýbajúcej funkčnosti v knižnici open-scap, bol som nútený upraviť návrh grafického okna. Problém bol v nadpisoch a identifikátoroch jednotlivých záznamov. Keďže boli príliš dlhé na to, aby panel so zoznamom a panel s detailami daného záznamu mohli byť umiestnené horizontálne vedľa seba (viď obr. 5.3).

6.4 Problémy pri implementácii

Pri implementácii sa vyskytlo niekoľko problémov, ktoré stáli za presúvaním termínu dokončenia niektorej funkcionality. Tým najzávažnejším problémom bol návrh API knižnice pre každý štandard SCAP, kde sa nedorozumením začalo pracovať na dvoch nezávislých návrhoch a bez ďalšej diskusie na ich implementovaní. Za týmito problémami stála hlavne odlišnosť v spôsobe komunikácie medzi tímom v Red Hate a tímom v G2. Často zlyhávala mailová komunikácia a problematická bola synchronizácia vzájomných stretnutí. Po určitom čase sa tieto problémy odstránili stanovením ľudí na oboch stranách, ktorí zodpovedali za určitú časť projektu a komunikácia bola obmedzená iba na dvoch ľudí. Ďalej bol založený projektový mailing list a projektový IRC kanál na serveri #freenode, čo zlepšilo komunikáciu a odstránilo riziko podobných zlyhaní.

6.5 Zdrojové kódy a programová dokumentácia

Zdrojové kódy k programu Lock-Down sa nachádzajú na priloženom dátovom nosiči. Návod na spustenie nástroja sa nachádza v prilohe B. Dokumentáciu k projektu Lock-Down je možné vygenerovať programom epydoc priamo zo zdrojových súborov.



Obrázok 6.1: Akutálna grafická podoba okna so zabezpečovacími politikami

Zdrojové súbory knižnice open-scap sa tiež nachádzajú na priloženom dátovom nosiči, v prípade záujmu sa na stránke `http://www.open-scap.org` nachádza aktuálna verzia. Programová dokumentácia pre knižnicu open-scap je distribuovaná spolu so zdrojovými kódmi a nachádza sa v podadresári *docs/* v zdrojovom adresári knižnice. Žial dokumentácia nie je momentálne dostupná pre štandard OVAL a na jej vytvorení sa bude pracovať.

6.6 Záver implementácie

Proces implementácie začal len nedávno a preto veľká časť funkcionality knižnice a teda aj projektu Lock-Down je v štádiu návrhu. Projekt Lock-Down je príliš závislý na knižnici open-scap a preto veľká časť implementácie bude prebiehať až po dokončení knižnice. Keďže nie som v projekte v postavení, kedy by som mohol ovplyvniť chod a plán projektu, nemohol som ani zmeniť termíny alebo zmeniť návrh projektu do takej miery, aby som do termínu odovzdania tejto práce stihol splniť požadovanú funkcionalitu. S týmto rizikom som nepočítal, keďže pri zadávaní práce som mal na projekte pracovať sám a projekt bol iba veľmi malá časť z toho, na čom sa momentálne pracuje.

I napriek týmto problémom je prototyp projektu v spustiteľnej verzii (nachádza sa na priloženom dátovom nosiče). V prílohe B sa nachádza popis ako naimplementovaný nástroj spustiť.

Jedným bodom zo zadania práce bolo naimplementovať systém Lock-Down s použitím nástroja OVALDI a Augeas. Nástroj Augeas bol kvôli spomenutým dôvodom (viď 3.4) z projektu Lock-Down vyradený. Nástroj OVALDI tiež nebude následne používaný, ale pre názornosť funkčnosti systému je priložený prototyp Lock-Downu naimplementovaný s použitím tohto nástroja.

Kapitola 7

Záver

Na projekte Lock-Down sa začalo pracovať v júli 2008 a počas šiestich mesiacov sa z malého projektu vyvinul projekt, od ktorého sa očakáva veľký prínos v oblasti zabezpečenia linuxových systémov. Na konci septembra roku 2008 sa uskutočnila konferencia „4th Annual IT Security Automation Conference“, kde bol predstavený nový smer bezpečnostnej politiky, ktorým sa moderný svet hodlá uberať. Myšlienky tejto konferencie sa stali základom pre vznik komunitného projektu open-scap, za ktorého vznikom stoja inštitúcie a firmy Red Hat, G2, MITRE a NIST. Počas ďalších štyroch mesiacov sa pracovalo na návrhu, upravovala sa analýza projektu a v januári 2009 sa naplno začalo s fázou implementácie. V momentálnom štádiu je implementovaná veľká časť funkcionality knižnice open-scap a prezeranie získaných definícií v projekte Lock-Down.

