**Jihočeská univerzita v Českých Budějovicích**
**Přírodovědecká fakulta**


# Útočné stromy

**Bakalářská práce**

**Pavel Mai**

Školitel: Ing. Petr Břehovský

České Budějovice 2011

**Bibliografické údaje**
Mai. P, 2011: Útočné stromy.
[Attack trees. Bc.. Thesis, in Czech.] – 37 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

# Anotace

Útočné stromy poskytují formální, metodický způsob, jak popisovat bezpečnost systémů založenou na různých způsobech útoků. Strom reprezentuje útoky na systém pomocí stromové struktury s cílem úspěšného útoku, který je vyjádřen vrcholem stromu.

Potřebujeme přehledně znázornit hrozby, které ohrožují počítačové systémy. V případě, že budeme schopni analyzovat všechny způsoby, kterými může být systém napaden, můžeme navrhnout protiopatření k zamezení těchto útoků. Pokud dokážeme pochopit, kdo jsou útočníci, jejich schopnosti, motivaci a cíle, budeme moci zavést vhodná opatření k tomu, abychom zabránili skutečným hrozbám.

# Abstrakt

Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a system in a tree structure, with the goal as the root node [1].

We need a model of threats against computer systems. If we can understand all the different ways in which a system can be attacked, we can likely design countermeasures to thwart those attacks. And if we can understand who the attackers are - not to mention their abilities, motivations, and goals - maybe we can install the proper countermeasures to deal with the real threats [1].

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 12. 12. 2011

Pavel Mai

# Poděkování

Rád bych poděkoval školiteli **Ing. Petrovi Břehovskému** za odbornou asistenci a konzultace. **RNDr. Jaroslavu Ichovi** za odbornou pomoc. Rodině za veškerou podporu.

# Obsah

# 1 Úvod

Důkladně popsat a následně plně zabezpečit rizika určitého systému je velmi složitou a namáhavou záležitostí. Většinou se jedná o dlouhodobou akci, která vyžaduje odbornou firmu s přítomností několika odborníků a specialistů v dané problematice. Pokud mluvíme o bezpečnosti systému, může se jednat o informační systém, ale není vyloučeno, že by mohlo jít i o fyzické zabezpečení určitého objektu např. banky, obchodu,… Ve skutečnosti se jedná o podobný princip popisu rizik, jejich hodnocení a následné řešení. Pro tyto případy byl v terminologii bezpečnosti informačních systémů zaveden termín „Útočný strom“, který by měl i běžnému uživateli zprostředkovat ucelený a jednodušší náhled do problematiky zabezpečování a bezpečnosti. První uvedl tento termín Bruce Schneier v roce 1999 [1]. Hlavním důvodem bylo spíše zjednodušit a vytvořit náhled na bezpečnost systémů s cílem pomoci v zabezpečování a správnému postupu zabezpečování systémů před útočníky. Je žádoucí zabezpečovat systém od dostupnějších a méně sofistikovanějších prvků bezpečnosti k nákladnějším a více komplexnějším. Tímto postupem předcházíme nežádoucímu průniku do systému. Pokud budeme systém správně zabezpečovat, dokážeme tím lépe předejít průniku od potenciálního útočníka. Předpokladem je, že útočník musí mít pro daný druh útoku určité schopnosti, vědomosti, vybavení a prostředky. V případě, že budeme správně postupovat, riziko průniku se sníží.

## 1.1 Motivace

V České republice je tento termín „Útočného stromu“ značně nezdokumentovaná problematika, a proto jsem se rozhodl touto problematikou zabývat. Rozšířit si vlastní obzor, prozkoumat dosud neprozkoumané a přispět touto dokumentací v českém jazyce. Z dlouhodobého vyhledávání informací jsem dospěl k názoru, že na českém trhu a v česky psaných publikacích bylo o tomto tématu napsáno velmi málo. Proto bych chtěl svou prací přispět k rozšíření povědomí o tomto způsobu popisu bezpečnosti systémů. Jedná se o velmi zajímavé formulování problematiky.

V zahraničí je tato problematika rozsáhleji popsána a je jí věnováno podstatně více času. Dokazuje to i fakt, že několik zahraničních firem, převážně amerických, se pokouší psát a vyvíjet programy pro kreslení a modelování útočných stromů. Příkladem těchto nástrojů jsou vyspělejší komerční verze programů SecurITree od kanadské společnosti Ameneza [2], AttackTree+ od americké společnosti Isograph, které mají mnoho funkcionalit pro kreslení útočného stromu a vyhodnocování rizik informačních systémů. Jejich velkou nevýhodou je pouze komerční použití za poplatek. Je možné vyzkoušet si demo verzi, která má limitované funkcionality, nebo je časově omezená.

## 1.2 Očekávané výsledky

Jak je již známo, ve světě je k dispozici několik komerčních verzí programu pro kreslení útočných stromů. V tomto portfoliu se informace o komerčním software uchovávají velmi důkladně a nejsou volně k dispozici širší veřejnosti. Vše je orientováno na spotřebitele s cílem finančního zisku.

Tato problematika je na univerzitní půdě minimálně popsána. Dalo by se říci, že v česky psaných dokumentacích a knihách není téměř žádná možnost setkat se s česky psaným materiálem. Proto jsem se rozhodl přispět informacemi k této problematice v českém znění.

Dále jsem chtěl přispět do světa volně šiřitelného software, svou prací na aplikační části práce. Vše co jsem vytvořil, je možné dále obohacovat o funkcionality a znalosti. V rámci univerzitního vývoje je veškerý materiál volně k dispozici.

## 1.3 Cíle práce

Cílem práce je vytvořit dokumentaci k chápání útočného stromu jako takového. Vytvoření aplikace, která nám bude pomáhat vyhodnocovat bezpečnostní rizika systému a usnadňovat orientaci ve struktuře zabezpečení.

V případě dlouhodobého vývoje aplikace by se mohlo dospět k velmi zajímavé a užitečné pomůcce, která by mohla usnadnit formu zabezpečování a chápání bezpečnosti systémů.

# 2 Analýza rizik

Z článku „Co je to riziko a analýza rizik“, jsem použil přehledný souhrn informací o analýze rizik. Informace jsou součástí webu BusinessInfo.cz:

Prvním krokem procesu snižování rizik je přirozeně jejich analýza. Analýza rizik je obvykle chápána jako proces definování hrozeb, pravděpodobnosti jejich uskutečnění a dopadu na aktiva, tedy stanovení rizik a jejich závažnosti.

Analýza rizik zpravidla zahrnuje:

1. **identifikaci aktiv** – vymezení posuzovaného subjektu a popis aktiv, které vlastní
2. **stanovení hodnoty aktiv** – určení hodnoty aktiv a jejich význam pro subjekt, ohodnocení možného dopadu jejich ztráty, změny či poškození na existenci či chování subjektu
3. **identifikaci hrozeb a slabin** – určení druhů událostí a akcí, které mohou ovlivnit negativně hodnotu aktiv, určení slabých míst subjektu, které mohou umožnit působení hrozeb
4. **stanovení závažnosti hrozeb a míry zranitelnosti** – určení pravděpodobnosti výskytu hrozby a míry zranitelnosti subjektu vůči dané hrozbě

Kvalitní řešení jakéhokoliv problému v jakékoliv oblasti je vždy postaveno na kvalitní analýze rizik, která je základním vstupem pro řízení rizik.

Hodnocení rizik představuje neustálé zvažování:

a) poškození aktivit, která mohou být způsobena naplněním hrozeb, přičemž je nutno vzít v úvahu veškeré potenciální důsledky
b) reálné pravděpodobnosti výskytu takových rizik z pohledu převažujících hrozeb, zranitelnosti a aktuálně implementovaných opatření

Výsledky hodnocení rizik pomohou určit odpovídající kroky vedení organizace, priority pro zvládání rizik a realizaci opatření určených k zamezení jejich výskytu. Je možné, že proces hodnocení rizik a stanovení opatření bude třeba opakovat několikrát, aby byly pokryty různé části subjektu (organizace) nebo jednotlivé činnosti.

V každém případě je nutné si již na počátku stanovit úroveň, na jakou chceme analyzovaná rizika eliminovat. Snaha o odstranění všech rizik by samozřejmě vedla k neúměrným nákladům při realizaci příslušných opatření, a v každém případě by se zákonitě podepsala i na funkčnosti daného subjektu.

Z tohoto důvodu v rámci analýzy rizik posoudíme otázky zbytkových rizik, které se snažíme vymezit na základě jejich posouzení ve vztahu k hrozbám, úrovni zranitelností a navrhovaných protiopatření. Na základě toho pak vybíráme konkrétní přístup a metodu analýzy rizik [7][16].

## 2.1 Data útočného stromu

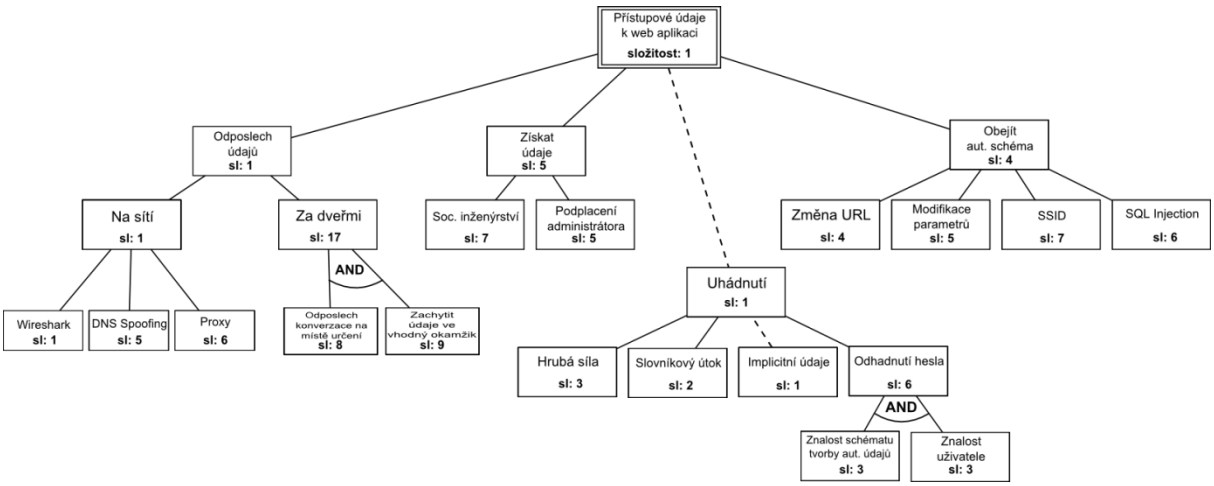
Analýza rizik je pro útočný strom zdrojem vstupních dat. Těmito daty je struktura stromu naplněna a využívá je k následné analýze hrozeb bezpečnostního systému. Zpracování těchto dat je u útočného stromu specifické v tom, že se struktura stromu prochází od listů ke kořenu

stromu, který představuje úspěšný průnik do systému. Pokud získáme dostatečné množství dat, které představují popis hrozby, riziko napadení se snižuje na velmi malou hranici. Jsme schopni tyto hrozby eliminovat.

## 2.2 Historie útočného stromu

V souvislosti s analýzou rizik a jejich hodnocení, se vůbec poprvé o útočném stromu zmínil Bruce Schneier v publikaci Dr. Dobb's Journal v prosinci 1999 [1].

Pojem „Útočný strom“ je odvozen od struktury zápisu do stromové struktury. Takovou stromovou strukturu může reprezentovat například implementace do XML souboru, grafické znázornění nebo textová forma zápisu. Ve stromu se jednotlivé útoky do systému označují uzlem nebo listem stromu. Cílem každého útoku je průnik do systému, který je v grafu stromu reprezentován vrcholem stromu tzv. cíl (GOAL). K vrcholu je možné se dostat několika cestami, které jsou ve skutečnosti postup určitého útoku na systém. Pokud máme vše potřebné k tomu, abychom se dostali od listu na vrchol stromu, znamená to, že došlo k infiltraci do systému [1].




**Obrázek 2.1: Grafická podoba útočného stromu**

Na obrázku 2.1 je vidět, jak může takový strom vypadat a co může obsahovat za rozhodující parametry. Název útoku je důležitý pro člověka, naproti tomu parametry jsou důležité pro rozhodující logiku automatu, který nám bude říkat, kde začít zabezpečovat, kolik nás bude zabezpečení dané rizikové cesty stát a případně časovou náročnost úkonu. Parametry mohou být např. výhodnost útoku, složitost, vyžadované finance, potřebné speciální vybavení nebo nutnost určité útoky kombinovat, za účelem dosažení nadřazeného uzlu.

