# VYHLEDÁVÁNÍ REGULÁRNÍCH VÝRAZŮ VE VYSOKORYCHLOSTNÍM SÍŤOVÉM PROVOZU

**Jan Kaštil**
Výpočetní technika a Informatika, 1. ročník, prezenční
Supervisor: Doc. Ing. Zdeněk Kotásek, CSc.,

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetěchva 2, 612 66 Brno

ikastil@fit.vutbr.cz

**Abstract**. S rozvojem moderních sítí stoupají požadavky na systémy zajišťující jejich bezpečnost. Téměř všechny pokročilé systémy detekce útoků na síti spoléhají na operaci vyhledávání vzorů v síťovém provozu. Samotné vyhledání vzorů však již moderním systémům na ochranu sítě nedostačuje a je třeba jeho výsledky podrobit dalšímu zpracování. Tím se také mění požadavky na vyhledávací jednotky. Tato práce formuluje požadavky kladené na vyhledávací jednotku vhodnou pro nasazení v moderních systémech a navrhuje možnou strukturu takové jednotky.

**Keywords**. Regulární výraz, Intrusion Prevention, Snort, Bro

## 1 Úvod

Bouřlivý rozvoj rychlosti počítačových sítí s sebou přináší velké množství nových hrozeb. Ať již se jedná o počítačové viry, útoky na síťové služby nebo i neúmyslné poškozování sítě například jejím přetěžováním pomocí P2P sítí. Takovou nebezpečnou síťovou aktivitu je třeba identifikovat v reálném čase a využít získaných informací k zastavení útoku a zajištění potřebné kvality síťového přenosu. Tyto služby zajišťuje software označovaný jako IDS (Intrusion Detection System). Pokud je software schopen odhalenou hrozbu také automaticky odstranit, bývá označován jako IPS (Intrusion Prevention System). Tyto systémy mohou pracovat na různých principech, ale poslední experimenty ukazují, že nejlepších výsledků dosahují systémy kombinující více metod rozpoznávání útoku. Základní schéma IDS se pak skládá z několika vrstev. Nejnižší vrstva přijímá data ze sítě a provádí jejich zpracování. Výstupem této vrstvy jsou jednotlivé události (např. nalezení vzoru) či statistiky síťového provozu. Navazující vrstva provádí detekci útoků na úrovni jednotlivých síťových spojení. Některé typy útoků však není možné detekovat ani na této úrovni. Takovým útokem může být například skenování portů. Tyto útoky detekuje poslední vrstva.

Z výše uvedeného popisu struktury IDS je patrné, že vyhledávání regulárních výrazů v paketu je prováděno ihned zpočátku zpracování a není tedy možné využít znalostí získaných ostatními metodami analýzy síťového provozu ke zmenšení zátěže navrhované jednotky a ta proto musí pracovat na rychlosti vstupní linky, tedy 10Gb/s.

Problémem rychlého vyhledávání vzorů se zabývá velké množství výzkumných týmů. Původní práce se zaměřovaly na algoritmy vyhledávání řetězců, ale Sommer a Paxson ve své práci [14] ukázali, že přidáním kontextové informace je možno zvýšit kvalitu IDS systémů. Ačkoliv jsou algoritmy vyhledávání řetězců velmi rozpracované, jejich rozšíření pro regulární výrazy není často možné.

Metody pro vyhledávání regulárních výrazů jsou téměř vždy založené na použití konečných automatů. Tyto metody pak lze rozdělit do dvou základních směrů. Metody založené na nedeterministických konečných automatech (NFA) vyžadují menší množství paměti, ale jejich implementace nepracuje s lineární časovou složitostí. HW implementací NFA lze dosáhnout lineární časové složitosti, ovšem pouze za cenu nárůstu počtu výpočetních jednotek. Použití Deterministických konečných automatů (DFA) se naopak vyznačuje až exponenciálním nárůstem paměťové složitosti oproti NFA.