Projekt má priniesť podporu pre komunitu zaoberajúcu sa implementovaním a nasadením štandardov SCAP hlavne v oblasti vývoja knižníc poskytujúcich podporu pre implementáciu nových nástrojov podporujúcich štandardy SCAP. Projekt Lock-Down bol následne rozšírený o diagnostický nástroj a postavený do čela záujmu tímu „Security standards“ firmy Red Hat.

V práci som priblížil svet bezpečnosti operačných systémov Linux a načrtol základné pojmy z oblasti počítačovej bezpečnosti. Oboznámil som sa so základnými problémami, cieľmi a riešeniami pri zabezpečení systémov pomocou implementovania princípov štandardov SCAP a objasnil teoretické základy a motivácie pre vznik nového projektu na zabezpečovanie operačných systémov.

Druhá časť práce bola zameraná na analýzu, návrh a implementáciu aplikácie určenej na diagnostiku a zabezpečenie linuxových systémov. Pri návrhu boli použité netradičné postupy, ktoré pramenia zo snahy manažéra projektu pristupovať modernými návrhovými postupmi a následným experimentovaním podľa princípov logiky návrhu konkrétneho projektu. Pri tomto projekte sme najprv navrhli GUI a následne pokračovali v klasickom návrhovom postupe.

Ďalším postupom v práci bude pokračovanie v iteráciach implementácie projektu. Prvou časťou bola implementácia knižnice a časti frontendu. Prvý krok projektu bude zavŕšený implementáciou štandardu OVAL. Ďalšími krokmi bude doimplementácia užívateľského rozhrania, implementácie alternatívnych ciest v komunikačnom rozhraní, manažmente klientov a implementovaní ďalších častí vyžiadaných manažmentom.

Na projekte ďalej spolupracujú Peter Vrabec – popredný bezpečnostný inžinier firmy Red Hat, Steve Grubb – vedúci tímu Security standards, konzultanti zo spoločností G2, NIST a MITRE a softwaroví inžinieri spomenutého tímu – Dan Kopeček, Tomáš Heinrich a internista Lukáš Kuklínek.

7.1 Zhodnotenie projektu

Projekt bol zadaný firmou Red Hat ako nástroj na zlepšenie zabezpečenia operačných systémov Linux. Problematický zvrat nastal, keď bolo zadanie projektu firmou Red Hat zmenené a projekt nabral na vážnosti a samozrejme na veľkosti. Z pôvodného nástroja, ktorý sa mal stať sprievodcom bezpečnostných štandardov, ktorý zabezpečuje systém pomocou štandardizovaných postupov s využitím existujúcich nástrojov sa skoro všetká funkcionalita týchto nástrojov musela preimplementovať v projekte a ten sa stal pre jedného človeka v rozumnom čase nezvládnuteľný. Red Hat preto vyčlenil celý tím vývojárov, ktorí sa mali podieľať na jeho vývoji a projekt sa rozdelil na dva menšie projekty, Lock-Down a open-scap. Ďalší obrat nastal, keď sa o projekt začala zaujímať americká vláda a do projektu vstúpila organizácia G2. Projekt sa stal pre firmu Red Hat jedným zo strategických projektov, o ktorý sa začali zaujímať ďalší zákazníci a možnosť ovplyvnenia smeru, ktorým sa projekt ďalej vyvíjal sa z mojej strany stala nepatrná. Projekt je open-source a teda prístupný (po dokončení) celej verejnosti. Projekt prinesie komunite zlepšenie bezpečnosti systémov a možnosť vývoja ďalších nástrojov.

Projekt bol vyvíjaný modelom vodopád, ktorý nie je ideálny a jeho slabé stránky sme sa snažili odstrániť malými iteráciami v analýze, návrhu aj implementácii. Každý koniec iterácie bol kontrolovaný manažmentom a bolo tak odstránených veľa chýb. V ďalšom projekte by som po podobných skúsenostiach rád použil iný, možno agilný spôsob manažmentu projektu.

