ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU
KATEDRA ZA TELEKOMUNIKACIJE

– diplomski rad –

Nadgledanje računarskih mreža pomoću programskog paketa Nagios™

autor: Zoran Milenković

profesor: Dr Đorđe Paunović
mentor: Mr Nenad Krajnović

Beograd, avgust 2004.

Sadržaj

Uvod

Tema ovog rada je analiza mogućnosti programskog paketa Nagios za nadgledanje i upravljanje računarskim mrežama, kao i njegova implementacija na računarskoj mreži Elektrotehničkog fakulteta. Ovaj rad se može posmatrati i kao nastavak diplomskih radova kolega Saše Kostića i Stefana Pijevca koji su se takođe bavili problematikom nadzora i kontrole računarskih mreže s tom razlikom da su oni analizirali i implementirali *open source* programski paket SNIPS.

Iako je prvobitno zamišljeno da tema i ovog diplomskog rada bude dorada nekih nedostataka u pomenutom paketu SNIPS, uvidom u dokumentaciju i mejling liste oba paketa odlučio sam se da ipak detaljno ispitam mnogo bogatije mogućnosti Nagiosa.

Rad se sastoji iz nekoliko celina. Prvi deo rada je posvećen opisima brojnih funkcija implementiranih u jezgro programa. U drugom delu rada razmotreni su najvažniji Nagiosovi eksterni programski moduli – plaginovi. Kao osnovni materijal za pisanje ovih delova rada korišćena je zvanična Nagiosova dokumentacija[1]. U trećem delu opisan je postupak instalacije programa. U četvrtom delu detaljno sam opisao postupak konfigurisanja programa za potrebe nadgledanja računarske mreže Elektrotehničkog fakulteta što je ujedno bio i praktičan deo mog završnog ispita.

Određene celine rada koje su bile suviše detaljne da bi ušle u glavni tekst rada, a ipak su po mišljenju autora bile važne, date su u formi pet dodataka.

Programski paket Nagios

Uvodne napomene

Nagios je besplatan *open source* Linux softver namenjen nadgledanju i analizi stanja mrežnih resursa i komponenti. Razvija ga i održava *Ethan Galstad*. Originalni projekat je započet pod imenom *Netsaint*, a njegova poslednja zvanična verzija je 0.0.7. Dalji razvoj projekta je nastavljen pod novim registrovanim imenom Nagios™ čime su izbegnuti potencijalni pravni problemi oko korišćenja prethodnog imena Netsaint. Inače, novo ime je ostalo u istom duhu potičući od grčke reči *agios* koja znači svetac (takođe, engl. *saint* = svetac) i prefiksa *n* koje je prvo slovo engleske reči *network* (mreža).

Nagios je originalno razvijen za rad pod Linux platformom, ali paket je podržan pod skoro svim ostalim Unix kompatibilnim platformama. Licenciran je pod uslovima GNU GPL licence verzija 2 koju je objavila *Free Software Foundation.* Ovim se pod uslovima pomenute licence daje legalna dozvola za neograničeno kopiranje, distribuiranje i/ili menjanje izvornog koda Nagiosa.

Neke od mnogih mogućnosti Nagiosa uključuju:

- nadgledanje mrežnih servisa (SMTP, POP3, HTTP, NNTP, PING, itd.)
- nadgledanje lokalnih resursa hostova (opterećenje procesora, iskorišćenost memorije hard diskova, stanje mrežnih interfejsova, itd.)
- jednostavni plagin (*plug-in*) koncept koji dozvoljava korisniku da lako razvija i implementira sopstvene plaginove za nadgledanje specifičnih servisa
- paralelno nadgledanje servisa
- otkrivanje i razlikovanje hostova koji su nedostupni od onih koji su pali pomoću ugrađenog koncepta roditeljskih hostova i mrežne hijerarhije
- obaveštavanje u slučaju pojave neregularnog rada hostova ili servisa i njihovog oporavka (putem e-maila, pejdžera, SMS-a ili nekom drugom korisnički definisanom metodom)
- mogućnost da se definišu hendleri događaja (*event handlers*) koji su aktivni za vreme izvršavanja servisa ili dešavanja događaja na hostu i koji mogu da rešavaju probleme proaktivno
- automatsku rotaciju log-a
- podršku za implementaciju redundantnih servera za nadgledanje mreže
- podršku za implementaciju distribuiranog nadgledanja mreže
- opcioni web interfejs za uvid u tekući status mreže, obaveštavanje i uvid u istoriju problema, log datoteka itd.
- jednostavna šema autorizacije na web interfejsu kojom se lako kontrolišu korisnička prava pristupa informacijama