Při vyhledávání řetězců pomocí konečných automatů mají vyhledávací automaty velmi výhodnou implementační vlastnost – jejich grafický zápis neobsahuje cykly, a proto lze pro jeho uložení do paměti použít metody ukládání orientovaných acyklických grafů. Tímto tématem se zabývá například [15][16][17]. Analýza provedená v rámci [17] však ukázala, že tyto metody nejsou vhodné pro implementaci vzorů založených na regulárních výrazech. Je to způsobeno faktem, že automaty popisující regulární výrazy často obsahují množství krátkých cyklů a smyček.

## 2 Definice problému

Základním požadavkem na vyhledávací jednotku je rychlost vyhledávání. Vyhledávací jednotka musí pracovat na multigigabitové rychlosti, aby byla schopna detekovat vzory i ve velmi objemných tocích, jakými jsou například streaming videa, či přenos velkých souborů. Nejhorší možný případ je pro vyhledávací jednotku je reprezentován jedním tokem, který zabírá veškerou kapacitu linky.

Druhým významným požadavkem je dostatečná velikost množiny vyhledávaných vzorů, aby bylo možné akcelerovat vyhledávání všech vzorů potřebných pro provoz IDS. Se vzrůstající množinou vzorů však stoupají nároky na paměť pro vyhledávací jednotku. Velké množiny vzorů je tedy nutno implementovat s pomocí velkých, avšak pomalých pamětí. Požadavek na velikost množiny vzorů je tak v přímém protikladu s požadavkem vysoké rychlosti.

Třetím podstatným požadavkem je, aby vyhledávací jednotka pracovala na úrovni síťových toků. To znamená, aby byla schopna detekovat vzory na rozhraní jednoho nebo více paketů. Tento požadavek reprezentuje velmi složitý problém, neboť pro jeho stoprocentní splnění by bylo potřeba seskládat z paketů celý TCP tok ještě před započetím vyhledávání, což je v prostředí páteřní sítě nemožné. Dalším možným řešením je ukládat si ke každému prohledávanému toku pomocnou informaci, která nahradí znalost předcházejících paketů. Problémem jsou však toky, kde pakety chodí ve špatném pořadí.

Čtvrtým zásadním požadavkem pro použití v moderních IDS je možnost rychlé změny vyhledávaných výrazů, aniž by se přerušil provoz zařízení. Důvody pro tento požadavek jsou v zásadě dva. Prvním je nutnost reagovat na nově vznikající hrozby, jako jsou například počítačové viry. Na nové hrozby je třeba reagovat dostatečně rychle. Worm jménem Witty [1] byl v rámci svého útoku schopen infikovat většinu potencionálních cílů v rámci 45 minut od počátku svého útoku. Druhým důvodem pro rychlou změnu vyhledávaných výrazů je samotná architektura moderních IDS, která umožňuje vyšším vrstvám přidávat či ubírat vyhledávané vzory.

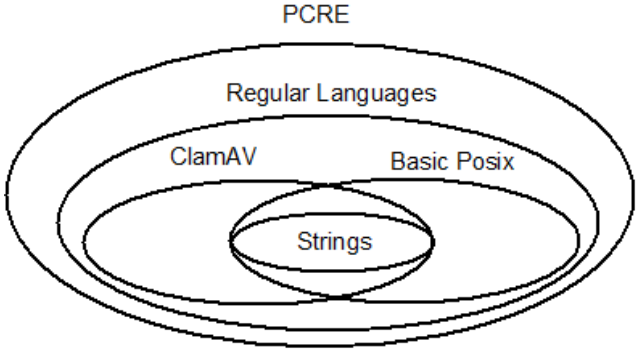
## 3 Předpokládané řešení problému

Ve své práci se zaměřím na návrh jednotky vhodné pro implementaci v HW. Hardwarová implementace vyhledávací jednotky umožní dosáhnout rychlostí srovnatelných s rychlostmi moderních páteřních sítí a zároveň implementovat vyšší vrstvy IDS na hostitelském počítači. Značnou výhodou hardwarové implementace je možnost přizpůsobení obvodové architektury vyhledávací jednotky tak, aby byla schopná zpracovat vstupní data každý hodinový takt, což by u procesorového řešení nebylo možné. Maximální frekvence vyhledávací jednotky založené na FPGA obvodech firmy Xilinx se bude pohybovat okolo 250MHz. Z této hodnoty je patrné, že vyhledávací jednotka musí pro zpracování vysokorychlostních toků přijmout více znaků v jednom hodinovém taktu. Tento problém