7.1.1 Vlastný prínos

Môj prínos do projektu bol všestranný. Zo začiatku som v projekte zastupoval úlohy analytika a návrhara. Keď sa projekt dostal do štádia ukončenia analýzy, bol návrh projektu zosúladený s predstavami všetkých zainteresovaných (záujmových) skupín a dokončil sa návrh projektu. V štádiu implementácie zastávam úlohu analytika v smere analýzy a návrhu prípadných zmien a úlohu implementátora celého projektu Lock-Down a podielam sa taktiež na implementácii knižnice open-scap – konkrétne na implementácii API pre python a perl a návrhára súčastí, ktoré sa budú v budúcnosti implementovať.

Od júla 2009 budem zastávať vo firme Red Hat v tíme Security Standards pozíciu experta na bezpečnosť spojenú so štandardami SCAP a budem sa podieľať na ďalšom vývoji nových projektov týkajúcich sa open-scap alebo Lock-Down.

Celá analýza, ktorá je v práci uvedená spolu s návrhom projektu Lock-Down je môj výtvor. Pri návrhu bolo API knižnice spolu s vnútorným zložením navrhnuté v kooperácii s členmi tímu Security standards vo firme Red Hat a s programátormi z G2. Je len veľmi ťažko odhadnuteľné aký pomer mojej práce hral v konečnom návrhu knižnice rolu. Implementačne som jediným programátorom systému Lock-Down a z knižnice open-scap som programoval API pre jazyky python a perl a momentálne pracujem na implementácii CEE štandardu.

7.2 Budúcnosť projektu

Budúcnosť projektu sa zatiaľ javí ako pozitívna. Veľa zákazníkov prejavilo o produkt záujem a veľa externých ľudí sa o projekt zaujíma pasívne (štatistiky podľa nových zapísaných na projektovom „mailing-liste").

7.2.1 Plánované rozšírenia

Ako vyplýva z textu tejto práce, projekt open-scap sa stal prvou komplexnou implementáciou štandardov SCAP v oblasti open-source. Ďalšie využitie projektu je skoro neobmedzené. Aktuálne v spolupráci s Fakultou Informačných Technológii VUT v Brne vzniká nový projekt zadaný ako diplomová práca [18]. Jej účel je rozšíriť záber projektu Lock-Down o nový spôsob merania zraniteľnosti systémov pomocou knižnice open-scap a databázy NVD[1].

Ďalším možným rozšírením projektu je podpora testovania sily hesiel v systéme a kontrola integrity kľúčových bezpečnostných nástrojov.

[1] National Vulnerability Database

Literatúra

[1] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997

[2] Quinn, S.D., Ziring, N.: Specification for the Extensible Configuration Checklist Description Format (XCCDF) Verison 1.1.4, NIST a NSA, 2008.

[3] Mann, D.: An Introduction to the Common Configuration Enumeration. Dostupný na URL: URL `http://cce.mitre.org/about/documents.html`, 2008.

[4] Harris, C.: Schema documentation - CWE Version 1.0. Dostupný na URL: `http://cwe.mitre.org/documents/schema/index.html`, 2008.

[5] Open Vulnerability and Assessment Language. Dostupný na URL: `http://oval.mitre.org/`, 2008.

[6] Information Security Automation Program Overview. Dostupný na URL: `http://nvd.nist.gov/scap.cfm`, 2008.

[7] Augeas - a configuration API. Dostupný na URL: `http://augeas.net/`, 2008

[8] OVAL definitions for Red Hat Enterprise Linux 3,4,5. Dostupný na URL: `https://www.redhat.com/security/data/oval/`

[9] A Complete Guide to the Common Vulnerability Scoring System Version 2, Jún 2007. Dostupný na URL: `http://www.first.org/cvss/cvss-guide.pdf`

[10] MITRE Corporation, Common Event Expression, The CEE Board, Jún 2008

[11] B. Toxen: Bezpečnost v Linuxu, ISBN 80-7226-716-7, 2003

[12] David Mann, An Introduction to the Common Configuration Enumeration White Paper Version 1.7, 2008

[13] Maroš Barabas, Nástroj pro snazší zabezpečení počítačů s OS Linux, Semestrální projekt, 2009

[14] David Orenstein, QuickStudy: Application Programming Interface (API), Computerworld, 2000

[15] MDA Guide Version 1.0.1. Dostupný na URL: `http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf`

[16] Jim Arlow, Ila Neustadt, UML2 a unifikovaný procec vývoje aplikací, ISBN 978-80-251-1503-9, 2007

[17] Tarek Ziadé, Expert Python Programming, ISBN 978-1-847194-94-7, 2008

[18] Pratyusa Manadhata, Jeannette M. Wing, Measuring a System's Attack Surface, CMU-CS-04-102, Január 2004

[19] SWIG Users Manual, Jún 1997, Dostupný na adrese `http://www.swig.org/Doc1.1/PDF/SWIGManual.pdf`

[20] Petr Hanáček, Jan Staudek, Bezpečnost informačních systémů, 2000

Dodatok A

Architektúra

Architektúra projektu sa skladá z troch hlavných častí:

1. FrontEnd: skladá sa z užívateľského rozhrania, ktoré ponúka abstrakciu pre narábanie s klientmi, konfiguráciu a samotné spustenie akcií na vzdialených klientoch.