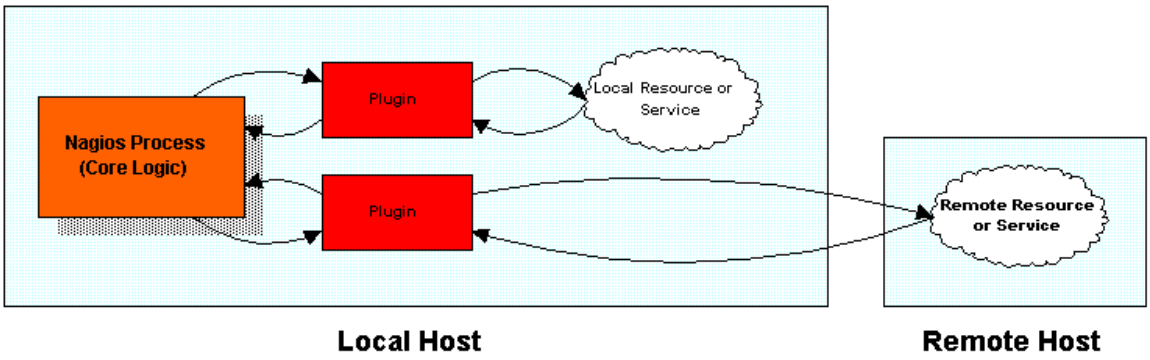
Trenutno, stabilna verzija je 1.1 dok je Nagios 2.0 još uvek u beta fazi. Razlozi zbog kog verzija 2.0 još uvek nije ozvaničena ne leži toliko u programskim bagovima koliko u nedostatku potpune dokumentacije. U ovom radu opisana je instalacija i konfiguracija Nagios 1.1 pod Gentoo Linux-om.

Plagin arhitektura

Programski paket Nagios zasnovan je na jednostavnoj *plug-in* arhitekturi čiji koncept podrazumeva funkcionalnu podelu paketa na jezgro programa i eksterne programe koji se nazivaju plaginovima (engl. *plug-in)*. Jezgro Nagiosa, koje čini centralni proces, u dokumentaciji još označavan kao *Nagios Process* ili *Core Logic*, ne poseduje nikakve interne mehanizme provere statusa nadgledanih objekata. Umesto toga, ono se u potpunosti oslanja na plaginove koji obavljaju celokupni posao nadgledanja. Samim tim, Nagios jezgro je beskorisno bez svojih plaginova.

Princip rada Nagiosa se ukratko može opisati na sledeći način. Nagios izvršava plagin kad god se stvori potreba za proverom statusa određenog servisa ili hosta koji se nadgleda. Plagin radi **nešto** (ovde treba primetiti ovaj uopšteni izraz) da bi izvršio proveru i jednostavno Nagiosu vraća rezultate te provere. Primljeni rezultati se obrađuju i preduzimaju se neophodne akcije ako je tako nešto potrebno i, naravno, definisano u konfiguracionim datotekama (pokretanje *event* hendlera, slanje obaveštenja, itd.)

Na slici 1. prikazan je način na koji su plaginovi odvojeni od jezgra programa. Nagios pokreće plaginove koji potom proveravaju lokalne ili udaljene resurse ili servise nekog tipa. Kada plaginovi završe proveru resursa ili servisa, prosto predaju rezultate provere nazad Nagiosu na obradu.



Slika 1. Plugin arhitektura Nagiosa

Dobra strana plagin arhitekture je što pruža praktično neograničene mogućnosti nadgledanja. Ako se proces provere nečega može automatizovati, onda se to može nadgledati pomoću Nagiosa. Uz Nagios distribuciju dolazi određeni broj standardnih plaginova koji pokrivaju provere gotovo svih uobičajenih mrežnih servisa i resursa kao što su dostupnost TCP/UDP portova, opterećenje procesora, popunjenost particija hard diska, brzina odziva pinga, SNMP promenljive, itd. U slučaju potrebe nadgledanja nekog specifičnog servisa, korisnik veoma lako može da napiše sopstveni plagin, budući da za to nije potrebno poznavanje izvornog koda jezgra i da postoje dokumentovane smernice za razvijanje sopstvenih plaginova[2], koje su dosta jednostavne.

Loša strana plagin arhitekture jeste činjenica da Nagios apsolutno nije "svestan" onoga što se nadgleda. Nadgledani objekat može biti napon na procesoru, brzina nekog ventilatora, popunjenost hard diska, statistika mrežnog saobraćaja, temperatura u mašinskoj sali, ili nešto sasvim drugo. Kao takav, Nagios ne može da generiše grafike promena osim za egzaktne vrednosti statusa resursa tokom vremena. Dakle, on može da prati samo promene statusa tih resursa. S druge strane, plaginovi su specijalizovani programi koji tačno "znaju" šta nadgledaju i kako da izvedu proveru za koju su specijalizovani. Takođe, oni mogu opciono da vraćaju i

podatke o performansama zajedno sa statusnim informacijama. Ovi podaci o performansama se potom mogu predati nekoj eksternoj aplikaciji koja bi mogla da generiše grafike informacija specifičnih za servis (iskorišćenost diska, opterećenje procesora, saobraćaj određenog interfejsa rutera, itd).

Konfiguracija

Nagios čuva svoju konfiguraciju u određenom broju tekstualnih datoteka ili u MySQL bazi podataka koja sadrži tabele analogne konfiguracionim datotekama. U ovoj implementaciji konfigurisanje je zbog jednostavnosti izvedeno u formi tekstualnih datoteka. Datoteke generalno mogu biti smeštene bilo gde na sistemu, ali dobra praksa je da sve budu smeštene u `/etc/nagios/` direktorijumu, što je i ovde ispoštovano. Definisanje organizacije i strukture konfiguracionih datoteka objekata je većim delom prepušteno korisniku što zahteva nešto više truda pri planiranju inicijalne konfiguracije, ali istovremeno pruža mogućnost potpunog prilagođavanja programa specifičnostima nadgledane mreže.