lze řešit transformací vstupní abecedy z ASCII řetězce o délce n na jediný symbol v abecedě vyhledávací jednotky. Transformace abecedy navíc přispěje k úspoře místa při implementaci vyhledávací jednotky [2].

Navrhovaná vyhledávací jednotka bude založena na konečném automatu. Ten umožňuje vyhledávat vzory se složitostí lineární k délce prohledávaného řetězce a je modelem pro rozpoznávání regulárních výrazů. Tedy i vzory, které lze s jeho pomocí vyhledat patří do množiny regulárních výrazů.

Prvním krokem vedoucím k návrhu struktury hardwarové vyhledávací jednotky však musí být důkladná analýza vzorů používaných v současných systémech, a to jak na teoretické úrovni, tak na úrovni praktické. Nejrozšířenějšími IDS systémy jsou Snort [3] a Bro [4]. Další vzory použitelné pro analýzu lze získat z aplikace NetFilter [5], která slouží pro rozpoznávání protokolů pomocí vyhledávání vzorů v síťové komunikaci. Každý systém má však jinou syntaxi zápisu vzorů. Snort používá Perl Compatible Regular Expression (PCRE) [6]. PCRE zápis umožňuje značné množství konstrukcí, což velmi komplikuje konstrukci vyhledávací jednotky pro takto zapsané vzory. Hlavním problémem však je skutečnost, že jednotlivé zápisy mají různou popisnou sílu.




*Obr. 1: Popisná síla jednotlivých typů zápisu vzorů: PCRE odpovídá programům Snort a L7, regular expression pak programu Bro. Basic Posix je využíván v L7 pro popis kernel space vzorů. ClamAV pak pro vzory Opensource antiviru stejného jména.*

Z obr. 1 je patrno, že PCRE je nadmnožinou regulárních výrazů. Tento rozdíl je způsoben operací nazývanou backreference, která odkazuje na již přijatý řetězec. Tedy PCRE výraz *(.*)\1* přijímá řetězce, které se opakují: .* přijme libovolný řetězec, avšak *\1* přijme pouze ten řetězec, který byl rozpoznán v rámci předcházející závorky. Tuto strukturu nelze vyjádřit pomocí konečného automatu. Takovýto vzor však může být pouze teoretický případ, který se v praxi nevyskytuje. Je tedy nutné zpracovat detailní analýzu používaných pravidel a na jejím základě rozhodnout, zda rozšířit podporu vyhledávací jednotky i na použití backreferencí a pokud ano, do jaké míry.

Analýza pravidel používaných pro detekci protokolů v L7 filtru odhalila překvapivou skutečnost. Přestože L7 filtr umožňuje použití pravidel popsaných rozšířeným standardem POSIX, který podporuje i backreference, tak tento typ zápisu se v oficiální sadě pravidel vůbec nepoužívá. Analýza byla tedy provedena pouze na pravidlech programu Snort. Sada pravidel vydaných 9. března 2009 obsahovala 1950 jedinečných PCRE vzorů. Z toho pouhých 163 vzorů obsahovalo backreference, což je méně než 10%. Ze 163 backreferencí je 77 použito pro označení řetězce, toto použití backreferencí lze snadno nahradit operací alternace: řetězec označený uvozovkami *nebo* řetězec označený apostrofem. Program Snort má databázi pravidel rozdělenou do souborů, podle různých typů útoků. Provedli jsme tedy analýzu i po jednotlivých souborech a získali jsme ještě zajímavější výsledky. Z 54 souborů se backreference vyskytly pouze v 5 z nich. (V počtu 6,95,60,1,1). Na základě této analýzy jsem se rozhodl omezit pouze na regulární výrazy.