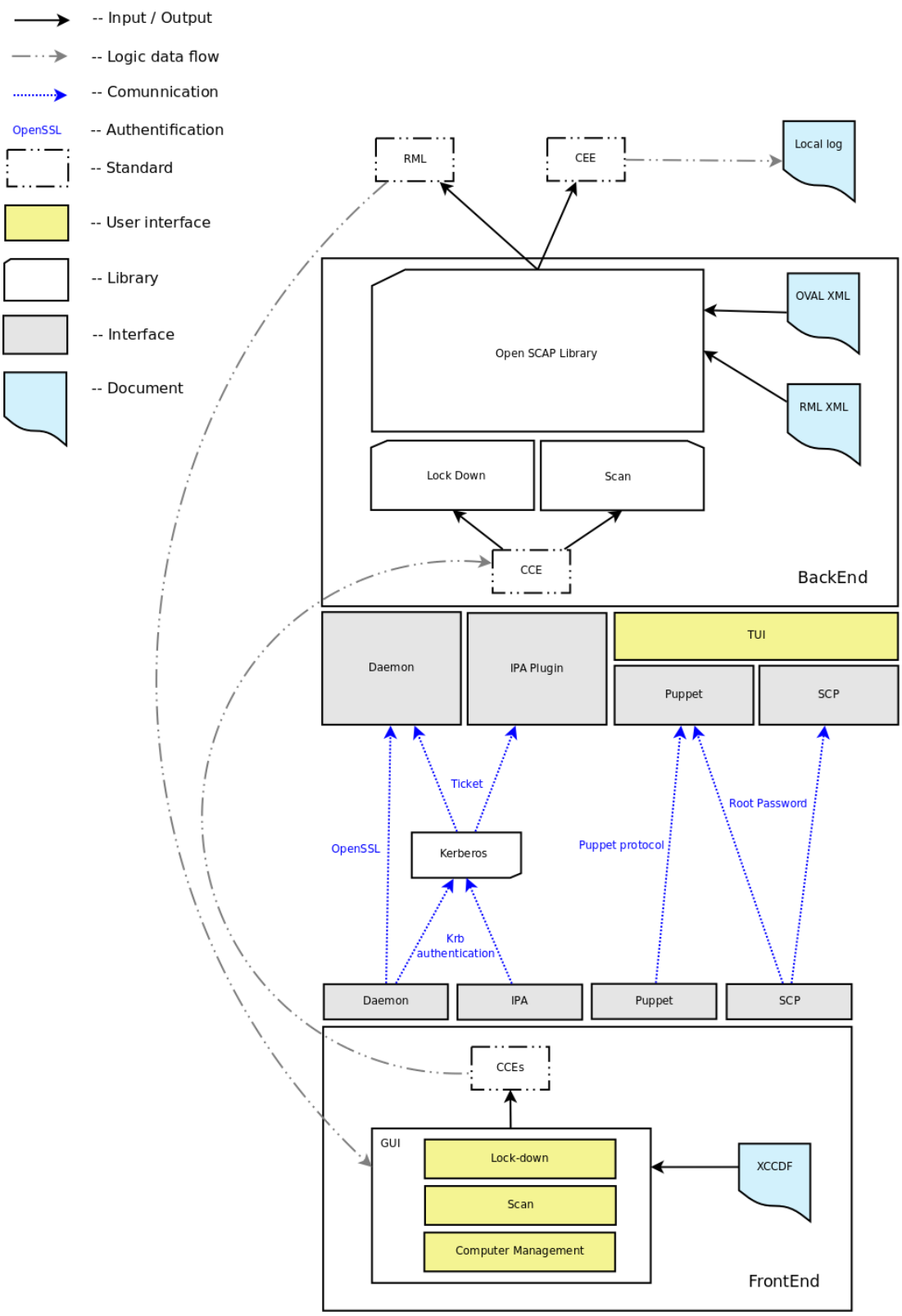
2. Komunikačné rozhranie: ma za úlohu poskytnúť funkcie na komunikáciu s backendom. Táto vrstva sa delí na dve oddelené časti, kde jedna je určená pre frontend a druhá pre backend. Možnosti implementácie tejto vrstvy sú podrobne rozpísané v kapitole 5.3.3.