Osnovne konfiguracione datoteke su `nagios.cfg`, `cgi.cfg` i `resource.cfg`. Glavna konfiguraciona datoteka, `nagios.cfg`, sadrži brojne direktive kojim se globalno utiče na rad jezgra programa. U njoj se definišu imena i putanje do svih ostalih konfiguracionih datoteka uključujući i tzv. konfiguracione datoteke objekata. Na sličan način, `cgi.cfg` svojim direktivama kontroliše način rada CGI skriptova, dok je osnovna svrha risors datoteke, `resource.cfg`, čuvanje korisnički definisanih makroa koji obezbeđuju poverljive informacije (imena korisničkih naloga, lozinke ili parametri za pristup bazi podataka) od neautorizovanog pristupa.

Pomenute konfiguracione datoteke objekata sadrže definicije objekata pri čemu se termin "objekat" koristi kao zajednički izraz za sve relevantne podatke koji Nagiosu opisuju nadgledanu mrežu. U ovim datotekama se čuvaju definicije svih servisa i hostova od interesa, definicije procedura kojim će se nadgledanje izvršavati, kao i definicije kontakata koji će se obaveštavati u slučaju neregularnosti. Pregled svih tipova objekata koji se koriste u Nagiosu i kratak opis dati su u tabeli 1.

Generalno postoje dva načina da se definišu objekti u Nagiosu. Prvi način, koji je koristio i Netsaint, podržan je samo zbog kompatibilnosti unazad. Ovaj metod podrazumeva unošenje potpunih definicija **svih** objekata u konfiguracione datoteke. Obzirom da broj objekata u konfiguraciji može biti jako veliki (od nekoliko desetina do više hiljada), ovaj način se ne preporučuje. Sa ciljem da se izbegnu glomazne i nepregledne konfiguracione datoteke i omogući fleksibilno i lako definisanje objekata, (samim tim i održavanje Nagiosa) objektima je ugrađena mogućnost nasleđivanja osobina nekog drugog objekta koji objektu "nasledniku" služi kao obrazac i još neke zgodne osobine koje će kasnije biti objašnjene. Odatle se drugi način definisanja objekata naziva metod na bazi obrazaca (*template based method*). Mogućnosti i detaljno razrađeni primeri ovog preporučenog metoda, dati su u okviru glave sa pregledom svih Nagiosovih konfiguracioih datoteka.

Nagios kao i većina drugih *open source* projekata deo svoje funkcionalnosti temelji na drugim *open source* projektima kao što je na primer *Apache* HTTP server. Stoga je jasno da je za ispravan rad Nagiosa potrebno konfigurisati i sav dodatni softver na koji se Nagios oslanja. U okviru **Dodatka B** dat je detaljan postupak konfigurisanja ovog servera za potrebe Nagiosovog web interfejsa. Pri tom treba naglasiti da Nagios može da radi jednako dobro i sa drugim HTTP

serverima.

tip objekta	opis
host (*host*)	Opisuje hardverski uređaj ili komponentu koja puža neki servis (server, radna stanica, ruter, svič, mrežni štampač itd.)
servis (*service*)	Opisuje servis koji pruža neka hardverski uređaj ili server (HTTP, DNS, email itd.)
grupa hostova (*host group*)	Individualni hostovi se mogu grupisati u grupe hostova. Pri tom host može biti član više grupa. Ovo pojednostavljuje pridruživanje servisa čitavoj grupi hostova.
kontakt (*contact*)	Opisuje kontakte koje će Nagios da alrmira u slučaju neregularnog ponašanja servisa ili hostova u mreži
grupa kontakata (*contact group*)	Individualni kontakti se mogu grupisati u grupe kontakata pri čemu jedan kontakt može biti član više grupa kontakata. Ovim se pojednostavljuje pridruživanje servisa ili hosta grupi nadležnih kontakata.
komanda (*command*)	Opisuje način izvršavanja plaginova ili eksternih programa
vremenski period (time period)	Opisuje dnevni i/ili nedeljni vremenski interval u toku kog je dozvoljeno izvršavanje provera hosta ili servisa i/ili obaveštavanje kontakata o statusu istih
eskalacija servisa (*service escalation*)	Opisuje metod eskalacije obaveštenja za dati servis
zavisnosti servisa (*service dependencies*)	Opisuje zavisnost određenog servisa od statusa drugih hostova i/ili servisa
eskalacije hosta (*host escalations*)	Opisuje metod eskalacije obaveštenja za određeni host
zavisnosti hosta (*host dependencies*)	Opisuje zavisnost statusa određenog hosta od statusa drugih hostova
eskalcije grupe hostova (*hostgroup escalations*)	Opisuje metod eskalacije obaveštenja za određenu grupu hostova

Tabela 1. Tipovi objekata u Nagiosu

Na kraju, funkcionalnost Nagiosa se može proširiti korišćenjem brojnih Nagiosovih dodataka. Najvažniji od njih su `nrpe` (*Nagios Remote Plug-in Executor*) i `nsca` (*Nagios Service Check Acceptor*). Ovi dodaci poseduju zasebne konfiguracione datoteke koje kontrolišu njihov rad. Više reči o radu i korišćenoj konfiguraciji pomenutih pomoćnih daemon-a dato je u okviru dodataka C i D.

