HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Poboljšanje performansi NIDS-a za velike brzine

CCERT-PUBDOC-2002-04-02

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Povećanje količine mrežnog prometa, isto kao i svakotjedno povećanje kritičnih sigurnosnih nedostatka u aplikacijskom sloju znači da mrežni sustavi sa otkrivanje neovlaštenih aktivnosti korisnika (*Network Intrusion Detection Systems – NIDS*) moraju zadovoljiti potrebu za ubrzanjem tehnika analiza napada prilikom nadgledanja potpuno zasićenih mreža, a istovremeno zadržati dobar omjer lažnih pozitivnih i lažnih negativnih detekcija.
Iako povećanje procesorske snage, odnosno brzine CPU i povećanje radne memorije NIDS sustava pomaže poboljšanju performansi, postoji točka gdje količina potrošenog novca na nove sklopovske komponente više nije proporcionalna povećanju brzine analize. Općenito, postoji samo par stvari koje korisnik može učiniti u svrhu poboljšanja performansi NIDS sustava, kao što je ograničenje broja napada koje NIDS prati korištenjem preklopnika u sedmom sloju, sve ostalo ovisi o samim dizajnerima NIDS sustava.

2. Uska grla

U snortu postoje četiri glavna područja koja uzimaju priličnu količinu vremena, ali samo jedno od njih daje razumno povećanje performansi, uz istovremeno zadržavanje portabilnosti.

1. Skidanje paketa sa mreže. Snort koristi libpcap za skidanje paketa sa mreže. Libpcap je gotovo standard za skidanje paketa i koriste ga mnoge aplikacije za dekodiranje protokola. Libpcap biblioteka je dobra za većinu aplikacija, ali za potrebe brzog dohvata podataka i nije tako pogodna, jer se u pojedinom vremenskom trenutku može koristiti samo jedna libpcap funkcija. Bilo je moguće razviti posebne upravljačke programe za snort, koji bi bili OS-specifični i možda NIC-specifični, ali to bi uvelike utjecalo na njegovu portabilnost.
2. Pražnjenje struktura podataka. Svaki paket koji dođe mora se pohraniti u jedan od tipova podataka. To dalje znači da svaka od tih struktura kasnije mora biti ispražnjena, da bi se omogućilo prihvaćanje novih paketa. Ovaj postupak je obično moguće optimirati razmatranjem pojedinih pretpostavki i boljom organizacijom programskog koda.
3. Prepoznavanje uzoraka. U snortu 2.0 za prepoznavanje uzoraka koristit će se drugačiji algoritmi prilagođeni za zasićene mreže velikih brzina. Njihovom implementacijom snort 2.0 bi mogao dobiti i do 500% poboljšane performanse.
4. Provjera kontrolnih suma. U cilju provjere integriteta paketa snort provjerava sve kontrolne sume. To znači da za svaki paket snort mora izračunati kontrolnu sumu, te provjeriti da li odgovara kontrolnoj sumi paketa.

Performanse također mogu biti naručene raznim predprocesorima i parametrima koji se učitavaju zajedno s njima. Parametri koji se specificiraju predprocesoru mogu također utjecati na omjer lažnih pozitivnih detekcija i na efikasnost NIDS-a. Treba pogledati kako predprocesori mogu utjecati na peformanse.
Svaki predprocesor ima različitu funkciju (npr. `frag2` – reasembliranje IP fragmentacije, `stream4` – provjera TCP reasembliranja, `http_decode` – normalizacija HTTP zahtjeva itd.). Performanse mogu biti popravljene ili pogoršane ovisnoo sljedećim parametrima:

1. Vrsta prometa koju NIDS nadgleda. Ovo također podrazumijeva i količinu prometa za pojedini protokol.
2. Parametri koji su specificirani predprocesoru. Parametri određuju podatke kao što su pojedini portovi, koliko memorije treba koristiti, koliko dugo valja čuvati informacije itd.