Pro vyhledávání dostatečně velké množiny vzorů při zachování vysoké propustnosti je třeba se zaměřit na úsporu paměti. Jedním z možných řešení je rozdělení množiny vzorů do několika disjunktních množin a implementace každé takovéto množiny do samostatné, paralelně pracující

vyhledávací jednotky. Toto řešení je diskutováno v [7]. Diskutované řešení je však homogenní, tedy všechny paralelní vyhledávací jednotky jsou implementovány pomocí stejného algoritmu. Na základě experimentů prováděných nad dostupnými množinami vzorů však bylo zjištěno, že vlastnosti jednotlivých vzorů se velmi podstatně liší. Tyto vlastnosti mají velký vliv na efektivitu využití paměti ve vyhledávací jednotce. Příkladem může být existence cyklů v grafu konečného automatu, či zaplněnost přechodové tabulky, která se v pokusné množině pohybovala v rozsahu od 1 do 90 procent. V rámci své práce se tedy zaměřím na heterogenní systém, kde by paralelní jednotky byly implementovány pomocí odlišných algoritmů a každé jednotce by se přiřadila množina pravidel vhodných pro daný algoritmus.

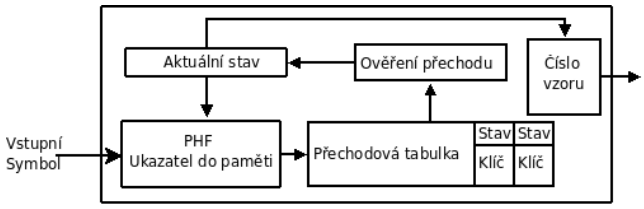
# 4 Dosažené výsledky

V rámci své dosavadní práce jsem se zaměřil především na minimalizaci paměťové složitosti vyhledávacího algoritmu založeného na použití deterministického konečného automatu.

Nejprve jsem provedl experimenty na dostupných sadách vzorů, převážně však na sadě vzorů programu Snort. Tyto experimenty prokázaly, že přibližně polovina pravidel generuje automat s méně než 30 procentní zaplněností přechodové tabulky. Takové automaty je vhodné implementovat s pomocí hash funkce (HF).

Při implementaci deterministického automatu založeného na HF v obvodu FPGA se HF implementuje pomocí rekonfigurovatelné logiky a výsledek HF se použije jako ukazatel do externí paměti, kde jsou uloženy následující stavy. Vzhledem k omezené propustnosti sběrnice k externím pamětem a nutnosti čtení dat při každém přechodu v automatu je žádoucí minimalizovat počet přístupů do paměti v rámci jednoho přechodu. Přechod slouží jako klíč HF a výstupem je adresa do paměti, kde se nachází cílový stav. V případě, že pro některé přechody dojde ke kolizi HF, je nutné provést několik dotazů do paměti pro určení správného cílového stavu. Pro efektivní funkci vyhledávací jednotky je tedy nutné minimalizovat počet kolizí v HF.

K minimalizaci počtu kolizí HF je možné použít perfektní hash funkci (PHF). PHF je taková HF, která nemá pro předem stanovenou množinu klíčů žádnou kolizi. Vyhledání takové HF není pro obecnou množinu klíčů triviální a pro uchování nalezené PHF je nutné použít další paměť. Základní přehled o PHF je možné získat z [11]. Na základě získaných informací jsem identifikoval dva algoritmy vhodné pro implementaci konečného automatu [10][9]. Oba algoritmy mají lineární paměťovou složitost s počtem klíčů, avšak algoritmus [10] má v nejhorším možném případě exponenciální dobu konstrukce PHF, což by mohlo negativně ovlivnit rychlost změny vyhledávaných pravidel. Z tohoto důvodu jsem se rozhodl založit svou metodu na algoritmu představeném v [9].