Funkcije jezgra

U ovom delu biće opisane neke funkcije jezgra Nagiosa koje su bitne za razumevanje logike rada programa.

Određivanje statusa hostova

Glavna svrha Nagiosa je da nadgleda status servisa koji se izvršavaju na hostovima i mrežnim uređajima, tj. koji su obezbeđeni fizičkim hostovima ili uređajima na mreži. Jasno je da ako host ili uređaj na mreži padne (status DOWN), svi servisi koje on nudi će pasti zajedno sa njim. Slično, ako host postane nedostupan zbog pada nekog hosta čijim se posredstvom obavlja komunikacija između njega i Nagios servera (status UNREACHABLE), Nagios neće biti u mogućnosti da nadgleda servise pridružene tom hostu.

Jezgro Nagiosa "ume" da prepozna ovakvu situaciju i uvek pokušava da proveri ovaj scenario kada se pojave problemi sa nekim servisom. Kad god provera servisa rezultuje tzv. non-OK statusom, Nagios će pokušati da proveri da li je host na kome se izvršava servis "živ". Tipično, to se vrši pingovanjem hosta i posmatranjem da li se prima bilo kakav odgovor. Ako i provera hosta vrati non-OK status, Nagios će zaključiti da postoji problem sa hostom. U toj situaciji Nagios će prigušiti sva potencijalna obaveštenja za servise koji se izvršavaju na tom hostu i jedino će obavestiti odgovarajuće kontakte da je host pao ili da je nedostupan. Ako provera hosta vrati OK status, Nagios će zaključiti da je host živ i poslaće obaveštenje da se posmatrani servis ne ponaša na očekivan način.

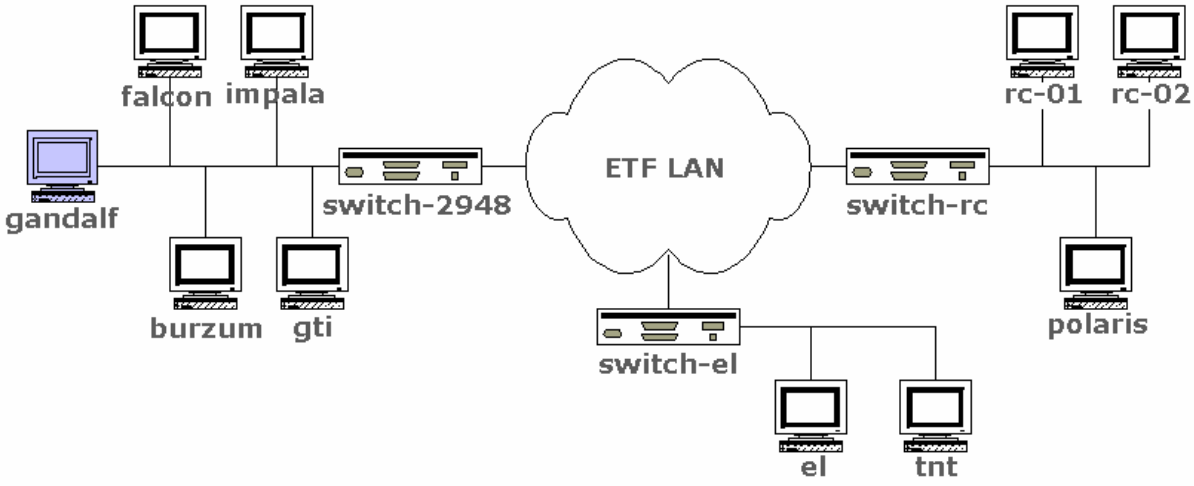
Da bi opisani mehanizam ispravno funkcionisao, pri definisanju hostova u konfiguraciji Nagiosa potrebno je jasno definisati hijerarhiju hostova u mreži u odnosu na Nagios.

Lokalni i udaljeni hostovi

Računar koji hostuje Nagios je uvek na vrhu u hijerarhiji hostova. Svi ostali hostovi se prema njemu odnose ili kao lokalni ili kao udaljeni hostovi. "Lokalni hostovi" (za Nagios) su hostovi na istom mrežnom segmentu na kom se nalazi računar koji hostuje Nagios tako da između njih nema rutera ili *firewall*-ova. Na slici 2 dat je primer nekih lokalnih hostova na mreži Elektrotehničkog fakulteta. Host `gandalf` hostuje Nagios i nadgleda sve ostale hostove i svičeve prikazane na dijagramu. Hostovi `falcon`, `impala`, `burzum`, `gti` i `switch-2948` se smatraju lokalnim hostovima u odnosu na `gandalf`