Komplexnost takového stromu záleží na tvůrci stromu. Jaké jsou možnosti útoků na cíl stromu. V jednoduchých systémech je snadné zachovat přehlednost a orientaci v dané stromové struktuře. S přibývajícími údaji se orientace a přehled snižuje. Pro ulehčení orientace nám může takový útočný strom posloužit a pomoci nám vyhodnotit rizika systému.

Tyto stromy mohou být chápany oboustranně. Ze strany útočníka mohou sloužit k vedení útoku, naproti tomu u bezpečnostního technika mohou být využity k efektivnímu zabezpečování systému. Na zahraničním trhu jsou k dispozici komerční verze softwaru, který umí vytvořit útočný strom z aktuálního zabezpečení informačního systému, vyhodnocovat bezpečnost systému a dokonce testovat bezpečnost systému. Testování je většinou prováděno s použitím dalšího testovacího softwaru. Toto jsou projekty velice sofistikované a komplexní,

na kterých se podílejí mnozí odborníci. Automatizovaný postup aplikace musí obsahovat spoustu bezpečnostních pravidel. Také implementace aplikace je velice náročná.

## 2.3 Metody analýzy rizik

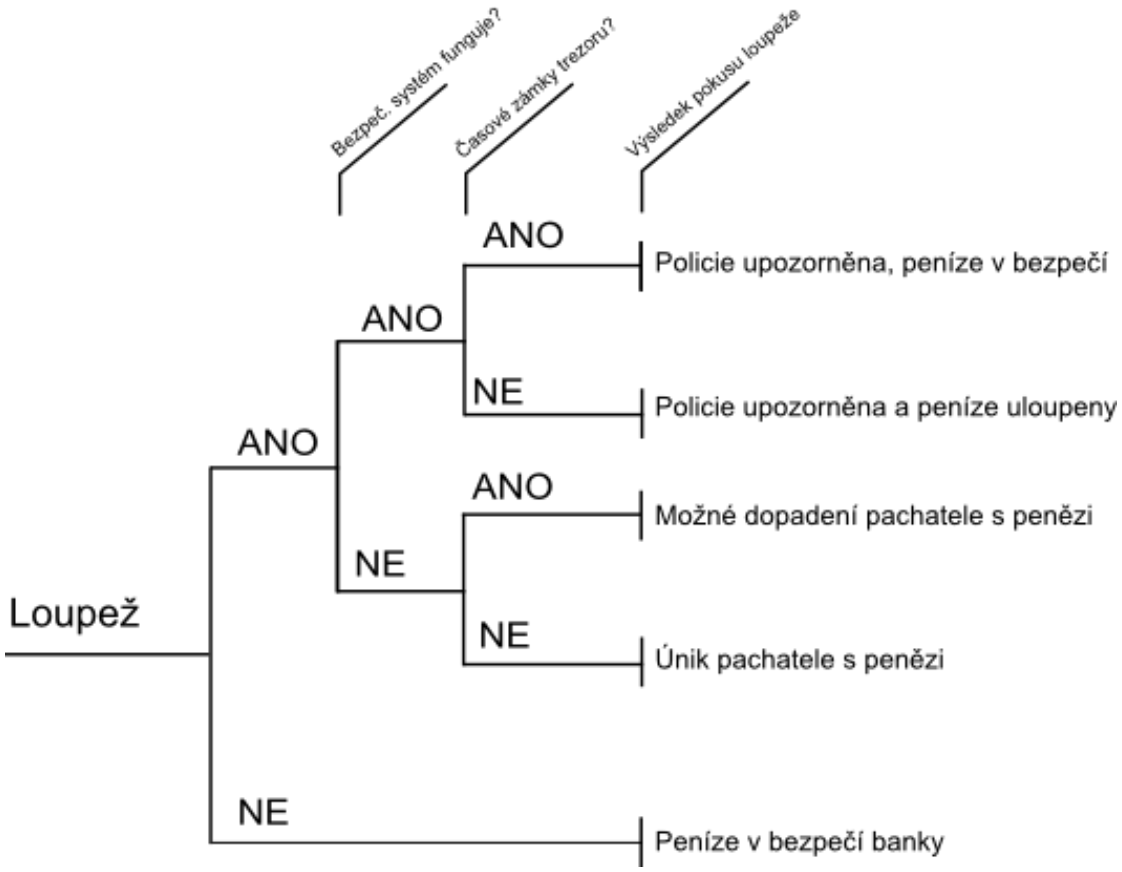
Při realizaci analýzy rizik je možné použít několik postupů. Je nezbytné si rozmyslet, jakých výsledků bychom chtěli dosáhnout, a jaké jsou priority při rozhodování. Záleží na postupu vyhodnocování a formě výstupních dat. Můžeme dosáhnout několika stavů v případě ETA, nebo pouze jednoho výsledného stavu při FTA. Detailněji bude vše popsáno v dalších kapitolách. Tyto techniky jsou součástí buď vpřed orientované analýzy, nebo vzad orientované analýzy. Tyto metody se používají běžně při zpracování rizik. Jejich kombinováním je možné vytvářet komplexní stromy, které nám říkají nejen, co se stane, když něco v systému selže, ale také co by se mohlo stát [2][3].

### 2.3.1 Vpřed orientovaná analýza

Tato forma zpracování rizika je běžně známá. Postupuje se podle struktury stromu od hlavní události k listům, které jsou ve skutečnosti výsledek události. Tuto metodu reprezentuje tzv. strom událostí. Viz obrázek 2.2. Startovní pozice pro vpřed orientovanou analýzu představuje základní narušení systému. Zjišťujeme, zda se integrita a bezpečnost systému narušila ve vztahu k uživateli, prostředí nebo systému zabezpečení. Sledujeme, k jakému narušení systému tyto události vedou, ať už se riziko snižuje nebo zvyšuje. Můžeme se snažit tyto posloupnosti událostí eliminovat. Pro takovéto analýzy se používá např. struktura stromu událostí (event tree) [3].

### 2.3.2 ETA (event tree analysis)

Na následujícím stromu událostí, který je k vidění na obrázku 2.2, si můžeme představit, jak taková analýza funguje.




**Obrázek 2.2: Příklad stromu událostí (event tree)**

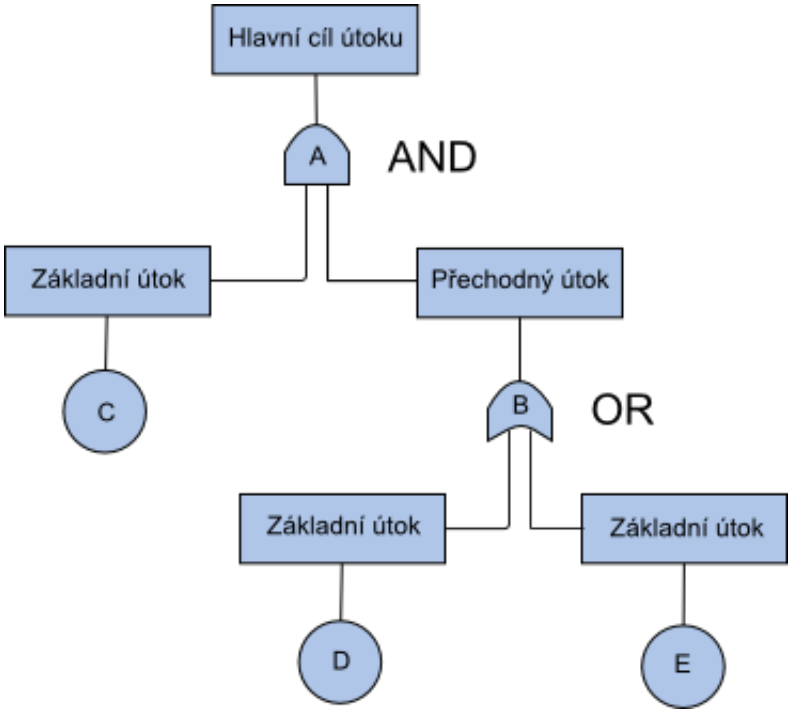
Na úplném začátku stromu máme hlavní událost. Tato událost může představovat narušení systému. V našem případě se jedná o názorný strom, který se snaží jednoduše popsat výsledek a možný důsledek loupeže. Ve skutečnosti se jedná o binární strom, který obsahuje dvě následující větve a rozhodování se řídí binární logikou. Pokud je podstata události naplněna, vydáme se cestou, která je označena slovem „ANO“. V opačném případě se vydáme pokračující cestou „NE“. Každý uzel obsahuje pouze tyto dvě možnosti. Tímto způsobem se pokračuje ve stromu událostí. V každém kroku se musí brát ohled na událost, která může nastat. Ve výsledné fázi se dobereme výsledků, které mohou nastat. Výsledkem je například dopad na systém zabezpečení. Příkladem může být také výpadek elektrické energie a její dopad na systém.

### 2.3.3 Zpět orientovaná analýza

U tohoto typu analýzy je typické, že vychází z externě zjištěných napadení systému a zkoumá příčiny takového napadení. Tyto příčiny napadení se snažíme následně eliminovat nebo minimalizovat. Analýza vytváří přehled potenciálně slabých míst systému. Se znalostí takového slabého místa můžeme podniknout jisté protiopatření a riziko eliminovat nebo úplně vyřadit. Typickým použitím tohoto typu analýzy je FTA. V oblasti bezpečnosti je to právě útočný strom, který tuto metodu využívá [3].

### 2.3.4 FTA (fault tree analysis)

Cílem této analýzy je určení možných kombinací základních útoků, které mohou vést k dosažení hlavního cíle útoku.




**Obrázek 2.3: Příklad stromu selhání (fault tree)**

FT se rozrůstají od kořene stromu k listům a vytvářejí obrácenou korunu stromu. Tento strom je tvořen kombinací uzlů, bran a událostí. Na obrázku 2.3 je možné vidět základní strom, který obsahuje propojení uzlů pomocí bran AND, OR. Při rozhodování se využívá boolovské logiky a díky ní se vytvářejí kombinace útoků.

Strom selhání je uspořádán do úrovní, které jsou propojeny logickými operátory za účelem kombinace základních útoků. Pokud je v úrovni obsažena brána typu AND, je nutné, pro postup do vyšší instance, splnit všechny dílčí útoky tohoto uzlu. V případě, že je pouze jeden z nich nesplněn, je nemožné uskutečnit útok. U uzlu typu OR stačí naplnit podstatu pouze jednoho z útoků a přechodný útok je možné uskutečnit. Tímto způsobem vyhodnocování stromu dosahujeme potřebných hodnot pro rozhodování. Pro nás je vypovídající pouze hodnota základního útoku (C, D, E), kterou nemusíme již dále specifikovat. Z těchto hodnot vytváříme postupné útoky, které jsou na cestě k hlavnímu cíli útoku. FTA je analýza, která vychází z předchozího zjištění nějakého slabého místa. Vytváří se strom z těchto hrozeb a následně se systém testuje na zranitelnost. Zabezpečení systému se uskutečňuje postupně. Při postupu zabezpečení uvažujeme o všech aspektech rizikovosti a následně se snažíme tyto rizika odstraňovat. Postupujeme tak dlouho, až je celý systém zabezpečen a hrozby odstraněny. V průběhu tohoto zabezpečování je někdy nutné hrozby přidávat a upravovat. Útočný strom vychází právě z FTA.

Jaký je rozdíl v metodách ETA a FTA? FTA poskytuje grafické znázornění kombinací jistých událostí, které vedou k selhání nebo narušení systému. Hlavním úkolem je správné definování hlavního cíle útoku. FTA pomáhá vytvořit postup, který bude sloužit ke snížení rizika hrozby. Vytvořením protiopatření se tyto hrozby snažíme minimalizovat. ETA nám poskytuje grafickou podobu série událostí, způsobené jednou událostí, která vede k několika možným výsledkům. Tyto dvě metody dohromady nám mohou přinášet přehled bezpečnostních rizik. Přínosem kombinování je analýza problému a komponenty, která

způsobuje tento problém. Dále identifikace potenciálního problému a série událostí, které tento problém způsobují. Můžeme si uvědomit, nejen, co se stane, pokud v systému něco selže, ale zároveň, jaké to může mít potenciální důsledky. Jak již bylo dříve řečeno [20].

### 2.3.5 Analýza útočného stromu založena na FTA

Při analýze útočného stromu dochází k průchodu stromu a vyhodnocování rizik systému. Jak už bylo zmíněno, celý strom obsahuje uzly a logické brány. Zjišťujeme, jaké uzly kombinovat za účelem dosažení cíle (vrchol stromu). Cestu od listu k vrcholu vidíme na obrázku 3.1. Ta v reálu představuje použití implicitního hesla a průnik do systému. V případě, že této informace využijeme efektivně, budeme schopni implicitní údaje v počátku implementace systému změnit, a tím zabránit elementárnímu útoku. V dalších fázích analýzy narazíme na komplexnější útoky, které jsou i pro útočníka složitější a tudíž je budeme odstraňovat později. Je dokonce možné, že realizujeme protiopatření k jinému typu útoku, a tím ovlivníme i ostatní typy útoků. Můžeme jejich složitost snížit nebo zvýšit. Pokud například zavedeme šifrování dat, můžeme některé provedení útoku ztížit natolik, že nebude možného ho uskutečnit. Právě tímto se stává útočný strom útočným stromem. V průběhu eliminace hrozeb se útočný strom neustále upravuje, a tedy i zpřesňuje. Pokud se objeví slabé místo systému, my ho přirozeně zavedeme. Důsledkem toho všeho je třeba tento strom neustále aktualizovat a vyhodnocovat logické brány. Jedná se o neustálý proces vyhodnocování rizik.