*Obr. 2: Princip navrhovaného řešení*

Princip navrhované metody je patrný z obr. 2. Aktuální stav a vstupní symbol automatu vstupují do PHF, která spočítá pozici v externí paměti. Z té se vyčte hodnota následujícího stavu spolu s hodnotou klíče PHF, která slouží k ověření, zda daný přechod v automatu opravdu existuje. Tento postup se opakuje, dokud se nezpracují všechny znaky vstupního řetězce nebo dokud se na vstupu neobjeví neexistující přechod.

Metodu jsem srovnal s dalšími existujícími metodami. Výsledky jsem přehledně shrnul do tabulky 1.

| Experiment | Navržená metoda | Metoda z [16] | Metoda z [18] | Zaplněnost tabulky |
|---|---|---|---|---|
| 1 | 615 kb | 1666 kb | 5805 kb | 1.50% |
| 2 | 115 kb | 145 kb | 317 kb | 3.90% |
| 3 | 342 kb | 29 kb | 73 kb | 64.90% |
| 4 | 641 kb | 368 kb | 1453 kb | 2.60% |
| 5 | 530 kb | 4325 kb | 9140 kb | 0.50% |

Vstupem experimentů byla sada pravidel *virus_ruleset* IDS Snort. Vzory v ní obsažené byly rozděleny do 10 skupin algoritmem dle [7]. Pro každou skupinu byl zkonstruován jeden Regulární výraz, který byl převeden na automat. Výsledný automat byl rozšířen tak, aby bylo možno přijímat 4 znaky v jednom kroku (tady rychlost vyhledávání byla 10Gb/s). Tabulka 1 ukazuje spotřebu paměti potřebné na reprezentaci automatu navrhovanou metodou a dvěma zvolenými metodami jiných autorů.

Největší množství paměti je v metodě použito k uložení hodnoty klíče PHF. Proto bych se rád v budoucnosti zaměřil právě na minimalizaci této paměti. V rámci [12] je popsáno několik obecných postupů použitelných k redukci paměťové složitosti. Jako slibná cesta se však také jeví dekompozice problému. PHF by byla použita pouze pro určení následujícího stavu. Rozhodnutí o existenci přechodu by se provádělo nezávisle s rámci samostatného algoritmu. Tato jednotka by pak realizovala pouze dotaz na příslušnost klíče do množiny všech přechodů automatu. Problematika dotazování na příslušnost do množiny je řešena v [13].

Navrženou metodu jsem publikoval na Junior Scientific Conference ve Vídni jako poster a v současné době je přijata k publikaci na konferenci DSD 2009 jako short paper.

## 5 Cíle disertační práce

V rámci své práce bych se chtěl zaměřit na analýzu vzorů používaných v současných IDS systémech, které jsou podmnožinou všech možných vzorů a identifikovat takové vlastnosti těchto pravidel, které by bylo možno využít pro návrh efektivního algoritmu pro vyhledávaní regulárních výrazů v síťovém provozu na vysokorychlostních páteřních sítích.

Hlavním cílem mé disertační práce je navrhnout metodu vhodnou pro vysokorychlostní vyhledávání vzorů použitelnou v moderních systémech ochrany počítačových sítí. V práci bych se rád soustředil na využití perfektních hash funkcí pro implementaci deterministických konečných automatů, které v kombinaci s vhodnou metodou detekce přítomnosti přechodu v automatu může dosáhnout velmi vysoké efektivnosti při využití paměti.

## 6 Závěr

V rámci mého prvního ročníku studia jsem se seznámil se s existujícími metodami v oblasti vyhledávání vzorů a perfektních hash funkcí. Na základě těchto znalostí jsem identifikoval otevřené problémy při vyhledávání regulárních výrazů a navrhl jsem metodu, která je použitelná pro implementaci vysokorychlostní vyhledávací jednotky pracující na úrovni síťových toků. Tato metoda byla publikována jako poster na Junior Scientific Conference ve Vídni [8] a v době psaní tohoto článku je také přijata k publikaci jako short paper na konferenci DSD 2009. Na základě teoretického srovnání navržené metody s metodami běžně používanými byly stanoveny další směry vývoje algoritmu.