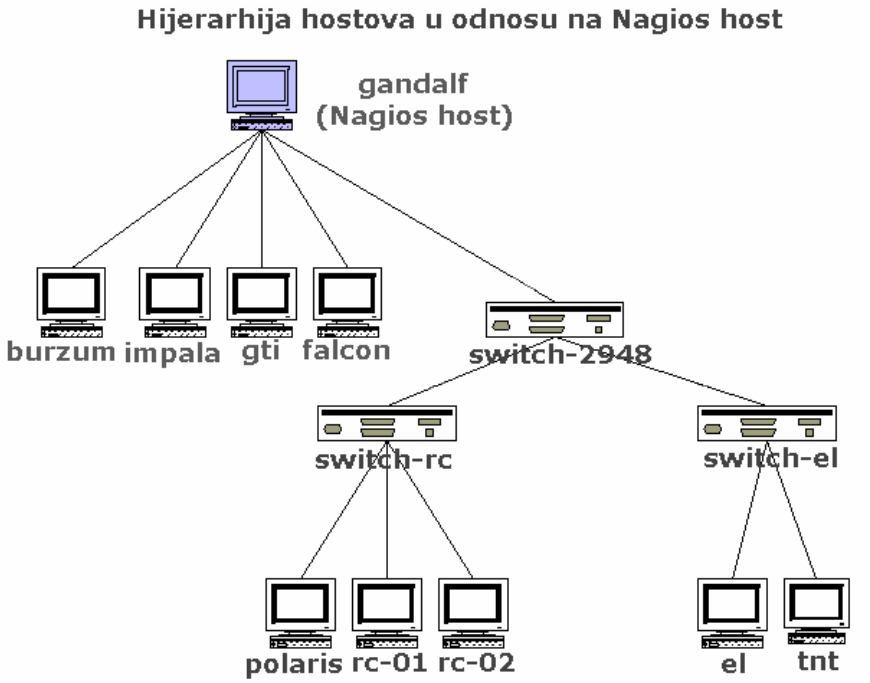
U okviru konfiguracije hostova, hijerarhija hostova u odnosu na host koji vrši nadgledanje se opisuje direktivom "parents" pri čemu ona u definiciji za lokalni host treba da ostane prazna (ili se izostavlja) pošto ovakvi hostovi nemaju zavisnosti niti hostove "roditelje".

"Udaljeni hostovi" su hostovi na segmentima mreže na kojima nije host sa Nagiosom. Na slici 2, svi hostovi `switch-rc`, `rc-01`, `rc-02`, `polaris`, `switch-el`, `tnt` i `el` su udaljeni hostovi u odnosu na host `gandalf`.



Slika 2. Koncept lokalnih i udaljenih hostova

Pri tom se može primetiti da su neki hostovi „dalji“ od drugih. U poređenju sa hostom `switch-el`, hostovi `el` i `tnt` su jedan hop dalje od hosta `gandalf`. Iz ovog posmatranja lako se može konstruisati drvo zavisnosti hostova kao na slici 3. koje je kasnije olakšava ispravno definisanje hostova u pogledu njihove hijerarhije.



Slika 3. Hijerarhija hostova sa stanovišta nadgledanja

Opcija “parents” u definiciji udaljenih hostova bi trebalo da bude kratko ime hosta koji je direktno iznad datog hosta na dijagramu drveta. Na primer, roditeljski host za host `tnt` je host `switch-el`. Na isti način, roditeljski host za `switch-el` je host `switch-2948`. Host `switch-2948` nema svog roditelja pošto je na istom mrežnom segmentu kao i `gandalf`, računar koji hostuje Nagios.

Nadgledanje lokalnih i udaljenih hostova

Provera lokalnih hostova na mreži je veoma jednostavna. Slučajno (ili namerno) isključivanje mrežnog kabla iz jednog od ovih hostova ne može da ugrozi nadgledanje ostalih hostova u mreži pošto ne postoje ruteri ili eksterne mreže između hosta koji vrši nadzor i drugih hostova na lokalnom segmentu.

Ako Nagios želi da proveri da li je lokalni host živ, prosto će izvršiti komandu provere ka tom hostu. Ako komanda vrati OK status Nagios tumači da je host živ (status UP). Ako komanda vrati bilo koje drugo stanje, Nagios tumači da je host pao (status DOWN).

Provera statusa udaljenih hostova je malo komplikovanija od provere lokalnih. Ako Nagios ne uspeva da izvrši proveru nekog servisa na udaljenom hostu, on će prvo da utvrdi da li je host koji pruža taj servis pao ili je nedostupan. Postojanje hijerarhije hostova omogućava Nagiosu da razlikuje ova dva slučaja. Ako komanda provere udaljenog hosta vrati non-OK status, Nagios će "prošetati" drvetom zavisnosti kao na slici 3 dok ne dosegne njegov vrh ili dok provera nekog od roditeljskih hostova ne vrati OK status. Na ovaj način Nagios utvrđuje da li je problem servisa nastao padom hosta, padom mrežnog linka (dakle, zbog nedostupnosti) ili se zaista radi samo o padu servisa.

Iz prethodnog razmatranja se još može videti koliko je važna pozicija u mreži servera koji hostuje Nagios.

Virtuelni hostovi

U hijerarhiju hostova se po potrebi mogu uključiti i virtuelni hostovi (*dummy hosts*) koji fizički ne postoje. Njihovo uvođenje nije neophodno. Motivacija za definisanje ovakvih hostova je najčešće povećanje vizuelne preglednosti status mape hostova koju generiše `statusmap.cgi` skript u slučaju kada je grupa hostova u mreži fizički distribuirana po prepoznatljivim celinama koje korisniku olakšavaju uvid u stanje određenih delova mreže. Virtuelni host najčešće predstavlja neku prostoriju ili sprat zgrade kom određena grupa računara pripada fizički.