Ispravna konfiguracija predprocesora zahtjeva određeno vrijeme za pravilno podešavanje. To podrazumijeva sljedeće:

1. Dobro razumijevanje mreže koja se nadgleda, njezine protokole, aplikacije i uzorke prometa.
2. Dobro razumijevanje mrežnih protokola. Rukovatelj NIDS-om mora biti sposoban analizirati promet da bi odredio granične vrijednosti, lažne pozitivne detekcije itd.
3. Poznavanje predprocesora i načina na koji rade, te kako se konfiguriraju. Neke od predprocesora valja samo definirati (npr. `frag2`, `telnet_decode`), neki pak zahtijevaju par argumenata za koje nije nužna detaljna analiza mreže (npr. `rpc_decode`, `http_decode`, `unidecode`), dok postoje i predprocesori čije podešavanje zahtijeva

mnogo istraživanja i finog podešavanja (npr. `stream4`, `stream4_reassemble`, `spade`).

3. Kratka povijest snorta i prepoznavanja uzoraka

U ranijim danima, snort je koristio doslovno prepoznavanje uzoraka koje je bilo vrlo sporo i predstavljalo je kariku koju je trebalo poboljšati. Prvo poboljšanje performansi postignuto je implementacijom Boyer-Mooreovog algoritma. Sljedeće poboljšanje donijelo je uvođenje dvodimenzionalne povezane liste sa rekurzivnim prolascima, što je značilo poboljšanje performansi od 200% do 500%. Danas snort koristi detekciju koja se služi povezanom listom funkcijskih pokazivača, koja se također naziva trodimenzionalna povezana lista.

4. Pregled snort pravila

Snort pravila podijeljena su u dva dijela; pravilo zaglavlja i opcije pravila. Pravilo zaglavlja predstavlja sve što se nalazi prije prve zagrade. Opcije pravila nalaze se unutar zagrada. Pravilo zaglavlja može biti mapirano prema RTN-u (*Rule Tree Node*), dok opcije pravila mogu biti mapirane prema OTN-u (*Optional Tree Node*).

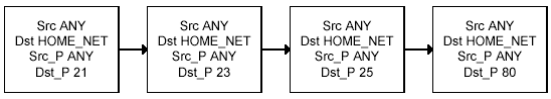
```
alert udp $EXTERNAL_NET any -> $HOME_NET 177 (msg:"MISC xdmcp query",
content: "|00 01 00 03 00 01 00|";reference:arachnids,476;
classtype:attempted-recon; sid:517; rev:1;)
```

Slika 1: *Snort pravila*

Postoji 35 ključnih rječu u snortu 1.8.3. koje se mogu koristiti kao opcije pravila, od njihe se 20 može koristiti u OTN-ovima. Od tih 20 ključnih riječi, 17 može imati vrijednost istinito/neistinito (jednako/nije jednako) ili veće od/manje. Snort može jednostavno proslijediti ove informacije kroz povezanu listu bez većih zadrški. Najveće zadrške u računanju pojavljuju se prilikom korištenje sljedeće tri ključne riječi: `content`, `content-list` i `uricontent`. Svaka od tih ključnih riječi podrazumijeva pregledavanje dijela podataka u cilju pronalaženja pojedinog uzorka. Zbog zadrške koju unosi prepoznavanje uzorka, ono je posljednji dio opcija pravila koji se provjerava. Od 1270 pravila, 1086 pravila sadrži ili "`content`" ili "`uricontent`" ključnu riječ.