Za tímto účelem je vhodné disponovat komponentou, která bude strom vyhodnocovat. V další časti práce, bude tato komponenta představena a vysvětlena. Umí nejen kompletně vyhodnotit a vypsat hierarchickou strukturu, ale také upozornit na nejslabší místa systému zabezpečení [2][3].

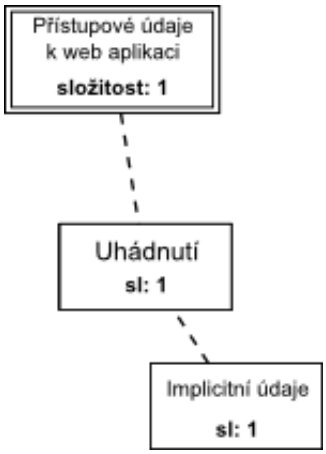
# 3 Popis datové reprezentace stromu

Pod pojmem útočného stromu si můžeme představit mnohé. Útočné stromy poskytují formální, metodický způsob, jak popisovat bezpečnost systémů založenou na různých způsobech útoků. Útočný strom je využíván pro účely zabezpečování a eliminace průniku do systému zabezpečení. Pokud dokonale zanalyzujeme útočníka, jeho motivace a cíle útoku, tak se jeho úspěšnost útoku snižuje. Jak z názvu vyplývá, útočný strom reprezentuje pohled útočníka a jeho možné kroky k průniku do systému. Pokud se nám podaří tímto stromem útočníka dokonale popsat, tak je velmi malá pravděpodobnost, že se mu podaří do systému proniknout [1].

V odvětví analýzy rizik a bezpečnosti se jedná o formální náhled na možná bezpečnostní rizika systému. Vytváříme si ucelený přehled o možném průniku do systému, vyhodnocujeme možnosti úniku dat a můžeme strom upravovat v závislosti na aktuálním dění. Tím může být myšleno, zjištění nových hrozeb systémů nebo vytvořené protiopatření hrozby našeho systému. Útočný strom se neustále upravuje za účelem maximálního bezpečí.

Grafické znázornění takového stromu můžeme vidět na obrázku 2.1 a představit si, jak se taková struktura vytváří. Grafická reprezentace útočného stromu není jediná varianta, jak strom pro uživatele reprezentovat. Na obrázku 5.2 je k dispozici textový formát útočného stromu.

Vrchol stromu představuje úspěšné získání přístupových údajů webovské aplikace. K naplnění této hrozby musíme podniknout několik kroků, které představují dílčí kroky celého útoku. Viz. obrázek 3.1.




**Obrázek 3.1: Vrchol stromu**

Pro nejjednodušší způsob získání těchto údajů je možné zkusit zadat implicitní údaje. Ve struktuře stromu 2.1 je vyznačeno čerchovanou čarou.

Tyto údaje jsou běžně určeny výrobcem zařízení, administrátorem, správcem dané služby a v nejhorším případě nejsou požadovány vůbec. Někdy je také možné setkat se s prvotně nastavenými údaji od poskytovatele. Tyto hodnoty jsou např. hesla nastavena na „1111“ pro všechny uživatele aplikace. V takovém případě se může stát, že útočník získá přihlašovací údaje klientů a umožní si přístup do aplikace pod identitou několika klientů.

Pro nejefektivnější zabezpečení systému se používá právě útočný strom, který nám takové slabé místo odhalí. V podstatě nám řekne, kde se nachází největší riziko a na nás, administrátorovi, bezpečnostním technikovi je tuto hrozbu minimalizovat. Vytvoříme nutné protiopatření, nebo tuto hrozbu zcela eliminujeme, a můžeme pokračovat v dalším

zabezpečování. Tímto způsobem pokračujeme do doby, dokud nejsou všechny hrozby odstraněny.

Abychom mohli tato data uchovávat a následně zpracovávat, je nutné používat vhodnou datovou strukturu, která nám bude tento strom představovat. Strom může ve výsledku všech auditů, analýz a postupů obsahovat velké množství dat, které je nutné uchovat, a co možná nejrychleji a nejefektivněji zpracovat. Pro takové účely může být použita syntaxe jazyka XML nebo jiná podobná syntaxe určena pro tvorbu struktury stromu.

Jak je vidět na obrázku 2.1, data útočného stromu, které budeme chtít uchovat, jsou hierarchicky uspořádána. Proto se syntaxe jazyka XML jeví jako vhodná volba. Struktura XML dokumentu tvoří také hierarchickou strukturu. V určitých případech se zápis XML dokumentu tváří jako stromová struktura.

Pro účely vytváření specifických stromů je vhodné využít syntaxi jazyka XML, která se jmenuje GraphML. Je to speciální nástroj k vytváření dokumentů XML, které reprezentují grafy, stromy,… viz kapitola 4.2.1

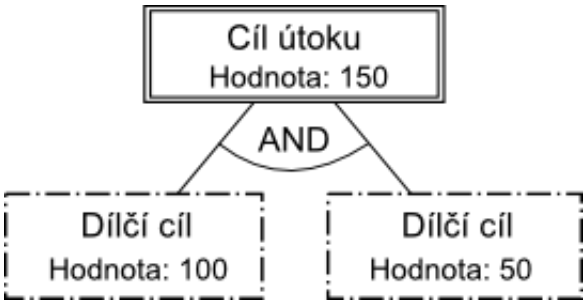
# 3.1 AND/OR uzly

V rámci útočného stromu a jeho vyhodnocení, je nezbytné zavést jisté metody rozhodování. Stejně, jako by se musel rozhodovat člověk, tak se bude muset rozhodovat i programová aplikace.

Ve struktuře stromu se objevují dva typy uzlů. AND a OR. Jedná se o dva klasické logické operátory, které se využívají k rozhodování a kombinování útoků, za účelem dosažení vyššího cíle. Tato funkcionalita je pro fungování a také modelování útočného stromu velmi důležitou součástí. Pokud bychom ji neměli k dispozici, jen těžko bychom popsali reálnou situaci některé z hrozeb.

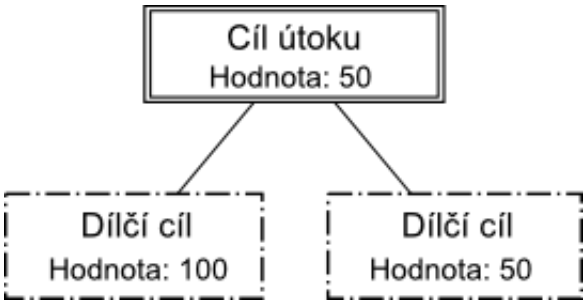
Uzel typu AND obsahuje dílčí cíle, které jsou reprezentovány konkrétním útokem. V uzlu, který je rodičem těchto dílčích cílů, je obsažena informace o typu uzlu. V případě uzlu AND je do rodiče vložena suma hodnot těchto dvou dílčích cílů útoku viz obrázek 3.2. Tyto dílčí cíle mohou být ve struktuře stromu reprezentovány dalším uzlem typu AND nebo OR, pod kterým se ukrývá další podstrom.

V opačném případě zde může být pouze konečný uzel, který se v terminologii stromů nazývá listem. List neboli konečný uzel se v anglické terminologii nazývá LEAF. Hodnota LEAF se v aplikační části práce využívá k identifikaci listu. Na obrázku 5.2 je vidět textová forma výpisu, kde se tato hodnota objevuje. Na obrázcích 3.2 a 3.3 je pro přehlednost rodič označen dvojitou čarou a list čerchovanou čarou.




**Obrázek 3.2: AND uzel**

Pokud při průchodu stromu narazíme na uzel typu OR, tak vybíráme do tohoto uzlu minimální hodnotu složitosti útoku z jeho potomků. Viz obrázek 3.3.




**Obrázek 3.3: OR uzel**

Speciálním případem uzlu typu OR je vrchol stromu, který se označuje jako GOAL. Toto označení je použito proto, aby se rozlišil obyčejný uzel OR od vrcholu stromu. Vrchol stromu je ve skutečnosti také uzel typu OR, jen s tím rozdílem, že uchovává minimální hodnotu, která představuje největší riziko pro systém.

Útočný strom je ve skutečnosti složen z množiny uzlů AND a OR. Propojením těchto uzlů se vytváří výsledný útočný strom, který představuje reálné hrozby systému. Postupem času se tyto hrozby mohou měnit v závislosti na postupu zabezpečování. Strom se může ve výsledku větvit takovým způsobem, že jeho přehledné znázornění a vizualizace nebude vůbec možná. Pokud bychom chtěli zobrazit nějaký disproporční strom, narazili bychom na problém jeho korektního zobrazení. Proto jako efektivnější varianta může být textová forma výpisu, která s přibývajícími daty roste pouze jedním směrem. Data představují další řádek, viz obrázek 5.2. Ve vizualizaci se jedná o další prostor na obrazovce. Proto u rozsáhlých a navíc disproporčních stromů může být velký problém, tato data přehledně zobrazovat.

Existují knihovny (spring.xsl, graphml2svg.xsl), které jsou k dispozici v rámci projektu GraphML [6]. Tyto knihovny umějí pomocí XSL transformace strom modifikovat o vektorové informace. Tyto informace se mohou použít k následné vizualizaci jako SVG soubor. Nevýhodou těchto knihoven je, že jsou použitelné pouze pro velmi malé grafy.

# 3.2 Parametr uzlu

Jak už bylo na obrázcích 3.2 a 3.3 naznačeno, uzly stromu musí obsahovat hodnoty, podle kterých se bude algoritmus prohledávání a vyhodnocování stromu rozhodovat.

Pro tyto účely je nutné zvolit konkrétní parametr uzlu, podle kterého se budeme ve stromu pohybovat. Je velmi obtížné zvolit parametr nebo parametry, které budou reprezentovat věci z reálného světa. Ve výsledku není zcela možné tuto realitu napodobit. Můžeme se ji pouze pokusit přiblížit. Do jisté míry se jedná o experimentování a analýzy, jak zvolit vhodný parametr pro útočný strom.

Na počátku bylo jednoduché zvolit běžné věci z reálného světa, které by mohli představovat parametry uzlu. Z názvu útočný strom je evidentní, že se jedná o pohled ze strany útočníka. Parametry představují ve skutečnosti nároky na útočníka, které by měl splnit, aby se mu podařilo uskutečnit útok na systém.

Předpokládané parametry:

- cena
- čas
- vybavení
- vědomosti

Ve finálním řešení není jednoduché všechny tyto parametry zahrnout do implementace. Narážíme na mnohdy neřešitelné duplicity nebo navzájem se ovlivňující se parametry jednotlivých útoků. Stejně tak není zcela možné přesně definovat čas potřebný pro provedení útoku.

Pokud řekneme, že na útok potřebujeme 10 minut, tak tato hodnota není vždy stejná. Museli bychom analyzovat situaci, ve které se útočník nachází, abychom mohli přesně tento čas definovat.

Stejná situace nastává při parametru cena útoku. Pokud bychom chtěli přesně definovat, kolik financí musí útočník vynaložit, aby se útok vydařil. Například při odposlechu na síti je potřeba počítač, který by byl obsažen u všech dílčích útoků a jeden z nich by se vybral. V tomto případě je vše v pořádku. Ale pokud by se v další cestě vybral například uzel uhádnutí heslo pomocí hrubé síly, tak k tomuto útoku je také potřeba počítač. Za předpokladu, že se budeme muset kombinovat tyto dva typy útoku, dochází k duplicitě a dvojnásobné hodnotě ceny za počítač. Je to jen exemplární příklad toho, jaký problém by se musel v aplikaci ošetřovat. Nebylo by jednoduché tyto duplicity hlídat a správně je interpretovat.

Pokud bych chtěl dosahovat lepšího obrazu reálného světa, bylo by vhodné vymyslet jistý koeficient, který by se vypočítal ze vstupních parametrů. Pro další zefektivnění rozhodování je možné vymyslet například koeficient „výhodnost“ pro útočníka, který bude reprezentovat zainteresovanost útočníka. Vstupními daty budou cena, čas, vědomosti,… Výsledkem bude hodnota, která bude vypovídat o celkové obtížnosti útoku. Pokud bychom chtěli ještě více experimentovat, mohli bychom koeficienty kombinovat. Další složka rozhodování by mohl být parametr přínos útoku. Museli bychom si nejspíše vytvořit nějakou skupinu tabulek, podle kterých bychom se řídili. Toto může být také námět pro další rozšiřování tohoto projektu. Jistě bychom něco takového mohli najít u komerčních společností, které se na tuto problematiku specializují. Toto se však považuje za velké tajemství.