Na primer, studentske radne stanice u prostorijama Računarskog centra Elektrotehničkog fakulteta su raspoređene u tri različite studentske učionice, pri čemu su sve priključene na zajednički *Layer 3* svič, `switch-rc`, koji sa stanovišta nadgledanja predstavlja njihov roditeljski host. Povećanje preglednosti u ovom slučaju je ostvareno definisanjem virtuelnih hostova za svaku učionicu. Da bi se održao hijerarhijski odnos sviča i radnih stanica, svaka učionica–virtuelni host definisana je kao host "roditelj" grupi računara koji se fizički nalaze u njoj, dok je svim učionicama kao roditeljski host definisan `switch-rc`.

Vrste statusa

Tekuće stanje, u širem smislu te reči, nekog servisa ili hosta u Nagiosu određeno je dvema komponentama: **statusom** i **stanjem**.

Nagios razlikuje tri vrste statusa hostova. Host može imati jedan od sledeća tri statusa:

- UP – regularni status kada je host operativan,
- DOWN – neregularni status kada host nije operativan i

- UNREACHABLE – neodređeni status hosta koji je nedostupan procesu nadgledanja i koji treba da reflektuje nemogućnost Nagiosa da utvrdi pravi status hosta.

Na isti način, u Nagiosu servis može biti u jednom od četiri definisana statusa:

- OK – regularni status servisa kada se servis ponaša na očekivan način,
- WARNING – status upozorenja koji signalizira pogoršani status servisa, koji pri tom još uvek funkcioniše,
- CRITICAL – neregularni status servisa koji signalizira grešku ili neregularno ponašanje servisa i
- UNKNOWN – neodređeni status servisa koji signalizira da iz nepoznatih razloga status servisa nije mogao biti određen.

Ovde treba primetiti da su iznad data samo uopštena tumačenja statusa hostova i servisa sa kojima jezgro operiše. Pravo tumačenje statusa hostova i servisa zavisi od konkretnog slučaja kao i načina provere. Primera radi, uobičajeni način provere statusa hostova je ispitivanje pomoću ping komande, tj. provera njegovog odziva na ICMP eho zahteve. Ako host ne odgovori na ping, imaće status DOWN i smatraće se neoperativnim. Međutim, u slučaju hosta sa više mrežnih interfejsova u istoj situaciji, mogli bismo **samo** da zaključimo da je dati interfejs hosta van funkcije.

Tipovi stanja

Tipovi stanja su ključni deo Nagiosovog jezgra jer se njima kontroliše pokretanje *event* hendlera i alarmiranje nadležnih konatakata. Nagios razlikuje dva tipa stanja – "SOFT" i "HARD" stanje.

Tokom rada programa, iako je nadgledani servis ili host u ispravnom stanju, usled izvesnih problema na mreži moguće je da za određeni servis ili host Nagios dobije negativan rezultat provere (non–OK status). Da bi se sprečili lažni alarmi, Nagios omogućava da se definiše broj ponovnih pokušaja provere servisa ili hosta pre nego što se zaključi da postoji realan problem. Maksimalni broj ovih pokušaja se kontroliše `max_check_attempts` direktivom u definicijama hostova i servisa. U zavisnosti od toga koji je pokušaj provere u toku, određen je tip stanja u kome se nalazi host ili servis. Postoji nekoliko izuzetaka u ovoj logici nadgledanja, ali to će biti razmotreno kasnije. U daljem tekstu opisani su načini ulaska serivisa u određeni tip stanja, kao i akcije koje se pri tom preduzimaju.

Soft stanje

Servisi i hostovi ulaze u SOFT stanje u sledećim situacijama:

- kada provera hosta ili servisa rezultuje non-OK statusom, a da pri tom još nije pokušana ponovna provera određeni broj puta. Ovaj broj je definisan `max_check_attempts` direktivom u definicijama servisa i hostova, respektivno. Ovo se naziva SOFT stanjem greške.
- Kada se servis ili host oporavi iz SOFT stanja greške. Ovo se naziva SOFT oporavkom.

Događaji u soft stanju

Kada servis ili host uđe u SOFT stanje greške ili doživi SOFT oporavak dešava se sledeće:

- SOFT greška ili oporavak se loguju ako je to omogućeno `log_service_retries` ili `log_host_retries` direktivom u glavnom konfiguracionom datoteci.
- Izvršavaju se *event* hendleri (ako su definisani) i hendluju SOFT grešku ili oporavak servisa ili hosta. (Pre nego što se izvrši bilo koji hendler, $STATETYPE$ makro uzima vrednost "SOFT").
- Nagios ne šalje obaveštenja ni jednom kontaktu jer ne postoji realni problem sa servisom ili hostom.

Kao što se može videti, jedina važna stvar tokom SOFT stanja je izvršavanje *event* hendlera. Korišćenje *event* hendlera može biti posebno korisno ako želimo da pokušamo da rešimo problem pre nego što host ili servis pređu u HARD stanje.

Na strani 44 dat je primer *event* hendlera koji je na lokalnom serveru `gandalf` koji hostuje Nagios imao zadatak da restartuje Apache HTTP server u slučaju otkaza HTTP servisa.