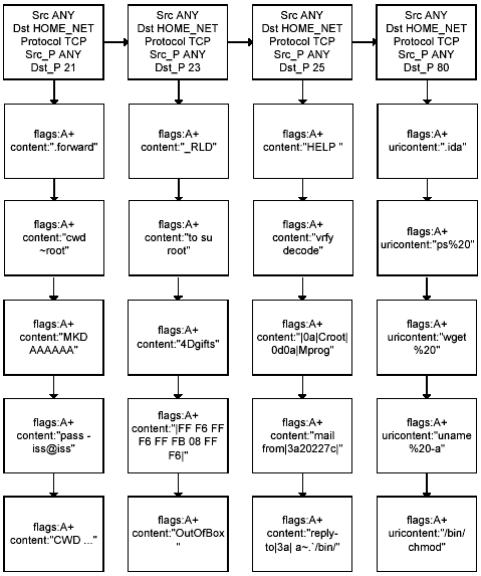
5. Učitavanje pravila i detekcija

Kada snort inicijalizira i provjeri pravila, on stvara zasebna pravila za TCP, UDP, ICMP i IP protokol. Unutar svakog stabla kasnije će se kreirati zasebne trodimenzionalna povezana lista sa RTN-ovima (prva dimenzija), OTN-ovima (druga dimenzija) i pokazivačima funkcija (treća dimenzija). RTN-ovi također uključuju i informacije o IP adresama i portovima.



Slika 2: *Primjer lanca zaglavlja (RTN)*

Kada snort pošalje paket kroz dio za detekciji prvo provjerava koji je protokol paketa, tako da ga može proslijediti na pravo stablo pravila. Kada je paket poslan na odgovarajuće stablo na evaluaciju, biti će provjeren u odonosu na svaki RTN, s lijeva na desno, dok se ne pronađe poklapanje. Prilikom provjere RTN-ova, snort prvo provjerava IP adresu, a zatim i port, ukoliko je potrebno. Ukoliko je pronađen RTN koji odgovara promatranom paketu, paket se provjerava u odnosu na OTN-ove, te se ponovno traži poklapanje. Svi OTN-ovi se ne provjeravaju za sve opcije, jer bi to predstavljalo preveliko trošenje resursa (npr. povjeravanje `content` sadržaja na `non-content` paketu). Umjesto toga svaki OTN ima povezanu listu pokazivača funkcija (treća dimenzija) za testove koje treba provesti u odnosu na svaki pojedini OTN.



Slika 3: *Primjer lanca zaglavlja (RTN) i lanca opcija (OTN)*

Snort koristi Boyer-Mooreov algoritam za provjeravanje uzoraka prilikom pregledavanja sadržaja paketa. Taj algoritam je jedan od najefikasnijih algoritama za provjeravanje stringova i često se koristi sa pretraživanje ili zamjenu dijelova teksta u editorima teksta. Boyer-Mooreov algoritam je dobar za pojedinačno pretraživanje stringova, ali prilikom rada NIDS-a, pojedini paket može djelomično odgovarati više različitih pravila, a za svako od njih algoritam se mora pokrenuti iz početka. Npr. paket se uspoređuje sa uzorkom `/cfdocs/cfmlsyntaxcheck.cfm`, a također i 15 idućih OTN-ova sadrži `/cfdocs/` direktorij na početku uzorka (*Slika 4*). Nakon pretraživanja paketa uočeno je da `/cfdocs/` ne postoji nigdje u paketu, ali svejedno će se provesti svih ostalih 15 provjera.
Potrebno je razviti novi algoritam za pronalaženje uzoraka da bi se ovaj nedostatak eliminirao. Testirani su neki algoritmi koji koriste dobre aspekte Boyer-Mooreovog algoritma i Aho-Corassick algoritam za poboljšanje potrebnih performansi.

```
/cfdocs/cfmlsyntaxcheck.cfm
/cfdocs/exampleapp/
/cfdocs/exampleapp/email/application.cfm
/cfdocs/exampleapp/email/getfile.cfm
/cfdocs/exampleapp/publish/admin/addcontent.cfm
/cfdocs/exampleapp/publish/admin/application.cfm
/cfdocs/examples/cvbeans/beaninfo.cfm
/cfdocs/examples/mainframeset.cfm
/cfdocs/examples/parks/detail.cfm
/cfdocs/expeval/
/cfdocs/expeval/displayopenedfile.cfm
/cfdocs/expeval/exprcalc.cfm
/cfdocs/snippets/
/cfdocs/snippets/evaluate.cfm
/cfdocs/snippets/fileexists.cfm
/cfdocs/snippets/gettempdirectory.cfm
```

Slika 4: *Primjer pravila snorta*

6. Boyer-Mooreov Algoritam

Prije opisa samog algoritma potrebno je utvrditi terminologiju:

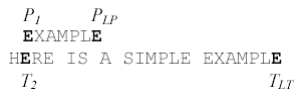
1. prepoznavanje uzoraka označavat će se sa *P*,
2. tekst koji će se provjeravati biti će označen sa *T*,
3. duljina uzorka (*P*) biti će označena sa *LP*,
4. duljina teksta (*T*) biti će označena sa *LT*,
5. prvi i posljednji znak uzorka *P* biti će redom označeni sa P_1 i P_{LP},
6. prvi i posljednji znak teksta *T* biti će redom označeni sa T_1 i T_{LP},
7. prilikom inicijalne usporedbe uzorka *P* i teksta *T*, zadnji znak u *P*, P_{LP}, biti će uspoređen sa T_{LP}.

Uzorak za pretraživanje: EXAMPLE
Tekst koji se pretražuje: HERE IS A SIMPLE EXAMPLE
Klasični algoritam pretraživanja bi tražio uzorak *P* u tekstu *T* na sljedeći način:

1. Poravnati lijeve krajeve *P* i *T*, tako da su P_1 i T_1 poravnati (*Slika 5*).

Slika 5

2. Usporediti znakove *P* u odnosu na *T* s lijeva na desno dok nije pronađeno neslaganje ili su svi znakovi u *P* uspoređeni. U tom slulaju P_1=E i T_1=H i E≠H (*Slika 5*).
3. Ukoliko se dogodi neslaganje algoritam pomiče *P* za jedan znak u desno, a postupak usporedbe se ponavlja (*Slika 6*). Ovaj puta P_1=E biti će poravnat sa T_2=E. Pošto je E=E, provjerit će se P_2=X u odnosu na T_3=R, pošto je X≠R, *P* će se ponovno pomaknuti za jedan znak u desno i usporedba će biti ponovno pokrenuta.



Slika 6

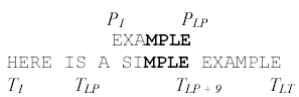
4. Proces će se nastaviti dok se ne pronađe kompletno poklapanje *P* unutar *T* ili dok posljednji P_{LP} ne pređe lijevo od T_{LT}.

U gornjem slučaju, klasičnom algoritmu bilo bi potrebno 28 pokušaja da pronađe poklapanje. Lako se može uočiti da ovakav način pretraživanja uzima mnogo vremena da bi pronašao poklapanje ili odredio da se poklapanje ne može pronaći. Prve inačice snorta radile su na sličan način.
Boyer-Mooreov algoritam ima tri poboljšanja koja unapređuju klasični algoritam i čine ga efikasnijim čak i u najlošijem mogućem slučaju.

1. Pretraživanje s desna na lijevo. To je suprotno od klasičnog algoritma koji tekst pretražuje s lijeva na desno. Lijevi kraj *P* je još uvijek poravnat sa lijevim krajm *T*, ali pretraživanje počinje sa desnog kraja *P* i pomiče se ulijevo dok ne dođe do neslaganja (*Slika 7*).

Slika 7

2. Pomicanje pogrešnog znaka. Prvi pokušaj poklapanja biti će na P_{LP}=E i T_{LP}=S (*Slika 7*). Kao i mnogi napredni algoritmi za pronalaženje uzoraka Boyer-Mooreov algoritam prethodno obrađuje uzorak i na temelju toga dolazi do heurističkih informacija. Na temelju tih informacija izračunat će koliko treba *P* pomaknuti u desno. Algoritam treba odrediti najdesniji znak u *P* koji se može poklopiti sa *T*. U ovom slučaju to bi bio znak E (*Slika 8*).