### 3.2.1 Složitost útoku

V mém případě jsem zvolil, po několikátém rozhodování, jednotný parametr pro všechny uzly. Zvolil jsem složitost útoku. Tento parametr nabývá v listech stromu hodnot 1-10 a v uzlech AND je možné dosáhnout i vyšších hodnot. Tento parametr byl zvolen s ohledem na všechny parametry, které jsou dříve vyjmenovány. Při volbě hodnot pro jednotlivé útoky se vycházelo ze zkušeností školitele a také jeho předmětů Bezpečnost informačních systému a Počítačová kriminalita (Kybernalita). V hodnotě jednotlivých složitostí jsem se snažil zahrnout všechny aspekty útoku tak, aby se co možná nejvíce blížily realitě.

# 4 Přehled použitých technologií

Při realizaci útočného stromu bylo nutné použít několik komponent, které spolu komunikují a jsou kompatibilní. Ve výsledné aplikaci spolu tyto komponenty vytváří funkční celek. Zjišťování kompatibility a možnosti nasazení v mé aplikaci bylo jednou z nejtěžších záležitostí. Nikde není přímo napsáno, jaké vlastnosti a chování je prioritní pro konkrétní využití. Na tyto poznatky musí tvůrce přijít postupem času sám. Pokud vychází ze zkušeností svých, nebo tvůrců, kteří mají lepší zkušenosti, může si značně ulehčit práci. Po konzultacích, pročítáním dokumentace a vyhledáváním informací jsem dospěl k těmto komponentám.

## 4.1 NetBeans IDE 7.0

Pro realizaci celé aplikace bylo použito vývojové prostředí NetBeans IDE verze 7.0. Toto vývojové prostředí je velmi dobře zdokumentováno. Je k dispozici rozsáhlý tutoriál. Navíc jsem toto vývojové prostředí využíval v rámci zadaných semestrálních projektů. Mám s ním velmi dobré zkušenosti.

Další výhodou je možnost přidávání pluginů, pomocí kterých je možná přidávat další funkcionality. Pro mne vhodná funkcionalita je přeložení projektu do jar archívu, který lze spustit z příkazové řádky. Debugger je velmi dobře zpracovaný, a při vývoji byl několikrát využit. Velmi užitečnou funkcí je lehké přidávání knihoven do projektů. V rámci implementační části jsem musel přidat knihovnu, která není implementována v balíku Java knihoven. Mnohé další základní funkce jsem využil při vývoji.

## 4.2 Java

Tento objektově orientovaný jazyk jsem použil z důvodu širokého rozšíření a podpory. Z tohoto důvodu je na tuto platformu vyvíjeno mnoho knihoven a aplikací. V dnešní době jde o velmi populární programovací jazyk, proto i knihovna XOM, kterou jsem využíval v aplikaci, je napsána pro tento jazyk. Abych uvedl na pravou míru, pro tento jazyk je napsáno mnoho knihoven pro zpracování XML dokumentů. Jedná se o různě zaměřené knihovny. Například DOM je knihovna pro zpracování dokumentu uloženého v paměti v podobě stromu. Pro průchod takového stromu se využívá rekurze. Naproti tomu SAX knihovna je zaměřena na událostmi ovlivněné zpracování, kdy doplňujeme těla tříd, za účelem specifického zpracování dokumentu. Dalšími knihovnami jsou například dom4j, jdom, StaX,… Podle dalšího vývoje je evidentní, že se čím dál tím více lidí snaží napsat knihovnu lepší než předchozí tvůrci. Někdy bohužel není možné k dané knihovně najít potřebnou dokumentaci. Po předchozím prostudování dokumentací jsem zvolit XOM (XML object model), který se jeví jako velmi efektivní nástroj, s velmi pěknou dokumentací a funkcionalitou [15].

## 4.3 XOM (XML object model)

Jedná se o programové vybavení, které je napsáno pro jazyk Java. Využívá se ke zpracování XML dokumentů. Toto vybavení je dílem pouze jednoho člověka. Elliotte Rusty Harold se při vývoji řídil třemi pravidly. Správnost, jednoduchost a výkonnost. V tomto pořadí. Výsledkem jeho práce se stala knihovna velmi jednoduchá na pochopení, ale zároveň velmi efektivní a rychlá. Při zpracování velkých objemů dat je u knihovny zaručena velká rychlost

zpracování. Za předpokladu, že má uživatel znalosti XML, měl by být schopen rychle porozumět syntaxi knihovny. Není zde vůbec nutné studovat knihovny do hloubky. Vše je velmi zřetelné a jasné. Při práci s touto knihovnou jsem narazil jen na několik problémů, které nebyly v dokumentaci zcela popsány. Jednalo se o zprovoznění XPath dotazů na XML dokument a Namespace deklareace [17].

Jinak považuji tuto knihovnu za velmi zdařilou a efektivní. Pro programátora představuje usnadnění v podobě zkrácení a zpřehlednění zdrojového kódu. Zápis příkazů je v této knihovně krátký a názorný [17].

V začátcích tvorby parseru, jsem zkoušel aplikaci vytvářet ve spojení s knihovnou DOM. Tato knihovna nebyla využita při realizaci hned z několika důvodů. Hlavním důvodem byla její složitá a zdlouhavá forma zápisu příkazů do zdrojového kódu. Už při jednoduché operaci, musí vývojář použít mnoho příkazů, aby dosáhl požadovaného výsledku. Knihovna využívá mnoho operační paměti. U většího souboru je paměťová náročnost značná, proto může dlouho trvat jeho zpracování. Zpracování dokumentu probíhá rekurzivně, a to není vždy nutné.

Snad jedinou nevýhodou knihovny XOM je, že není součástí balíku Java Util. V takovém případě musíme knihovnu přidávat do vývojového prostředí. K tomuto nám poslouží jar archív, který je k dispozici na stránkách věnovaných XOM knihovně[17].

## 4.4 XML

Extensible Markup Language (zkráceně XML, česky rozšiřitelný značkovací jazyk) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Je zjednodušenou podobou staršího jazyka SGML. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat [8].

Vyhovujícím faktem je nezávislost na platformě a programovacím jazyce. Jeho procházení, zpracovávání a modifikace je možné provádět ve všech možných programovacích jazycích, pro které byla patřičná knihovna napsána. V současné době je k dispozici několik knihoven, které umožňují procházení XML dokumentů [8][17].

## 4.5 XPath

XPath (XML Path Language) je jazyk, pomocí kterého lze adresovat části XML dokumentu. Pomocí tohoto jazyka lze z XML dokumentu vybírat jednotlivé elementy a pracovat s jejich hodnotami a atributy. XPath se používá v mnoha aplikacích XML, mezi nejvýznamnější patří využití v XSLT. Jazyk XPath je standardem vydaným organizací W3C [19].

Využití tohoto jazyka, který je velmi efektivní, bylo pro mne velkým ulehčením práce programátora. Tato syntaxe dokáže v nemalé míře ušetřit spousty řádků zdrojového kódu. Co je nutné složitě a dlouze zapsat v rámci programové knihovny, tak pomocí výrazu XPath je možné zapsat v jednom řádku.

Při výběru knihovny pro zpracování XML je nutné, aby použitá knihovna podporovala tento jazyk. Při mém rozhodování a zkoušení knihoven jsem narazil na nekompatibilitu s tímto jazykem. Současná nejnovější verze 2.0 je lepší variantou toho jazyka. Bohužel knihovny většinou podporují pouze starší verzi 1.0, která je složitější na zápis výrazů a deklaraci jmenných prostorů (Namespace). Není to však velká komplikace, pouze vás může jiná forma zápisu překvapit.

## 4.6 Oxygen XML editor

Oxygen XML editor je užitečný nástroj pro tvorbu XML dokumentů. Předností je v tomto nástroji celá řada. Tento nástroj má velké spektrum využití. Můžeme ho používat na tvorbu dokumentu XML, kontrolu well-formed a validaci vůči XSD schématu. Pokud napíšeme XSL šablonu, je možné vstupní dokument zformátovat na výstupní dokument XML. Vestavěný XPath jazyk byl pro mne velkou výhodou. Prováděl jsem pomocí editoru testování XPath dotazů na XML strukturu. Následně jsem tyto dotazy implementoval do vlastní aplikace. Tato funkcionalita mi v editoru velice pomáhala. Odladění dotazů pro plnou funkčnost aplikace nebylo vždy jednoduché.

Jedná se o velmi využitelnou aplikaci. Bohužel se jedná o placený software, i přesto mohu tento software doporučit. Dále je možné použít XMPSpy od firmy Altova, který je k dispozici i jako zásuvný modul pro vývojové prostředí Microsoft Visual Studio nebo Eclipse.

V poslední řadě je k dispozici volně šiřitelný editor yEd Graph Editor. Tento editor je vyvinut společností, zabývající se aplikacemi pro tvorbu grafů. Jedná se o komerční subjekt, který disponuje velkou škálou specializovaných knihoven a aplikací. Za tyto služby si nechává platit v podobě licencí. Dává však k dispozici tento yEd editor, který je specializovaný na vytváření, editaci a modifikaci dokumentů typu GraphML. Je možné tento editor využívat k vytváření dokumentů GraphML[18].

GraphML syntaxi jazyka XML, která se využívá pro tvorbu grafů, jsem využil pro vytvoření vlastního útočného stromu. Detailní popsání syntaxe viz kapitola 4.7.

## 4.7 GraphML

GraphML je komplexní, obecně psaná a snadno použitelná forma XML syntaxe, která se využívá pro popis grafů. Grafem je také strom, ve kterém dochází k propojení uzlů navzájem.

GraphML se skládá z hlavního jazyka, který slouží pro popis struktury grafu. V této části je na tvůrci rozmyslet si propojení jednotlivých uzlů k vytvoření požadované struktury. Tato syntaxe představuje využitelnou možnost, jak si vytvořit stromovou strukturu útočného stromu. Formát obsahuje mnoho možností, jak uchovat data a potřebné parametry pro zpracování. Uchovávat i tzv. META data, která se využívají při programovém zpracování.

Další možností je jednoduchá rozšiřitelnost. V rámci této syntaxe je předem definována struktura souboru XML, která je kontrolována validátorem proti XML schématu nebo DTD. V případě potřeby je možné toto schéma upravit a dodat specifikace, které jsou vyžadovány. Pro dodatečná data je v rámci restrikcí GraphML použít element <key>, který obsahuje synovský element <data>, který data obsahuje. Data jsou omezena typově. Jsou použity stejné datové typy, které se využívají v syntaxi jazyka Java. Zahrnuje datové typy boolean, int, long, float, double a string. Množina umožňuje uchovávat všechny potřebná data. Tento typ je předem nutné definovat v elementu <key> jako atribut attr.type="string". Všechny specifikace této syntaxe je možná zjistit z dokumentace GraphML Primer[6]. Poskytuje vám přehledný návod jak vytvářet dokumenty typu GraphML.

Využil jsem standardní rozšiřitelnosti pomocí konstrukce <key> a <data>. Dále popisuji, jak jsem tuto konstrukci použil. Ukázka v příloze č. 1, je můj vlastní dokument, který reprezentuje útočný strom[6].

Pro definování a vytvoření výsledného stromu je použito konstrukce <node> a <edge>, která jsou součástí elementu <graph>, viz příloha č. 1. Propojení těchto uzlů, do výsledné hierarchie, se vytvoří pomocí elementu <edge>. Představuje hranu mezi dvěma

uzly. Na příkladu 4.1 vidíme, jak jsou vytvořeny hrany z hlavního uzlu. *ID* uzlu GOAL představuje odkud (source), kam (target) se má vytvořit hrana propojení. Na příkladu jsou definovány čtyři hrany do uzlů s id n1, n2, n3, n4.

```
<edge id="edge0001" source="GOAL" target="n1"/>
<edge id="edge0002" source="GOAL" target="n2"/>
<edge id="edge0003" source="GOAL" target="n3"/>
<edge id="edge0004" source="GOAL" target="n4"/>
```

**Příklad 4.1: Ukázka elementů hran**

V návrhové části je rozepsáno, co muselo být doplněno do dokumentu GraphML. Jakým způsobem se muselo postupovat, proč se dané konstrukce začlenily do dokumentu. Výsledná forma dokumentu je důležitá pro možnost uchování dat.