Hard stanje

Servisi ulaze u HARD stanje u sledećim situacijama (ulazak hostova u HARD stanje će biti objašnjen kasnije):

- kada provera servisa vrati non-OK status pri čemu je prethodno pokušan određeni broj ponovnih provera definisan u definiciji servisa. Tada je servis u HARD stanju greške.
- Kada se servis oporavi iz HARD stanja greške. To se naziva HARD oporavkom.
- Kada provera servisa rezultuje non-OK statusom pri čemu je odgovarajući host ili DOWN ili UNREACHABLE. Ovo je odstupanje od opšte logike nadgledanja, ali ima savršenog smisla. Ako host nije živ, onda nema smisla pokušavati ponovnu proveru servisa.

Hostovi ulaze u HARD stanje u sledećim situacijama:

- kad provera hosta rezultuje non-OK statusom pri čemu je izvršen određeni broj ponovnih provera definisan u definiciji hosta. To je hard stanje greške.
- Kada se host oporavi iz HARD stanja greške. Ovo je HARD oporavak.

Promene hard stanja

Pre nego što prodiskutujem šta se dešava kada host ili servis uđe u HARD stanje, potrebno je znati kada se dešavaju promene HARD stanja. Promene HARD satanja se dešavaju kada se:

- OK status u HARD stanju promeni u non-OK status u HARD stanju.
- non–OK status u HARD stanju promeni u OK status u HARD stanju.
- non–OK status neke vrste u HARD stanju promeni u non–OK status druge vrste u HARD stanju (npr. iz WARNING statusa u UNKNOWN status)

Događaji u hard stanju

Šta se dešava kad servis ili host uđe u HARD stanje greške ili doživi HARD oporavak zavisi od toga da li se dogodila promena HARD stanja (opisano iznad).

Ako se dogodila promena HARD stanja i servis ili host je u non-OK statusu dogodiće se sledeće:

- problem servisa ili hosta se loguje

- izvršavaju se *event* hendleri (ako su definisani) koji hendluju HARD problem servisa ili hosta. (Pre nego što se izvrši bilo koji hendler, $STATETYPE$ makro uzima vrednost ”HARD”).
- kontakti će biti obavešteni o problemu hosta ili servisa (ako logika obaveštenja to dozvoli)

Ako se dogodila promena hard stanja i servis ili host je u OK statusu, dogodiće se sledeće:

- hard oporavak servisa ili hosta se loguje
- izvršavaju se *event* hendleri (ako su definisani) koji hendluju HARD oporavak servisa ili hosta. (Pre nego što se izvrši bilo koji hendler, $STATETYPE$ makro uzima vrednost ”HARD”).
- kontakti će biti obavešteni o oporavku hosta ili servisa (ako logika obaveštenja to dozvoli)

Ako se ne dogodi promena HARD stanja, a pri tom je servis ili host u non-OK statusu, dogodiće se sledeće:

- kontakti će biti ponovno obavešteni o problemu hosta ili servisa (ako logika obaveštenja to dozvoli).

Ako se ne dogodi promena HARD stanja, a pri tom je host ili servis u OK statusu, ništa se neće dogoditi zato što je host ili servis u OK statusu kao što je bio i pre provere.

Raspoređivanje provera servisa u vremenu

Konfiguracione direktive

Način na koji jezgro Nagiosa raspoređuje provere servisa u vremenu, izvršava ih i obrađuje, kontroliše se pomoću nekoliko globalnih i nekoliko lokalnih konfiguracionih direktiva.

Svaka definicija servisa obavezno sadrži tri direktive kojim se lokalno, za posmatrani servis, definišu uobičajeni period provere servisa, period provere servisa kada se nađe u SOFT stanju i validni period u toku kog je dozvoljeno izvršavanje provera:

- `normal_check_interval`
- `retry_check_interval`
- `check_period`

Pored navedenih direktiva, postoje još četiri direktive u glavnoj konfiguracionoj datoteci koje globalno utiču na način izvršavanja provera servisa. To su:

- `inter_check_delay_method`
- `service_inerleave_factor`
- `max_concurrent_checks`
- `service_reapear_frequency`

U daljem tekstu detaljnije će biti objašnjeno na koji način sve ove direktive utiču na provere servisa. Prvo treba pogledati kako su servisi inicijalno raspoređeni kada se Nagios proces (re)inicijalizuje.

Uobičajeni period provere servisa

Dokle god je određeni servis u HARD stanju, njegove provere se normalno raspoređuju sa učestalošću definisanom *`normal_check_interval`* direktivom.

Raspoređivanje provera u problematičnim situacijama

Kada nastupe problemi sa određenim servisom, jedna od stvari koja će se dogoditi je preraspoređivanje narednih provera. Ako `max_check_attempts` parametar u definiciji servisa ima vrednost veću od 1, Nagios će ponoviti proveru servisa pre nego što zaključi da problem zaista postoji. Dok se servis ponovno proverava on je u SOFT stanju, a provere servisa se raspoređuju u skladu sa učestalošću definisanom `retry_check_interval` direktivom u definiciji servisa, što može biti brže ili sporije u odnosu na `normal_check_interval` direktivu.