Slika 8

3. Dobro pomicanje sufiksa. Jednom kada se dogodi poklapanje, algoritam će pokrenuti pregledavanja na desnom kraju *P*. Ovog puta P_{LP}=E odgovarat će T_{LP+9}=E (*Slika 8*). Nakon toga uspoređuje se P_{LP-1}=L sa T_{LP+8}=L. Ponovno je postignuto poklapanje, tako da se P_{LP-2}=P uspoređuje sa T_{LP+7}=P. Pošto se tekst i dalje poklapa sa uzorkom, uspoređuje se P_{LP-3}=M sa T_{LP+6}=M. Pošto je poklapanje opet postignuto, uspoređuje se P_{LP-4}=A sa T_{LP+5}=I. Ovaj puta dolazi do neslaganja. Pošto je u predobradi uzorka pronađeno da *T* sadrži string "MPLE",

algoritam će provjeriti sljedeće ponavljanje tog uzorka u *T*. Zatim će pomaći *P* u desno tako da string "MPLE" od *P* bude poravnat sa sljedećom pojavom stringa "MPLE" unutar *T* (*Slika 9*).

Slika 9

4. Kada je pomicanje završeno, usporedba znakova će ponovno početi s desne strane *P*. U ovom slučaju svaki znak unutar *P* odgovara *T* počevši od P_{LP}=E i T_{LT}=E i potpuno poklapanje je pronađeno.

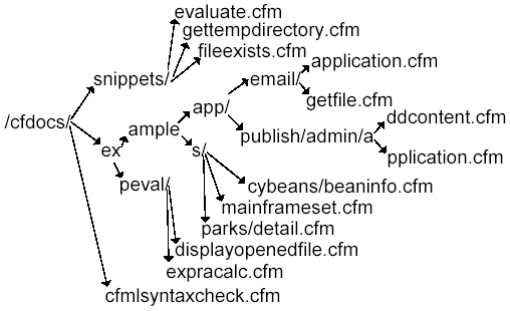
Sa Boyer-Mooreovim algoritmom gornje pretraživanje traži samo 12 pokušaja prije nego što se pronađe poklapanje. Ovo je više nego dvostruko brže od klasičnog algoritma.

7. Aho-Corasick_Boyer-Mooreov kombinirani algoritam

Iako samo ime implicira da je novi algoritam mješavina Aho-Corasick i Boyer-Mooreovog algoritma, to nije tako. To je ustvari algoritam sličan Booyer-Mooreovom primijenjen na skup ključnih riječi sadržanih u stablu ključnih riječi koje podsjeća na Aho-Corassickovo. Na taj način su izolirane najbolje značajke oba algoritma.

1. Sličnosti:
   a. Boyer-Moore – pomicanje pogrešnog znaka,
   b. Aho-Corasick – stablo ključnih riječi.
2. Varijacije:
   a. Boyer-Moore – umjesto korištenja originalnog pomicanja dobrog sufiksa novi algoritam koristi pomicanje dobrog prefiksa,
   b. Boyer-Moore – dok se paket (tekst, *T*) pretražuje sa desna na lijevo, stablo (uzorak, *P*) se pretražuje se lijeva na desno,
   c. Aho-Corasick – umjesto građenja stabla temeljenog na sufiksima, stablo se gradi na temelju prefiksa.

Vrativši se malo na originalni problem (*Slika 4*), može se uočiti šesnaest različitih pravila koja dijele neke zajedničke informacije. Ukoliko bi se tih 16 pravila organiziralo u stablo ključnih riječi (kao u Aho-Corasick_Boyer-Mooreovom algoritmu) to bi izgledalo drugačije (*Slika 10*).



Slika 10: *Prikaz ključnih riječi organiziranih u stablo*

Na prvi pogled ovo stablo može izgledati još složenije nego na *Slika 4*, ali ukoliko se bolje prouči, lako je uočiti njegovu efikasnost. Neslaganje odmah može eliminirati mnoga pravila, te na taj način isključiti suvišna i nepotrebna pretraživanja. Sa običnim Aho-Corasickovim algoritmom stablo ključnih riječi bilo bi pretraživano znak po znak, kao sa klasičnim algoritmom pretraživanja. Dodatkom Boyer-Mooreovog pomicanja prefiksa i pomicanja neodgovarajućih znakova, algoritam može brzo odrediti da li se poklapanje događa ili paket ne odgovara niti jednom od uzoraka.