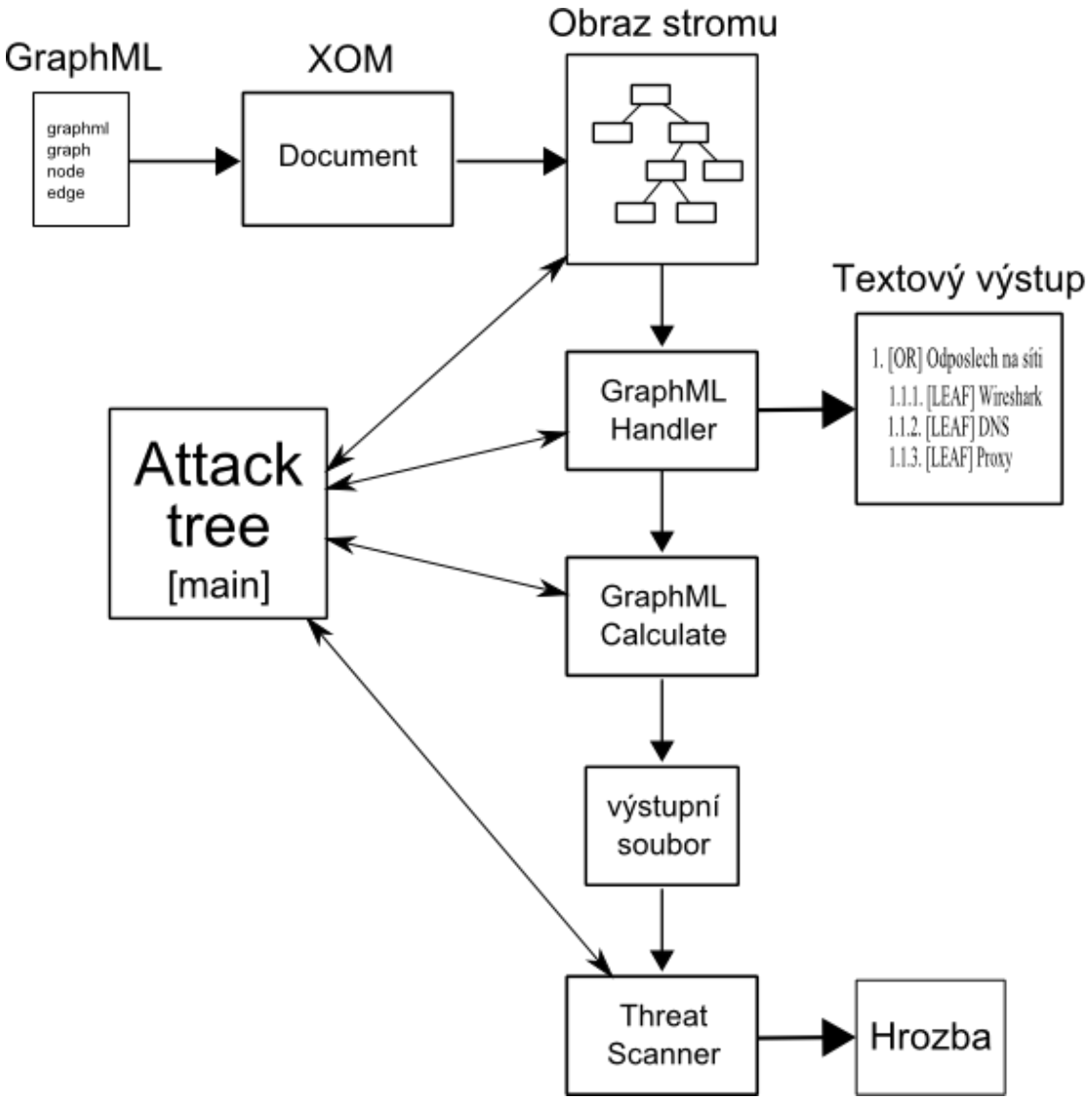
# 5 Návrh řešení

Útočný strom, jako takový, je založen na velmi jednoduchém konceptu formulace rizik. Příklady stromů, jako například fyzické zabezpečení budovy, stromy menšího rozsahu může člověk jednodušeji nakreslit pomocí obyčejné tužky a papíru. Ovšem v případech, kdy se rozsah stromu zvětšuje, se zvětšujícím se systémem zabezpečení, je čím dal tím více obtížné tuto realitu zachytit. Situace se stává méně přehlednou, a proto se člověk v takovém složitém stromu špatně orientuje.

Jiná situace by nastala v případě, kdybychom měli k dispozici nějakou softwarovou utilitu, která by nám pomáhala se v takovém množství dat orientovat. Situace by se výrazně zlepšila. Pokud by podpůrný systém dokázal navíc útočný strom vyhodnotit a na největší rizika upozornit, tak by to uživateli značně usnadnilo práci s útočným stromem. Člověk nemůže mít v paměti všechny informace parametrů tak, jako aplikace.

Z tohoto důvodu jsem pracoval na parseru, který usnadní práci uživateli a umí vyhodnotit útočný strom. Jedná se o programovou část, která se skládá z vlastní aplikace. Aplikace je napsána v jazyce Java ve vývojovém prostředí NetBeans IDE 7.0 za přítomnosti knihovny XOM, která je volně šiřitelná. Knihovnu napsal vývojář Elliotte Rusty Harold. Tuto knihovnu jsem využíval pro extrakci dat, úpravu a výstup z XML dokumentu. Knihovna obsahuje další funkcionality, které mohou být velmi užitečné. Byla využita podpora XPath verze 1.0, a také serializace výstupních dat [17].

Zkoumáním, zkoušením a obohacováním se o další informace vznikl výsledek návrhu, který můžeme vidět na obrázku 5.1. Tento blokový návrh ukazuje, jakým způsobem se vstupní data zpracovávají. Od předložení vstupního souboru se dostáváme až po výstupní formát vyhodnoceného útočného stromu. Pro uživatele byla zvolena textová forma výstupu, kde je hierarchicky vyobrazen útočný strom. V poslední fázi analýzy se výstupní soubor prochází a vyhodnocuje se nejslabší místo celého stromu.




**Obrázek 5.1: Blokové schéma programového zpracování**

# 5.1 Vstupní dokument GraphML

Před začátkem řešení celé problematiky útočného stromu je nutné zvolit a vytvořit si vstupní dokument, který bude reprezentovat vlastní útočný strom. Nastává otázka, zda je vhodné vyvíjet si vlastní dokument s vlastním schématem, nebo využít již hotového konceptu. V případě tvorby vlastní struktury nastává otázka rozšiřitelnosti. Jedná se většinou o proprietální strukturu, která je použitelná pouze v daném řešení. Naopak pokud použijete rozšiřitelnou a hlavně vhodnou strukturu vstupního souboru, tak je toto řešení široce použitelné. Jako řešení vstupního souboru bylo využito GraphML struktury, která je syntaxí jazyka XML.

Jedná se o strukturu, která se využívá při tvorbě grafů a podobných struktur. Její možnosti jsou velmi rozsáhlé. Nespornou výhodou je vestavěná implementace rozšiřitelnosti o potřebná data, která je použita i v mém souboru, viz kapitola 5.1.1.

Podoba tohoto souboru je vidět v příloze číslo 1. Celá syntaxe je jednoduchá a přehledná. Dokument obsahuje pouze základní strukturu, která je vytvořena tímto způsobem:

```
<graphml>
        <key></key>
        <graph>
                <node>
                        <data></data>
                </node>
                <edge>
                </edge>
        </graph>
</graphml>
```

Struktura je dána XSD schématem a rozšíření je možné vytvořit pouze konstrukcí elementu <key> a <data>. Element <key> se definuje na začátku dokumentu mezi elementy <graphml> a <graph>. Element <data> se vkládá do elementu, pro který byl vytvořen. Tímto je možné elementy uzlů a hran obohacovat o požadovaná data.

### 5.1.1 Vlastní rozšíření dokumentu

Element <node>, který představuje uzel stromu, uchovává typ, popis uzlu a hodnotu složitosti. V dokumentu jsou tyto <key> elementy, která uchovávají data. Elementy <edge> uchovávají pouze propojení uzlů.

```
<key id="0" for="node" attr.name="typ" attr.type="string"/>
```

Tato definice obsahuje jedinečný identifikátor *id* v celém dokumentu. Validátor kontroluje jedinečnost. Další hodnota *for* říká pro jaký element je toto rozšíření dostupné. *Attr.name* obsahuje název tohoto rozšíření a poslední záležitostí je určení datového typu. Ve výsledku je možné do elementu <node> přidat konstrukci:

```
<data key="0">GOAL</data>
```

V dokumentu, příloha číslo 1 můžeme vidět, jak a kde je tento element vkládán. Může obsahovat hodnoty typu AND, OR, LEAF nebo GOAL. Vrchol stromu je označen jako GOAL a list stromu jako LEAF.

```
<key id="1" for="node" attr.name="jméno" attr.type="string"/>
```

Definice jména uzlu. Jedná se o popis a pojmenování uzlu, jak je vidět dále.

```
<data key="1">Přístup k web aplikaci</data>
```

Atribut typ útoku není v implementaci použit, ale je zde v dokumentu obsažen. Dalším krokem bylo realizovat možnost přidávání protiopatření. To by ovlivňovalo složitosti ve stromu. Pokud by se vytvořilo protiopatření, byl by útok složitější. Definice je tato:

```
<key id="2" for="node" attr.name="typ_útoku" attr.type="string">
    <default>útok</default>
</key>
```

Je to možný nápad pro další rozšiřování této implementace.

```
<key id="3" for="node" attr.name="složitost" attr.type="long">
    <default>1</default>
    <and>SUM</and>
    <or>MIN</or>
</key>
```

Hlavním prvkem celého rozšíření struktury je parametr složitost, který uchovává důležitou informaci pro vyhodnocování celého útočného stromu. Tento element <key> obsahuje i další informace. Pokud do uzlu <node> přidáme <data> a nezvolíme hodnotu, tak se standardně vloží hodnota 1. Další informace jsou pro potřeby procházení aplikací, ve které je definováno, co se má s touto hodnotou provádět. Pokud je to uzel typu AND, provede se součet hodnot a v případě uzlu OR se vybírá minimální hodnota.

Pro vytvoření stromu je v dokumentu použito konstrukce <node> a <edge>. Přidáváním a upravováním těchto elementů se výsledný strom definuje. Celou strukturu a propojování elementů kontroluje validátor proti XSD schématu. Nemůže se stát, že by například hrana odkazovala na neexistující uzel stromu. Stejně tak nemůže nastat, že není definován počáteční uzel hrany.

Všechny operace se vstupním dokumentem byly prováděny v prostředí Oxygen XML editor. Vytvoření, editace, validace a testování XPath výrazů.

## 5.2 Funkce implementace

Vlastní návrh implementace se skládá z několika částí. Ve výsledku celého snažení by měla být k dispozici aplikace, která bude napomáhat v zabezpečování. Záleží pouze na uživateli, jak tento nástroj využije a čeho bude chtít dosáhnout. Funkční celek bude umět projít celou strukturu stromu, vyhodnotit všechny uzly a cesty, vyobrazit tento útočný strom a v poslední fázi říci, kde se nachází nejslabší místo systému zabezpečení. K tomuto všemu by měl pracovat navíc efektivně a dostatečně rychle. Byl kladen nárok na rychlost zpracování XML dokumentu zvolenou knihovnou XOM. Jedná se o jeden z nejrychlejších nástrojů ke zpracování XML. Vše je dáno jeho implementací [17].

Celá cesta od začátku do konce se skládá ze 4 dílčích částí:

1. Zpracování dokumentu, který reprezentuje útočný strom (GraphML)
2. Vyhodnocení uzlů stromu
3. Vyhledání nejslabšího místa stromu
4. Výstupy

### 5.2.1 Zpracování vstupního dokumentu

Jedná se o stěžejní část celé implementace. V kapitole 5.1 je vysvětleno, jak takový soubor vypadá a co musí obsahovat. Soubor představuje pro aplikaci vstupní data. Tento vstupní soubor, v podobě XML dokumentu, vytvořený v rámci syntaxe GraphML, představuje pro aplikaci objekt, který uchovává data. V aplikaci nebyl vytvářen vlastní objektový model, který bude data ze souboru uchovávat. Soubor se chová pro aplikaci jako objekt, který tyto data uchovává. Z obrázku 5.1 je patrné, že vstupní soubor se transformuje na XOM dokument. Z dokumentu se v paměti stroje vytvoří takzvaný obraz, který představuje objekt. K manipulaci s daty se využívá knihovna XOM. Aplikace si tyto data vybírá a zpracovává.

Pro tuto část zpracování vstupního dokumentu byla navrhnuta třída GraphMLHandler. Tato třída je navržena pro extrakci a zpracování dat ze vstupního dokumentu. Vstupní dokument jí předává hlavní třída AttackTree. Z této hlavní třídy je vše řízeno.