3. BackEnd: program obsahujúci knižnicu na narábanie so štandardmi SCAP a výstupný modul, ktorý obsahuje zaznamenávanie do logu a formátovací nástroj na upravenie výstupu.

Ako je vidieť na obr. A.1 šedou prerušovanou šípkou, medzi rozhraním frontendu a rozhraním backendu sa prenášajú informácie ako zoznam CCE (CCEs), ktorý slúži na „oznámenie" backendu, ktoré zásady štandardu je nutné diagnostikovať / zabezpečiť. Popis akcie, ktorá sa bude vykonávať, jej parametre a všetky informácie, ktoré je nutné prenášať medzi týmito časťami sú implementované protokolom komunikačného rozhrania a v obrázku naznačené modrými šípkami. Modrý popis pri šípkach označuje metódu autentifikácie.

Ďalej je v obrázku A.1 popísaný tok dát zo súborov s popismi štandardov (modré polia v čiernom ráme) do jednotlivých častí aplikácie a naznačené užívateľské rozhrania (žlté polia). Biele polia s prerušovanou čiernou čiarou značia jednotlivé SCAP štandardy, ktoré aplikácia využíva. Štandardy ako napríklad CVE alebo OVAL nie sú v aplikácii označené, pretože sú obsiahnuté v súbore označenom skratkou štandardu XCCDF.

Biele polia s plnou čiernou čiarou označujú knižnice a šedé polia rozhrania.



Obrázok A.1: Architektúra projektu

Dodatok B

Návod na spustenie nástroja Lock-Down

Spustenie samotného nástroja Lock-Down nie je nijak zložité. Problematické je splnenie všetkých predpokladov k jeho spusteniu.

Prvým predpokladom, je nainštalovaný linuxový systém. Nezáleží na distribúcii, ale systém musí mať nainštalované všetky potrebné knižnice, ktoré vyžaduje knižnica openscap. Nižšie uvádzam zoznam požadovaných knižníc, balíkov a programov. V zátvorkách je uvedená verzia, na ktorej bol nástroj testovaný:

- Program SWIG (1.3.29)
- Python (2.5.2)
- Knižnica libxml2 (2.7.3)
- Balík libxml2-devel (2.7.3)
- Balík pcre-devel (7.8)
- Glibc (2.9.3)
- Ovaldi (5.5-4)

Po nainšťalovaní všetkých potrebných súborov, stiahnite z priloženého DVD alebo z adresy `www.openscap.org` knižnicu open-scap.

Po rozbalení archívu postupujte podľa nasledujúcich príkazov:

```
$ cd openscap
$ ./autogen.sh
$ ./configure
$ make
$ make install DESTDIR=/tmp/scap
```

Nástroj Lock-Down na priloženom DVD je nastavený tak, aby knižnicu hľadal práve v adresári */tmp/scap/local/lib/python2.5/site-packages/* alebo */tmp/scap/local/lib64/python2.5/site-packages/*.

Po stiahnutí archívu s nástrojom Lock-Down ostáva posledný krok a to nakopírovanie všetkých *xml* a *xsd* súborov a adresára *xml* z adresára, kde bol ovaldi nainštalovaný do adresára so súbormi nástroja Lock-Down.

Po vykonaní všetkých predchádzajúcich krokov, spustíte program príkazom:

```
$ python front_end.py
```

Na obrazovke sa objaví základné okno, kde sa bude v budúcnosti pracovať s nakonfigurovanými klientami. Do okna s konfiguráciami sa dostanete kliknutím na tlačidlo **Policies.**

Po otvorení dialogového okna musíte importovať definičný súbor, ktorý je priložený na DVD. Jeho názov je *scap-rhel5-oval.xml*. Tento súbor samozrejme bude fungovať pri testovaní iba na systéme Red Hat Enterprise Linux 5. Preto som na DVD pridal súbor s pozmenenými pravidlami, ktoré slúžia ako ukážka funkcionality nástroja. Tento súbor má názov *scap-modified-oval.xml.*

Na testovanie určitej konfigurácie sa v okne nachádza odpovedajúce tlačítko.

Výsledky testovania systému sú uložene v adresári so zdrojovými súbormi programu Lock-Down ako html súbory (prezerateľné cez index.html) a ako xml súbor results.xml.