Aho-Corasick_Boyer-Mooreov pokazuje lagano poboljšanje performansi kada se koristi sa pravilima koja ne ovise o sadržaju, no pravo poboljšanje uočljivo je prilikom korištenja pravilima koja ovise o

sadržaju. U trenutnoj implementaciji snorta, broj sadržajnih pravila će značajno utjecati na performanse; zbog tog razloga se i razmatra uporaba novih algoritama za pronalaženje uzoraka.

8. Tekuća pitanja

Iako novi algoritam ima bolje performanse za sadržajna pravila, on također ima neke nedostatke koje treba poboljšati, ili barem uzeti u obzir prilikom implementacije.
Prvo, Aho-Corasick_Boyer-Mooreov algoritam započinje pregledavanje na desnom kraju paketa i pomiče se prema lijevom kraju, zbog toga može aktivirati različito pravilo nego originalni Boyer-Mooreov algoritam prilikom analize istog paketa. Npr., ukoliko postoje dva pravila koja se razlikuju samo po kontekstu unutar kojeg se pretražuje:

Sadržaj 1. pravila: `Firstpartofthepacket`
Sadržaj 2. pravila: `Lastofthepacket`
Paket (tekst): `Firstpartofthepacketandthenlastofthepacket`

Kada paket bude pretražen Boyer-Mooreovim algoritmom, on će aktivirati 1. pravilo. Ukoliko se paket pretražuje na temelju Aho-Corasick_Boyer-Mooreovog algoritma, on će aktivirati 2. pravilo. To je posljedica različitih pristupa pretraživanju paketa. Ukoliko se Aho-Corasick_Boyer-Mooreov algoritam podesi da provjerava podatke sa lijeva na desno (kao Boyer-Mooreov), ova anomalija bi se eliminirala. Jedini razlog zbog koje to nije učinjeno u inicijalnoj implementaciji jest da je to bio dokaz koncepta implementacije.
Snort predstavlja arhitekturu u koja favorizira redoslijed pravila (*engl. first rule match wins*). Problem koji se javlja sa Aho-Corasick_Boyer-Mooreovim algoritmom je taj da je prvo poklapanje najkraće poklapanje, što ne mora značiti da je to i ispravno poklapanje. Kad Aho-Corasick_Boyer-Mooreov algoritam sagradi stablo ključnih riječi, on gubi redoslijed pravila. Redoslijed pravila omogućuje trenutnoj implementaciji snorta da pronađe najdulje poklapanje.
Ovo pravilo najduljeg poklapanja preferira se u aplikacijama kao što su npr IP routing (EIGRP, OSPF, BGP-4) i regularni izrazi. Uporaba ovog pravila osigurava da će uvijek biti aktivirano najspecifičnije pravilo.
Jedina stvar koja je zadržana u Aho-Corasick_Boyer-Mooreovom algoritmu je sadržaj koji se traži. Bilo je potrebno pronaći način da bi se zadržale razne opcije koje ne ovise o sadržaju. Pošto je Aho-Corasick_Boyer-Mooreov algoritam trebao biti samo dokaz jednog koncepta, bilo je potrebno ili promijeniti predobradu pravila ili promijeniti nači na koji snort organizira RTN-ove i OTN-ove. Da bi se zadržala arhitektura koja bi što više nalikovala originalnoj, odabran je prvi pristup. Pravila su podijeljena na ona koja ovise o sadržaju i ona koja ne ovise o sadržaju, dok se opcije provjeravanju posebno, ovisno o tipu pravila. Prilikom separacije pravila, također je promijenjen način rukovanja pravilima koja ovise o sadržaju. Originalno, sve opcije se provjeravaju prije nego što se poziva Boyer-Mooreov algoritam da bi provjerio sadržaj. Prlikom uporabe Aho-Corasick_Boyer-Mooreovog algoritma sve ostale opcije se provjeravaju nakon što je algoritam pozvan da bi provjerio sadržaj pravila.