Třída GraphMLHandler navíc při zpracování vytváří textovou podobu útočného stromu. Tuto podobu následně vypisuje, viz obrázek 5.2.

Průchod celé struktury je realizován pomocí rekurze. Tato metoda totiž poskytuje možnost projít strom podobně jako metoda procházení stromu do hloubky. Procházejí se postupně jednotlivé větve stromu, zleva doprava. Zároveň je možné ihned vytvářet hierarchickou strukturu stromu.

Důležitou součástí zpracování dat XML struktury byl jazyk XPath. Tento jazyk, který jsem musel zprovoznit v knihovně XOM, byl velmi nápomocný. Velkou mírou ovlivnil rozsah zdrojového kódu, který by byl jinak velmi dlouhý a tudíž nepřehledný. V případech, kde by bylo nutné používat složité programové konstrukce se sérií vnoření, stačilo napsat jeden dotaz na XML dokument a vše bylo rázem vyřešeno. Lehkost a efektivita toho jazyka je nezměrná.

Chtěl bych pouze upozornit, že ve verzi jazyka 1.0, která je podporována knihovnou XOM 1.2.7, je nezbytně nutné v programu vždy definovat jmenný prostor XML souboru. Deklarace:

*private static XPathContext xpathcont = new XPathContext("default", "http://graphml.graphdrawing.org/xmlns");*

Pokud toto neuděláme, dotazy budou vracet prázdné hodnoty. Slovo default je zástupným znakem NS. Verze 2.0 nic takového nepotřebuje. V editoru Oxygen funguje bez NS. Výsledný XPath výraz vypadá například takto:

*//default:node[@id='GOAL']*

Tento výraz vybere uzel <node>, který má id hodnotu GOAL. Proti verzi jazyka XPath 2.0 je nezbytné v konstrukci výrazu uvést slovíčko *default*, které reprezentuje NS.

### 5.2.2 Automatické počítání parametrů uzlu

Tato funkce implementace zajišťuje, aby všechny uzly útočného stromu obsahovaly správné hodnoty složitostí. Celá struktura se rekurzivně prochází a výsledkem je kompletně vyhodnocený útočný strom. Při procházení stromu se zohledňují hodnoty bran AND a OR, které ovlivňují výběr správných hodnot. V kapitole 3.1 bylo vysvětleno, jak se tyto hodnoty generují do těchto uzlů.

Po zpracování vstupního souboru třídou GraphMLHandler se ten samý soubor předloží třídě GraphMLCalculate, viz obrázek 5.2.

Třída je navržena tak, aby ze vstupního souboru vytvořila výstupní soubor, který bude představovat vyhodnocený útočný strom. Zpracování struktury stromu probíhá tak, jak bylo popsáno v kapitole 2.3.5. Celá struktura stromu se zpracuje a do uzlů označených AND a OR se vloží správné hodnoty synovských uzlů.

Název výstupního souboru se odvíjí od názvu vstupního souboru. Výsledek se skládá ze slova evaluated_ a za něj je přidán název vstupního souboru. Příkladem může být evaluated_webapp.xml.

Výsledný soubor se využívá jako forma vstupních dat pro třídu ThreatScanner, která na základě tohoto kompletně vyhodnoceného útočného stromu dokáže najít nejslabší místo stromu.

### 5.2.3 Vyhodnocení rizika

Vyhodnocení rizika probíhá tak, že se kompletně vyhodnocený strom rekurzivně prozkoumá. Hrozby, které obsahují nejmenší hodnoty složitostí, vybereme a upozorníme na ně. Toto upozornění se uskuteční na konci celého zpracování. Výstupem bude výpis do konzolového okna ve formátu, který vidíme na příkladu 5.1.

## 5.3 Výstupy

Výstupem programu po zpracování XML struktury může být například tato textová podoba stromu, která nám přehledně znázorňuje útočný strom. Můžeme přehledně vidět jednotlivé hrozby systému. Součástí tohoto zobrazení jsou i logické operátory, které nám říkají, které akce je nutné kombinovat, aby bylo dosaženo úspěšně cíle. Pro lepší orientaci ve výstupu je k uzlům přiřazena číselná kombinace. Toto číslo odpovídá umístění v hierarchické struktuře stromu.

```
[GOAL] Přístup k web aplikaci
1. [OR] Odposlech údajů
        1.1. [OR] Odposlech na síti
                1.1.1. [LEAF] Wireshark
                1.1.2. [LEAF] DNS spoofing
                1.1.3. [LEAF] Proxy
        1.2. [AND] Odposlech za dveřmi
                1.2.1. [LEAF] Odposlecha konverzace na místě určení
                1.2.2. [LEAF] Zachytit údaj ve vhodný okamžik
2. [OR] Získání údajů
        2.1. [LEAF] Sociální inženýrství
        2.2. [LEAF] Podplacení administrátora
3. [OR] Uhádnutí
        3.1. [LEAF] Brutal Force
        3.2. [LEAF] Slovníkový útok
        3.3. [LEAF] Implicitní údaje
        3.4. [AND] Odhadnutí údajů
                3.4.1. [LEAF] Znalost schématu tvorby autentizačních údajů
                3.4.2. [LEAF] Znalost uživatele
4. [OR] Obejít aut. schéma
        4.1. [LEAF] Změna URL
        4.2. [LEAF] Modifikace parametrů
        4.3. [LEAF] Předpovězení SSID
        4.4. [LEAF] SQL Injection
```

**Obrázek 5.2: Textová podoba útočného stromu**

Výsledkem celé analýzy je výstup do terminálu nebo konzolového okna. Finální podoba analýzy útočného stromu obsahuje textový výpis stromu, který je vidět na obrázek 5.1 a další informací je výpis největší hrozby. Tato hrozba je výsledkem zpracování vyhodnoceného stromu. Výstupní formát je zvolen takto:

**(GOAL) Přístupové údaje k webovské aplikaci** ← Uhádnutí ← **Implicitní údaje (list)**

**Příklad 5.1: Výpis hrozby**

V dalších fázích vývoje aplikace můžeme aplikaci obohatit o další možnosti výstupů, které budou více napomáhat v orientaci a zabezpečování systémů. Pokud se rozhodneme pro realizaci grafického vykreslování grafu, měli bychom si uvědomit, že tato forma výstupu je mnohem náročnější. Nejde pouze o vykreslení, ale musíme vytvořit algoritmus, který bude přizpůsobovat zobrazení grafu jeho rozsahu a velikosti. Toto pojednání může být realizováno jako další rozšíření této aplikace. Jedná se o rozsáhlou problematiku, nejen o útočných stromech, ale také algoritmizaci a programovém zpracování.

# 6 Implementace

Výsledkem předešlého snažení bylo vytvoření software, který bude procházet a vyhodnocovat data útočného stromu. Poskytne nám možnost vyhodnotit riziko mnohem rychleji a tím eliminovat rizika na minimum. Výsledným produktem je volně šiřitelná programová část, která byla zhotovena v prostředí NetBeans. Programovacím jazykem se stala JAVA, pro její velkou rozmanitost a rozšiřitelnost. Pro programovací jazyk JAVA je také napsána zvolená knihovna pro zpracování vstupního dokumentu, viz kapitola 4.3. Vstupním souborem byl zvolen XML dokument, pro jeho možnosti zpracování a uchování dat. Vše co bylo v rámci implementace zvoleno, je vybráno s ohledem na další možnosti rozšíření aplikace nebo modifikace. Parser, který je hlavní částí práce, můžeme dále použít jako součást další programové aplikace. Vše bylo vytvářeno s ohledem na možné další použití.

## 6.1 Popis implementace

Vlastní program se skládá ze 4 tříd. Hlavní třídou, která řídí proces zpracování, je AttackTree.java. Obsahuje pouze hlavní třídu main. Tato třída inicializuje vytvoření dokumentu XOM ze vstupního dokumentu XML. Vstupní soubor se předává aplikaci formou parametru třídy main, args[0]. Pro spuštění analýzy byl zhotoven dávkový soubor (spustit_analyzu_at.bat), který spustí proces zpracování. V kapitole 6.2 můžeme vidět, jak takové spuštění provést. V posledním kroku analýzy se vypíše výstup do konzole, ve formátu, který byl popsán v kapitole 5.3.

V další části popíši, jak aplikace funguje a co je v jednotlivých třídách důležité. Navíc byla vygenerována dokumentace Javadoc z vývojového prostředí NetBeans. V dokumentaci jsou jednotlivé metody a třídy detailněji okomentovány.

### 6.1.1 AttackTree

Třída AttackTree je hlavní třídou programu, která komunikuje s ostatními třídami a ovlivňuje běh aplikace, viz obrázek 5.1. Je navíc důležitou součástí spustitelného souboru, protože obsahuje hlavní třídu main. Na konci kompilace bude vygenerován spustitelný soubor ve formátu parser_bak.jar. Soubor bude obsahovat i knihovnu XOM, která není součástí Java balíčku. Vše je zkompilováno tak, aby bylo možné analýzu spustit nad jakýmkoliv vstupním souborem.

Vstupní soubor se předává aplikaci názvem souboru, tedy pomocí parametru arg[0] třídy main. Pokud nezadáme vstupní soubor, aplikace nám sdělí tuto hlášku:

*Použijte formát příkazu: java -jar parser_bak.jar vstupniSoubor.xml*

Tato hláška je zde jako ošetření vstupu. V další části se dovíme, že je toto také ošetřeno v dávkovém soboru (spustit_analyzu_at.bat), který v případě nezadání souboru aplikaci nespustí.

### 6.1.2 GraphMLHandler

GraphMLHandler je třída, která umožňuje extrahovat data ze vstupního souboru. Tímto souborem je XML dokument webapp.xml, který představuje útočný strom. Celou tuto strukturu je nutné projít a získat potřebná data za účelem dalšího zpracování. Metoda

procházení stromu je založena na metodě procházení stromu do hloubky. Je využito rekurze, aby se zajistilo, že bude celý strom prohledán. Hlavní metoda této třídy je:

*public static void prochazeniUzluStromu(Node uzel, int odsazeni, StringBuffer sb)*

Jedná se o rekurzivně volanou metodu, která předává aktuální uzel průchodu a dále jej zpracovává. Třída navíc pomocí StringBufferu a pomocné odsazení generuje textovou podobu výstupu, která je uvedena na obrázku 5.2. Číselné kombinace uzlů se generují z vnitřního iterativního procházení potomků a z vyskočení z rekurze.

Další důležitou částí je metoda *vratPotomkyRodice,* která pomocí XPath výrazu vrací potomky rodiče. Metoda se využívá v rekurzivním průchodu.

*public static Nodes vratPotomkyRodice(Node rodic)*

Výsledek je vypsán do konzole a program pokračuje ve vyhodnocování stromu.

### 6.1.3 GraphMLCalculate

Nejsložitější částí celé aplikace byla implementace této třídy. Volba vhodné programové konstrukce pro zpracování byla namáhavá. Nebylo lehké zvolit algoritmus procházení stromu. Nakonec byla zvolena opět rekurzivní metoda průchodu, která prochází strom do hloubky.

*public static int vypocetUzluStromu(Node uzel)*

Tato metoda předává jako parametr aktuální uzel průchodu, a navíc vrací hodnotu typu int, která představuje složitost útoku uzlu. V případě listu stromu se tato hodnota pouze vrací. U uzlu typu AND a OR, dochází k vnoření a dalšímu porovnání nebo součtu hodnot. Výsledná hodnota je při skončení rekurze vrácena. Pomocnou metodou při úpravě uzlů ve stromu je metoda:

*public static void upravUzel(Node uzel, int hodnota)*

V případě potřeby, při rekurzivním průchodu, metoda vloží do aktuálního uzlu element

*<data key="3">6</data>*

Tento element představuje hodnotu složitosti. O jeho vytvoření se stará metoda vytvorHodnotuUzlu(int hodnota). Její přidávání se neprovádí do vstupního dokumentu, ten se používá jako zdroj informací, nýbrž do proměnné *Document vystup* v rámci této třídy. Třída *Document* je součástí knihovny XOM a reprezentuje vstupní dokument v programu. Tento dokument se vytváří ze vstupu a je k tomu použita metoda *build* ze třídy *Builder*.

Nejprve se prochází levá větev stromu, vyhodnotí se a pokračuje se další větví. Průchod probíhá zleva doprava. Na obrázku 2.1 nebo v souboru na CD vidíme strukturu uspořádání a listy stromu představují základní útoky. Tyto základní útoky se dosazují při vyhodnocování do postupných kroků útoku, až se všechny uzly kompletně vyhodnotí. Výsledkem je pro nás vyhodnocený útočný strom. Ve třídě je reprezentovaný proměnnou *Document vystup*.