Budoucí kroky práce lze rozdělit do několika směrů výzkumu. Zejména se pokusím snížit paměťovou složitost metody a současně se zaměřím se na paralelizaci vyhledávání do více jednotek, z nich každá bude pracovat na odlišném principu. Tím bude umožněno efektivní využívání dostupných

prostředků. Každá metoda je vhodná pro jistou podmnožinu vyhledávaných vzorů, avšak její vlastnosti se zhoršují pro jiné používané vzory. Pro použití více jednotek bude nutné nalézt algoritmus, který bude schopen automaticky rozdělit vzory do množin vhodných pro dané jednotky.

## 7 Poděkování

Tato práce vznikla díky podpoře Výzkumného záměru Fakulty informačních technologií: Výzkum informačních technologií z hlediska bezpečnosti, MSM0021630528.

## 8 Literatura

[1] C. Shannon and D. Moore, “The Spread of the Witty Worm ,” IEEE SECURITY and PRIVACY, vol. 2, no. 4, pp. 46–50, 2004

[2] M. Becchi and P. Crowley, “An Improved Algorithm to Accelerate Regular Expression Evaluation,” in ANCS ’07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems. New York, NY, USA: ACM, 2007, pp. 145–154.

[3] “Snort homepage.” [Online]. Available: www.snort.org

[4] “Bro homepage.” [Online]. Available: www.bro-ids.org

[5] “L7 homepage.” [Online]. Available: l7-filter.sourceforge.net

[6] “PCRE homepage.” [Online]. Available: www.pcre.org

[7] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, “Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection,” in ANCS ’06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems. New York, NY, USA: ACM, 2006, pp. 93–102.

[8] Kaštil Jan, Kořenek Jan: Deterministický konečný automat s perfektním hashováním pro rychlé vyhledávání vzorů, In: Proceedings of Junior Scientist Conference 2008, Vienna, TU-Wien, 2008, p. 103-104, ISBN 978-3-200-01612-5

[9] F. C. Botelho, R. Pagh, and N. Ziviani, “Simple and Space-efficient Minimal Perfect Hash Functions,” in In Proc. of the 10th Intl. Workshop on Data Structures and Algorithms. Springer LNCS, 2007, pp. 139–150.

[10] E. A. Fox, Q. F. Chen, and L. S. Heath, “A Faster Algorithm for Constructing Minimal Perfect Hash Functions,” in SIGIR ’92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY, USA: ACM, 1992, pp. 266–273

[11] Z.J. Czech, G. Havas, and B.S. Majewski. Fundamental study perfect hashing. Theoretical Computer Science, 182:1-143, 1997

[12] S. Lefebvre and H. Hoppe, “Perfect Spatial Hashing,” in SIGGRAPH 06: ACM SIGGRAPH 2006 Papers. New York, NY, USA: ACM, 2006, pp. 579–588.

[13] P. Briggs and L. Torczon, “An Efficient Representation for Sparse Sets,” ACM Lett. Program. Lang. Syst., vol. 2, no. 1-4, pp. 59–69, 1993.

[14] R. Sommer and V. Paxson, “Enhancing Byte-level Network Intrusion Detection Signatures with Context,” in CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security. New York, NY, USA: ACM, 2003, pp. 262–271

[15] Miroslav Balík, „Searching by Substring“, PhD. Thesis, Czech technical univeristy

[16] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, “Deterministic Memory-efficient String Matching Algorithms for Intrusion Detection,” in In IEEE Infocom, Hong Kong, 2004, pp. 333–340.

[17] Sailesh Kumar, Jonathan Turner, Patrick Crowley, Michael Mitzenmacher, "HEXA: Compact Data Structures for Faster Packet Processing," *Proceedings of IEEE ICNP'07*, Beijing, China, October, 2007

[18] G. Navarro and M. Raffinot, “New Techniques for Regular Expression Searching,” Algorithmica, vol. 41, no. 2, pp. 89–116, Nov. 2004