9. Analiza protokola

Prilikom spomena NIDS-a, većina ljudi pomisli na prepoznavanje uzoraka. No postoji još jedan način implementacije NIDS-a, a to je analiza protokola. Proizvodi koji se temelje na analizi protokola, postoje već dulje vrijeme i imaju mogućnost analize podataka u stvarnom vremenu, tako da mogu brzo odrediti koji se problemi dešavaju na mreži.
Arhitektura NIDS-a temeljenih na analizi protokola uvelike se razlikuje od one koja se temelji na prepoznavanu uzoraka. NIDS temeljen na analizi protokola će dekodirati svaki paket prema specifikaciji protokola, te će zatim provjeriti polja da provjeri da li onda odgovaraju standardu. Ukoliko paket ne odgovara standardnu, NIDS ga označava. Za pakete koji odgovaraju standardu, ali i dalje jesu napadi (npr. showcode.asp, bin/sh), NIDS može napraviti prepoznavanje uzorka u određenim poljima koje valja provjeriti, umjesto provjeravanja cijelog paketa. BlackICE npr. ima drugačiji pristup problemima s prepoznavanjem uzoraka koji bi mogli stvarati uska grla. U slučaju HTTP paketa korisnik može specificirati tekstualni uzorak u URI-ju. Npr. u paketu na *Slika 11* BlackICE

bi podijelio URI u različite komponente, u ovom slučaju "SAMPLE" i "showcode.asp". Nakon toga bi tražio kompletno poklapanje za svaku komponentu u svojoj bazi napada.
Snort je postao protokolno svjestan kada je dodana ključna riječ "uricontent". Time je korisnicima dana mogućnost pretraživanja samo URI dijela HTTP paketa umjesto pretraživanja cijelog paketa.

HTTP - Hyper Text Transfer Protocol
Command: GET
URI: /SAMPLE/showcode.asp
Version: HTTP/1.1..
Accept: */*..
Referer: http://www.victim.com/..
Accept-Language: en-us..
Accept-Encoding: gzip, deflate..
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)..
Host: www.victim.com..
Connection: Keep-Alive..
Cookie: RoxenUserID=0x673b30....

Slika 11: *Prikaz sadržaja HTTP paketa*

Poboljšanje performansi korištenjem ove metodologije ovisit će o mnogim faktorima; veličini paketa, veličini odgovarajućih polja, komponenti koje valja provjeriti itd.
Jedan od velikih nedostataka analize protokola jest ta da svaki proizvođač implementira protokol prema svojoj interpretaciji RFC-a. Na taj način moguće je dobivanje lažnih pozitivnih rezultata, ukoliko se paketi ne vrednuju na isti način od uređaja koji stvaraju promet i NIDS sustava.
Drugi nedostatak je mogućnost pronalaska načina rada NIDS sustava temeljenog na analizi protokola. Npr. pojavom napada Rain Forest Puppy zajedno sa Whiskerom, korisnici koji su koristili snort trebali su samo osvježiti bazu potpisa, dok su NetworkICE i BlackICE morali biti detaljno promijenjeni da bi se mogli nositi sa time.
Treći nedostatak analize protokola je način na koji NIDS tretira pakete koje ne može dekodirati. Jedna je mogućnost slanja upozorenja korisnika o detekciji paketa koji nije moguće dekodirati, a druga je pak mogućnost korištenja heuristike na paketu i provjeravanja da li je to poznati protokol koji se nalazi na nekom drugom portu.
Dobra strana analize protokola jest lakše pronalaženje novih napada, pošto se ne oslanjaju na prepoznavanje poznatog uzorka. Tehnike polimorfnih napada, ULR enkodiranja itd. također se lako detektiraju NIDS sustavima temeljenim na analizi protokola.
Ovisno o mrežnoj arhitekturi, pravilima i stupnju zabrinutosti moguće je pokrenuti NIDS unutar pojedine mreže. To je moguće napraviti na jednostavan način postavljanjem jednog NIDS-a koji nadgleda glavni dio mreže, ili postavljanjem složenog, raspodijeljenog NIDS okruženja, gdje je svaki NIDS konfiguriran da nadgleda promet specifičan pojedinom segmentu. Postavljanjem NIDS-a na LAN uočavaju se mnoge situacije gdje je NIDS temeljen na analizi protokola odgovarajući alat:

1. Većina velikih kompanija koriste snort ili neki interni routing protokol. Većina tih protokola osvježava se putem multicasta (OSPF, EIGRP) ili broadcasta (RIP, IGRP). U zadnjih nekoliko godina napisani su alati (irpas, nemesis, nmap) koji mogu izvlačiti informacija iz tih protokola, ili ih čak lažirati. Korištenjem tih alata, napadač može ozbiljno narušiti rad mreže ubacivanjem krivih ruta ili loših paketa. Još gora je mogućnost ukoliko napadač uspije lažirati gateway i hvatati pakete, te ih prosljeđivati pravom gatewayu. Mnogi se ne brinu o sigurnosti i vide je samo kao problem. Zbog tog stava i lažnog osjećaja sigurnosti zbog postojanja vatrozida, napadi na routere i rute mogu biti lako izvedeni i teško detektirani. NIDS može biti u mogućnosti detektirati routing protokole, ali ne zna koje vrste protokola bi trebale biti pokrenute i kako bi trebale biti konfigurirane. Npr. ukoliko je EIGRP jedini protokol koji se koristi
2. Mnogi lažiranje paketa u drugom sloju ne smatraju ozbiljnom prijetnjom. Neki žive u uvjerenju da su 100% prospojena okruženja sigurna od sniffera i drugih programa za hvatanje paketa. Postoje alati koji mogu zavarati switcheve da pakete koje ne bi trebali šalju napadačima. Pošto ti alati rade samo na LAN-u, mnogi ni ne pokušavaju detektirati te napade. Iako analizom protokola ne bi bilo moguće detektirani nikakve sumnjivo konstruirane pakete, u takve NIDS sustave moguće je ugraditi nekakvu funkciju za detekciju anomalija koja bi olakšavala njihovu detekciju.

10. Situacija danas i pogled u budućnost

Istraživanje koje je provelo Silicon Defense je slijedilo još jedno istraživanje koje su proveli Mike Fisk i George Varghese. Oni su implementirali algoritam koji se naziva Setwise Boyer-Moore-Horspool. Rezultati koje su polučili razlikuju se od onih Silicon Defenesea, ali također pokazuju da sa različitim algoritmima za pronalaženje uzoraka snort i drugi uređaji koji zahtijevaju velike brzine pronalaženja postaju manje osjetljivi na DoS napade koji koriste sporost algoritama za pronalaženja uzoraka. Naznačeno je da bi specijalni algoritam koji bi se dinamički mijenjao iz standardnog Boyer-Moore-Horspool u Setwise Boyer-Moore-Horspool i u Aho-Corasick_Boyer-Moore, ovisno o duljini uzorkaka, bio efikasniji od bilo koje druge varijante.

Todd Lewis, autor Hanka, je još više unaprijedio ideju Aho-Corasic_Boyer-Mooreovog algoritma. Umjesto već postojeće implementacije Silicon Defensea, on je krenuo korak nazad i promotrio na kojim područjima se algoritam može unaprijediti. Problemi sa utroškom radne memorije i pitanje najkraćeg poklapanja uzorka koje je Silicon Defense naznačio drugačije su riješeni u Hanku. Hank pokreće sva pravila koja odgovaraju poklapanju, a također mnogo bolje upravlja radnom memorijom.

Ovaj dokument je samo naznačio što se sve može učiniti da bi se poboljšale performanse NIDS-a prilikom pronalaženja uzoraka, te je ukazao da postoji još dosta prostora za istraživanja i poboljšanja.

Odgovorni za razvoj NIDS-ova još uvijek ne mogu odlučiti koja tehnologija je bolja; poklapanje uzoraka (snort 1.x, ISS RealSecure) ili analiza protokola (ISS Sentry, ranije NetworkICE Sentry). Novije NIDS implementacije (ISS 7.0, snort 2.0) kombiniraju te tehnologije. Kombinacija analize protokola i pronalaženja uzoraka trebala bi objediniti dobre strane obiju tehnologija i učiniti NIDS sustave brzima, robusnima i sposobnima da se u najkraćem roku prilagode novim napadima.