V poslední fázi zpracování je volána metoda *vystup(String nazevSouboru),* která realizuje zápis vyhodnoceného dokumentu v programu do souboru. Soubor, který má název evaluated_webapp.xml, je vytvořen serializací dat z instanční proměnné *Document vystup*. Podoba vyhodnoceného souboru se nachází na přiloženém CD.

Serializace probíhá pomocí třídy Serializer z knihovny XOM. Tento proces serializace je nutné uskutečnit, protože pokud bychom uložili data pouze ve formátu String, jak jsem v prvním pokusu zkoušel, nejednalo by se o well-formed XML dokument. Při určitém nastavení kódování je nutné výstupní data serializovat. Pro můj dokument bylo použito UTF 8 kódování, která podporuje diakritiku českého jazyka. Tato hodnota je uložena v hlavičce XML dokumentu.

*<?xml version="1.0" encoding="UTF-8"?>*

Pokud bychom tento proces serializace neudělali, v hlavičce dokumentu by informace o kódování nebyla, a tak by nám validační software hlásil chybu: Dokument není well-formed nebo validní. To znamená, že neodpovídá syntaxi jazyka XML, proto by nebylo možné spustit analýzu útočného stromu. Nejednalo by se o validní soubor.

### 6.1.4 ThreatScanner

Pokud máme k dispozici po předchozím úspěšném zpracování soubor, který představuje vyhodnocený útočný strom, je možné vybrat a následně i vypsat největší hrozbu systému.

V tomto případě se vybírá útok s nejmenší složitostí. Funkce implementace je napsána tak, aby se vybrala pouze tato možnost. V případě, že je ve stromu cesta útoku, která má stejnou nejmenší hodnotu, tak se vypíší obě. Tuto funkci implementuje metoda *vycetHrozeb(Node n, int hodnota)*. Třída prochází soubor evaluated_webapp.xml a vybírá nejslabší cesty stromu, které na konci procesu vypíše z proměnné StringBuffer sb. Formát výstupu byl zmíněn v návrhové části práce, viz příklad 5.1.

Vybráním nejslabšího místa stromu zjistíme, kam by se naše pozornost měla upírat a co za hrozbu bychom měli eliminovat. Tento systém výběru je takto zvolen proto, abychom postupovali v zabezpečování systému od největších hrozeb. Pokud odstraníme tuto hrozbu, spustíme v další fázi opět analýzu stromu a aplikace nás upozorní na další podobnou nebo složitější možnost útoku. Pokud budeme tímto způsobem pokračovat do úplného eliminování stromu, riziko průniku do systému zabezpečení se sníží nebo úplně odstraní. Toto je cílem veškerého zabezpečování.

V rámci dalšího vývoje, můžeme doimplementovat další funkcionality třídy, které nám budou podle našeho přání tyto útoky vybírat. Například nás bude zajímat pouze útok od 1 do 3, protože naše možnosti jsou omezeny financemi, atd.

## 6.2 Spuštění aplikace

Spuštění aplikace (analýzy) lze provést 3 způsoby. Jednak je možné spustit balíček projektu parser_bak ve vývojovém prostředí NetBeans, v takovém případě musím vložit knihovnu XOM, nebo můžeme využít přiloženého archívu parser_bak.jar, který je zkompilován v prostředí NetBeans. Archív navíc obsahuje knihovnu XOM, kterou není potřeba dodatečně vkládat do vývojového prostředí. Tento soubor je možné spustit na každém počítači. Kromě nainstalovaného balíčku java jdk a příkazové řádky není potřeba zvláštního vybavení. Spuštění v tomto případě provedeme příkazem:

*java -jar parser_bak.jar webapp.xml*

V lokálním adresáři je potřeba mít soubor graphml.xsd a graphml-structure.xsd. Jde o schémata, která jsou důležitá pro validaci vstupního souboru. Při průběhu programu se v adresáři vytvoří evaluated_webapp.xml, který reprezentuje vyhodnocený útočný strom.

Třetí a asi nejkomfortnější možností je spuštění programu pomocí dávkového souboru spustit_analyzu_at.bat. Tento soubor vypadá následovně:

```
@ECHO OFF
@chcp 1250

IF "%1"=="" GOTO BEZPAR
IF EXIST %1 GOTO JO
  ECHO ---------------------------------------------
  ECHO Soubor %1 neexistuje
  ECHO ---------------------------------------------
GOTO KONEC Přeskoč zbývající příkazy

:JO
  ECHO ---------------------------------------------
  ECHO Soubor %1 existuje
  ECHO ---------------------------------------------
  java -jar parser_bak.jar %1
  GOTO KONEC

:BEZPAR
  ECHO ---------------------------------------------
  ECHO Chybí parametr vstupního souboru
  ECHO ---------------------------------------------

:KONEC
```

V podstatě se jedná o automatizovaný postup zadávání příkazů. Do příkazové řádky napíšeme *spustit_analyzu_at.bat webapp.xml*. Tímto říkáme, aby se vykonala dávka s parametrem vstupního souboru. V dávce je ošetřen vstup, viz výpis souboru. Program proběhne a do příkazové řádky se vypíše výstup, který je popsán v návrhové části. Kapitola 5.3.

Důležitou součástí správného spuštění je nastavení správné znakové sady v příkazovém řádku. Nastavením písma Lucida Console a použití příkazu chcp 1250, který nastavuje znakovou sadu, bude vše vypsáno správně. V případě použití dávky je nutné nastavit jen správné písmo nebo kódování znaků.

V příloze č. 2 můžeme vidět konečnou podobu výpisu do konzole, která zahrnuje textovou podobu stromu a výpis hrozby, která má nejmenší složitost pro útočníka. Tím pádem je pro bezpečnost systému největší hrozbou.

# 7 Testování

Testování probíhalo jako opravdová forma zabezpečení systému. Testovacím souborem byl webapp.xml, který je součástí přiloženého CD a představuje pro nás útočný strom. Po každé analýze hrozeb jsem ze stromu odstranil největší hrozbu a opět nechal aplikaci analyzovat útočný strom. Aplikace fungovala přesně tak, jak byla navržena. Uzly AND a OR byly správně vyhodnoceny. Po odstranění vyhodnocené hrozby, která byla vypsána do konzole, následoval další průchod stromu. Výsledkem byla jiná hrozba s vyšší úrovní složitosti. Aktualizace vstupního souboru probíhala v editoru Oxygen XML editor. Výstup do konzole vždy odpovídal vstupnímu souboru. Každý krok hodnocení poskytoval přesný obraz vstupního souboru a tudíž i útočného stromu.

Příkladem testování může být odstranění základních hrozeb jako odposlech na síti pomocí Wiresharku a uhádnutí implicitních údajů. Pokud tyto hrozby z útočného stromu odstraníme, aplikace nás upozorní na další možnosti napadení. V mém případě se bude jednat o uhádnutí přístupových údajů pomocí slovníkového útoku. Pokud budeme realizovat protiopatření a eliminovat tyto hrozby, můžeme opět tuto hrozbu ze stromu odstranit. V rámci testování aplikace jsem postupně odstraňoval uzly, až jsem dospěl k úplnému odstranění hrozeb. Samozřejmě je také možné přidávání a editování stromu. Testování taktéž proběhlo. Aplikace se chovala přesně tak, jak byla navržena a je plně funkční.

# 8 Možný budoucí vývoj

Vývoj software je neustálý proces, který téměř nikdy nekončí. Při návrhu si definujeme, jak by aplikace měla vypadat a snažíme se tuto ideu udržet. Během vytváření aplikace nás mohou napadat nové myšlenky, které můžou ovlivňovat výsledný produkt našeho snažení. Naším nepřítelem, při časovém plánu projektu, je právě čas. Navzdory veškerému snažení není možné stihnout vše do aplikace zaimplementovat. Dále uvádím, jakými nápady je možné aplikaci obohatit a vylepšit.

- Vytvoření grafického uživatelského prostředí by bylo pro uživatele příjemnější. V souvislosti s tímto pohledem by stálo za zvážení, vytvoření editoru, který by umožňoval náhled i editaci útočného stromu. Nastává však značný problém při vykreslování stromu, viz kapitola 5.3, kde je potřeba vymyslet algoritmus vykreslování.

- V rámci projektu GraphML je k dispozici několik knihoven, které umožňují pomocí šablon transformace XSLT vytvářet graf pomocí SVG. Tyto knihovny jsou určeny pouze pro malé grafy. V případě modifikace by mohly být využity pro kreslení grafu útočného stromu. Jedná se o algoritmus určení pozic objektů a jejich vykreslení pomocí SVG.

- Pokud bychom nechtěli vytvářet grafický editor, mohli bychom pouze využít JTree komponentu jazyka JAVA, která by nám umožnila data vstupního souboru načíst a pomocí vystavěných metod podobu stromu editovat. Vzhled stromu by odpovídal obrázku 5.2, jen s tím rozdílem, že by byla možná dialogová interakce. Výsledné funkce by záležely pouze na tvůrci aplikace.

- Pokud bychom se chtěli přiblížit k reálnějšímu obrazu útočného stromu. Měli bychom vymyslet a navrhnout lepší parametr rozhodování. Příkladem může být jistý koeficient výhodnosti pro útočníka, který bude zahrnovat větší spektrum vstupních parametrů, viz kapitola 3.2.

- Další možností je zavedení více rozhodujících parametrů. Tím budeme muset změnit vyhodnocovací algoritmus a navíc bude nutné zavést priority, kterými řekneme, co je pro náš, jako pro útočníka (bezpečnostního technika), důležitější. Zda jsou pro nás důležité peníze, na úkor delšího času útoku, atp.

- Další možností rozšíření implementace je definování útočníka (útoku). Řekněme, že máme několik parametrů útoku (peníze, vědomosti, čas, speciální vybavení). Pokud předem specifikujeme, jaké jsou hodnoty těchto parametrů, můžeme ze stromu přesněji vybrat hrozbu, která jim odpovídá. Toto rozšíření je závislé na předchozím vytvoření rozhodující parametrů a priorit. Nejedná se o jednoduchou vlastnost aplikace.

- Vytvořením obraného uzlu budeme simulovat protiopatření, které bude přirozeně zvyšovat složitost útoku. Tím budeme zaznamenávat přesněji náš momentální postup zabezpečování.

# 9 Závěr

Kyberterorismus je pojem, který v dnešní době slýcháme velmi často. Neustálé zlepšování schopností útočníků nás ohrožuje čím dál víc. Je proto nezbytné podnikat taková protiopatření, která se budou snažit tato napadení eliminovat.

Cílem práce bylo vytvoření aplikace, která bude pomáhat uživateli vyhodnocovat rizika systému zabezpečení. Nezbytným předpokladem pro úspěšné provedení zabezpečení je bezchybná realizace všech přijatých opatření. V praxi často dochází ke kombinaci jednotlivých bezpečností (informační, fyzická, administrativní,…).

Při práci s útočnými stromy byla provedena analýza hrozeb konkrétního systému. Po získání potřebných dat útočného stromu byly tyto hrozby ohodnoceny a poskytnuty parseru. Ten data zpracoval a útočný strom určil největší hrozbu daného systému. Tato informace může sloužit bezpečnostnímu technikovi k realizaci konkrétního zabezpečení této hrozby. Celý proces zabezpečení je ukončen, až když útočný strom neobsahuje žádnou hrozbu.

Fungování aplikace bylo otestováno na množině konkrétních útoků, které byly vybrány pro realizaci útočného stromu. Procházení stromu proběhlo v pořádku, čímž se ověřila implementace řešení parsování XML souboru GraphML. Tímto ověřením byl cíl bakalářské práce splněn.

V případě, že bychom vytvořili skupinu útočných stromů, které by popisovaly bezpečnost jednotlivých systémů (např. zabezpečení domácího routeru, access pointu, atp.), mohli bychom dát aplikaci k dispozici běžnému uživateli, který by správným postupem zabezpečil svůj systém a docílil tak zabezpečení, jako kdyby ho prováděl bezpečnostní technik.

Celá tato aplikace je chápána jako volně šiřitelný software. Na akademické půdě není takové řešení realizováno. Práce by mohla přispět k dalšímu vývoji aplikací, které budou sloužit k rozšiřování povědomí o zabezpečování a nebezpečí ztráty dat.

Pokud by se do budoucna podařilo rozšířit funkce aplikace, jak bylo uvedeno v předchozí kapitole, jednalo by se o velmi užitečnou a v praxi žádanou aplikaci. S dalšími vlastnostmi by se použití zlepšilo a dosáhlo by se lepší efektivity zabezpečení. V případě, že budeme aplikaci dále rozšiřovat a obohacovat, můžeme očekávat její vylepšenou verzi.

# 10 Použité zkratky

AT - attack tree (útočný strom)

ET - event tree (událostní strom)

FT - fault tree (strom selhání)

ETA - event tree analysis (analýza stromu událostí)

FTA - fault tree analysis (analýza stromu selhání)

TLE - top level event (hlavní cíl útoku)

parser - programová aplikace, která prochází dokument určitého typu

NS - Namespace (jmenný prostor XML dokumentu)

XML - Extensible Markup Language (rozšiřitelný značkovací jazyk)

XPath - XML Path Language (XML dotazovací jazyk)

GUI - Graphical User Interface (grafické uživatelské prostředí)

SVG - Scalable Vector Graphics (škálovatelná vektorová grafika)

XSLT - eXtensible Stylesheet Language Transformations (rozšiřitelný transformační jazyk)

# 11 Bibliografie


[1] SCHNEIER, Bruce. Attack Trees [online]. Dr. Dobb's Journal. December 1999 [cit. 2011-01-30]. Modeling security threats. Dostupné z WWW: <http://www.schneier.com/paper-attacktrees-ddj-ft.html>

[2] Amenaza Technologies Limited [online]. 2001 [cit. 2011-01-30]. Attack Tree Methodology. Dostupné z WWW: <http://www.amenaza.com/methodology_2.php>

[3] OPEL, Alexander. Toengel.net [online]. Otto-von-Guericke University Magdeburg : March 2005 [cit. 2011-01-30]. Design and Implementation of a Support Tool for Attack Trees. Dostupné z WWW: <http://www.toengel.net/internship/data/report.pdf>

[4] PALLOS, Michael S. The Business Forum [online]. 2003 [cit. 2011-01-30]. Attack Trees: It's a Jungle out there. Dostupné z WWW: <http://www.bizforum.org/whitepapers/candle-4.htm>

[5] MOORE, Andrew P. ; ELLISON, Robert J.; LINGER, Richard C. Software Engineering Institute [online]. March 2001 [cit. 2011-01-30]. Attack Modeling for Information Security and Survivability. Dostupné z WWW: <http://www.sei.cmu.edu/library/abstracts/reports/01tn001.cfm>

[6] The GraphML File Format [online]. 2001 [cit. 2011-01-30]. Dostupné z WWW: <http://graphml.graphdrawing.org/>

[7] Co je to riziko a analýza rizik - BusinessInfo.cz [online]. 27.12.2006 [cit. 2011-04-06]. Co je to riziko a analýza rizik. Dostupné z WWW: <http://www.businessinfo.cz/cz/clanek/rizeni-rizik/co-je-to-riziko-a-analyza-rizik/1001617/42740/>

[8] Extensible Markup Language. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 13. 7. 2004, last modified on 23. 2. 2011 [cit. 2011-04-12]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Extensible_Markup_Language>

[9] SCAMBRAY, Joel; SHEMA, Mike. *Hacking bez tajemství*. Brno : Computer Press, 2003. 360 s.

[10] KISZKA, Bogdan. *1001 tipů a triků pro programování v jazyce Java*. Vydání první. Brno : Computer Press, 2003. 519 s. ISBN 80-7226-989-5.

[11] MCLAUGHLIN, Brett. *Java & XML*. 2nd Edition. Sebastopol (USA) : O'Reilly & Associates, 2001. 509 s. ISBN 0-596-00197-5.

[12] BRADLEY, Neil. *XML : kompletní průvodce*. První vydání. Praha : Grada Publishing, 2000. 540 s. ISBN 80-7169-949-7

[13] MARCHAL, Benoit. *XML v příkladech*. První vydání. Praha : Computer Press, 2000. 447 s. ISBN 80-7226-332-3.

[14] BENZ, Brian; R. DURANT, John. *XML Programming Bible*. New York : Wiley Publishing, Inc., 2003. 945 s. ISBN 0-7645-3829-2.

[15] HEROUT, Pavel. *Učebnice jazyka Java*. Třetí rozšířené vydání. České Budějovice : KOOP, 2007. 381 s. ISBN 978-80-7232-323-4.

[16] SMEJKAL, Vladimír; RAIS, Karel. *Řízení rizik ve firmách a jiných organizacích*. 3., rozšířené a aktualizované vydání. Brno : Grada Publishing, a.s., 2009. 360 s. ISBN 978-80-247-3051-6.

[17] RUSTY HAROLD, Elliotte. *XOM* [online]. 2002, 2011 [cit. 2011-11-08]. Dostupné z WWW: <http://www.xom.nu/>.

[18] *YWorks - The Diagramming Company* [online]. 2009 [cit. 2011-11-10]. Dostupné z WWW: <http://www.yworks.com/en/index.html>.

[19] XPath. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 18.8.2005, last modified on 1.6.2011 [cit. 2011-11-10]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/XPath>.

[20] Advanced Technology Solution [online]. 24 January 2010 [cit. 2011-11-19]. FTA & ETA. Dostupné z WWW: <http://www.atsolusi.co.id/index.php?option=com_content&view=article&id=65&Itemid=88>.

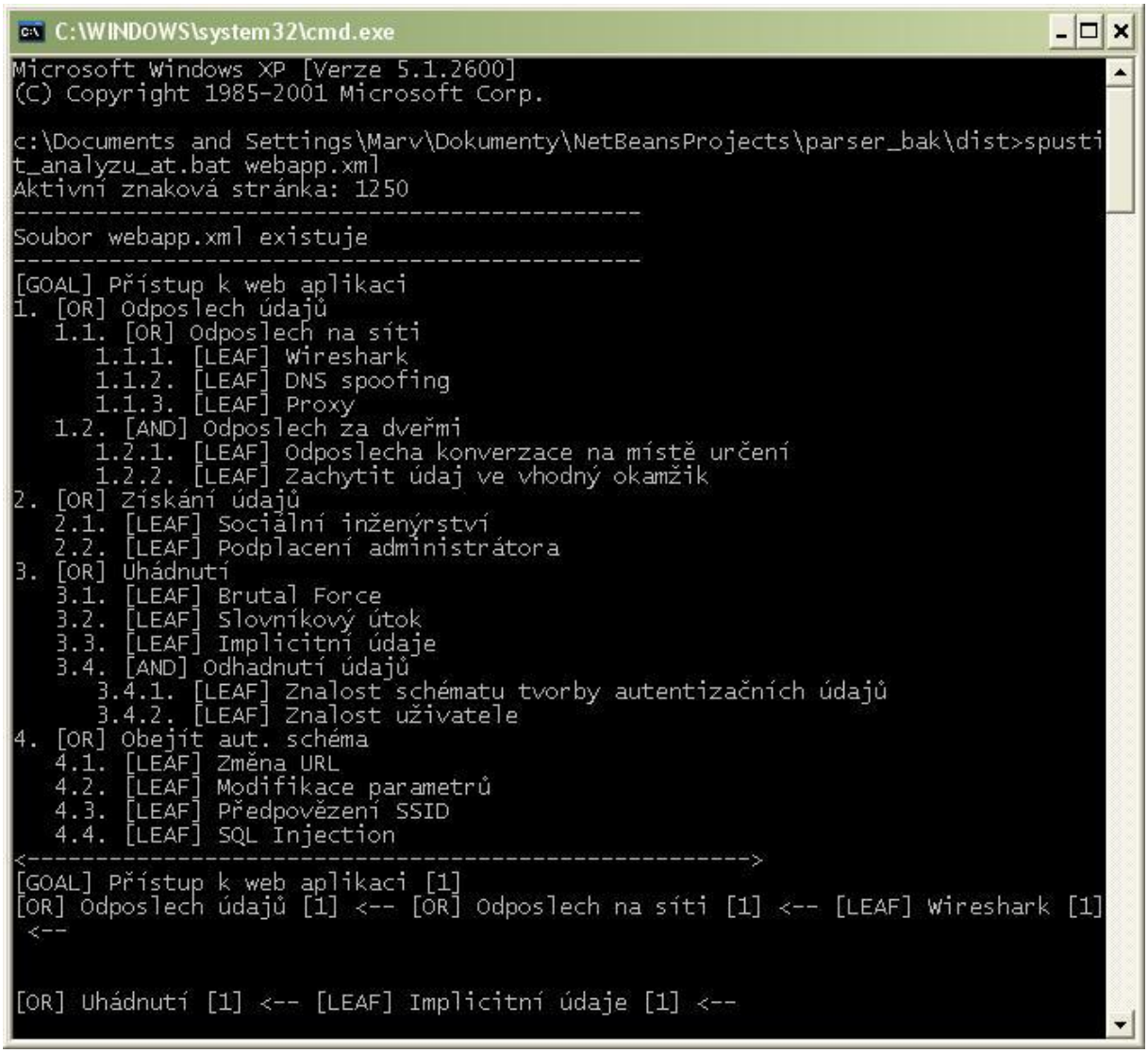
# 12 Příloha

**Číslo 1 – GraphML dokument webapp.xml (souhrn hlavních elementů a rozšíření)**

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="graphml.xsd">


   <!-- Pouze GOAL, AND, OR or LEAF jsou možné. -->
   <key id="0" for="node" attr.name="typ" attr.type="string"/>

   <!-- Popis uzlu -->
   <key id="1" for="node" attr.name="jméno" attr.type="string"/>

   <!-- Útočný nebo obranný uzel  -->
   <key id="2" for="node" attr.name="typ_útoku" attr.type="string">
      <default>útok</default>
   </key>

   <!-- Parametr uzlu složitost útoku  -->
   <key id="3" for="node" attr.name="složitost" attr.type="long">
      <default>1</default>
      <and>SUM</and>
      <or>MIN</or>
   </key>

   <!-- Popis a deklarace útočného stromu -->
   <graph id="Attack_Tree_web"
      parse.nodes="22"
      parse.edges="21"
      parse.maxoutdegree="4"
      parse.maxindegree="1"
      edgedefault="directed">

      <!-- Uzly AND a OR -->
      <node id="GOAL" parse.indegree="0" parse.outdegree="4">
         <data key="0">GOAL</data>
         <data key="1">Přístup k web aplikaci</data>
      </node>
      <node id="n1" parse.indegree="1" parse.outdegree="2">
         <data key="0">OR</data>
         <data key="1">Odposlech údajů</data>
      </node>
      <node id="n2" parse.indegree="1" parse.outdegree="2">
         <data key="0">OR</data>
         <data key="1">Získání údajů</data>
      </node>
```

```
        <node id="n3" parse.indegree="1" parse.outdegree="4">
           <data key="0">OR</data>
           <data key="1">Uhádnutí</data>
        </node>

<!-- Listy -->
        <node id="n7" parse.indegree="1" parse.outdegree="0">
           <data key="0">LEAF</data>
           <data key="1">Wireshark</data>
           <data key="3">1</data>
        </node>
        <node id="n8" parse.indegree="1" parse.outdegree="0">
           <data key="0">LEAF</data>
           <data key="1">DNS spoofing</data>
           <data key="3">5</data>
        </node>
        <node id="n9" parse.indegree="1" parse.outdegree="0">
           <data key="0">LEAF</data>
           <data key="1">Proxy</data>
           <data key="3">6</data>
        </node>

<!-- Hrany -->
        <edge id="edge0001" source="GOAL" target="n1"/>
        <edge id="edge0002" source="GOAL" target="n2"/>
        <edge id="edge0003" source="GOAL" target="n3"/>
        <edge id="edge0004" source="GOAL" target="n4"/>

        <edge id="edge0005" source="n1" target="n5"/>
        <edge id="edge0006" source="n1" target="n6"/>

     </graph>
</graphml>
```

**Číslo 2 – Výstup do konzolového okna**




**Číslo 3 – Obsah CD**

Na přiloženém CD se nachází:

1) Text bakalářské práce (formát pdf) *Mai_Pavel_bak_prace_final.pdf.*

2) Složka *AttackTree_Netbeans_projekt*, která obsahuje kompletní balíček programu *parser_bak*. Tento projekt obsahuje vše potřebné pro spuštění a další vývoj. V hlavní složce jsou XML dokumenty, které představují útočný strom.

3) *AttackTree_NetBeans_projekt_javadoc* obsahuje Javadoc dokumentaci k projektu. Dokumentace se spouští souborem *index.html*.

4) Složka *spustitelny_projekt_jar* obsahuje spustitelný soubor *parser_bak.jar* (jar), který se spouští z příkazové řádky nebo pomocí přiloženého dávkového souboru *spustit_analyzu_at.bat*. Vše ostatní je nutné pro spuštění programu. Ve složce *lib* je obsažena knihovna XOM, která se automaticky přiložila při komprimaci jar archívu.

5) Složka *utocny_strom_graphml_oxygen_projekt* obsahuje celý projekt vstupního dokumentu *webapp.xml*, který byl vytvářen v prostředí Oxygen XML editor. Ostatní soubory jsou schémata XSD, proti kterým se validuje tento soubor.
6) Obrázek *cmd_vystup* znázorňuje výstup aplikace.

7) *xom-1.2.7* - verze knihovny XOM, která byla použita při tvorbě aplikace.