Ako se za `max_check_attempts` parametar zada vrednost 1, servis se nikad neće proveravati u intervalima definisanim `retry_check_interval` direktivom. Umesto toga, servis odmah dobija non-OK status, ali ostaje u HARD stanju i njegove provere će se i nadalje raspoređivati nepromenjenom učestanošću.

Ako definisani broj ponovnih provera servisa opet vrati non-OK status servisa, on prelazi u HARD stanje, šalje se obaveštenje nadležnim kontaktima, i vrši se preraspoređivanje njegovih budućih provera u skladu sa učestanošću definisanom `normal_check_interval` direktivom.

Opisana procedura se ne primenjuje u sledećim situacijama. Kada provera servisa vrati non-OK status, Nagios će odmah zatim proveriti status hosta kome je taj servis pridružen. Ako host ne bude imao UP status (već DOWN ili UNREACHABLE), Nagios će servisu odmah pridružiti HARD non-OK status i resetovaće trenutni broj pokušaja provera na 1. Pošto je servis u HARD stanju, naredne provere ovog servisa će biti preraspoređena u skladu sa normalnom učestalošću provera definisanom `normal_check_interval` direktivom umesto `retry_check_interval` direktivom.

Validni period provera

Direktiva `check_period` u definiciji servisa definiše validan vremenski period tokom kog Nagios dozvoljava izvršavanje provera za taj servis. Bez obzira na status u kom se određeni servis nalazi, ako se vremenski trenutak za koji je predviđeno izvršavanje provere ne nalazi unutar definisanog validnog perioda provera, provera se neće izvršiti. Umesto toga Nagios će preraspoređiti sledeću proveru servisa za prvi sledeći validni period. U suprotnom, provera servisa se izvršava.

Ovde treba naglasiti da su raspoređivanje i izvršavanje provera dve jasno razgraničene (iako povezane) stvari. Iako provera servisa možda neće moći da se izvrši u zadato vreme, Nagios može da je rasporedi za to vreme. Ovo se najčešće dešava tokom inicijalnog raspoređivanja van validnog perioda provera. Kada dođe vreme za izvršavanje provere, Nagios će proveriti da li ona zaista može da se pokrene u to vreme. Ako ne može, Nagios ne izvršava proveru već je samo preraspoređuje za neki kasniji trenutak.

Inicijalno raspoređivanje

Kada se Nagios (re)inicijalizuje, pokušaće da izvrši raspoređivanje inicijalnih provera svih servisa na način koji će minimizovati opterećenje nadgledajućeg i nadgledanih hostova. To se vrši razdvajanjem provera u vremenu kao i njihovim učešljavanjem. Vremensko razdvajanje provera (kontroliše je direktiva `inter_check_delay_method`) vrši se u cilju minimizacije i ekvalizacije opterećenja lokalnog hosta na kom se izvršava Nagios, a učešljavanje se vrši u cilju minimizacije i ekvalizacije opterećenja nadgledanih hostova.

Iako se provere servisa inicijalno raspoređuju na ovaj način i tako balansiraju opterećenje na hostovima, inicijalno opterećenje može biti prilično neravnomerno. Razlozi za to su činjenica da period provere nije isti kod svih servisa, nekim servisima je potrebno više vremena da se izvrše nego drugim, problemi sa hostovima i/ili servisima mogu da izazovu vremensko pomeranje izvršavanja provere jednog ili više servisa, itd.

Uvid u informacije o inicijalnom rasporedu provera servisa može se dobiti pokretanjem Nagiosa iz komandne linije sa –s opcijom

```
/usr/nagios/bin/nagios –s /etc/nagios/nagios.cfg
```

Ovo će prikazati osnovne informacije o načinu raspoređivanja provera servisa za tekuću konfiguraciju i kreiraće novu status log datoteku sa tačnim vremenskim trenucima u kojima su provere inicijalno raspoređene za izvršavanje. Pri tom treba naglasiti da pozivanjem Nagiosa sa ovom opcijom proces nadgledanja se ne pokreće.

Interval između dve uzastopne provere

Kao što je pomenuto ranije, Nagios pokušava da ujednači opterećenje na mašini na kojoj se izvršava ravnomernim raspoređivanjem inicijalnih provera servisa u vremenu. Način izračunavanja intervala između dve uzastopne provere servisa se kontroliše specificiranjem vrednosti `inter_check_delay_method` direktive u glavnoj konfiguracionoj datoteci. Sledeća formula ilustruje princip rada "*smart*" izračunavanja pošto je to opcija koja se obično koristi u normalnom radu.

inter_check_delay = (prosečni interval provere servisa) / (ukupni broj servisa)

Na primer, recimo da imamo hiljadu servisa koji periodično treba da se proveravaju u intervalu od 5 minuta. U realnom slučaju, neće se svi servisi proveravati sa istom učestalošću, ali zbog jednostavnosti pretpostavimo da su intervali provera jednaki za sve servise. Ako se za `inter_check_delay_method` izabere *smart* opcija, iz date formule lako se dobija da će interval između dve uzastopne provere servisa iznositi 0,005 minuta tj. 0,3 sekunde, čime se za zadati slučaj postiže optimalno raspoređivanje opterećenja u vremenu na računaru koji hostuje Nagios.

Učešljavanje servisa (*interleaving*)

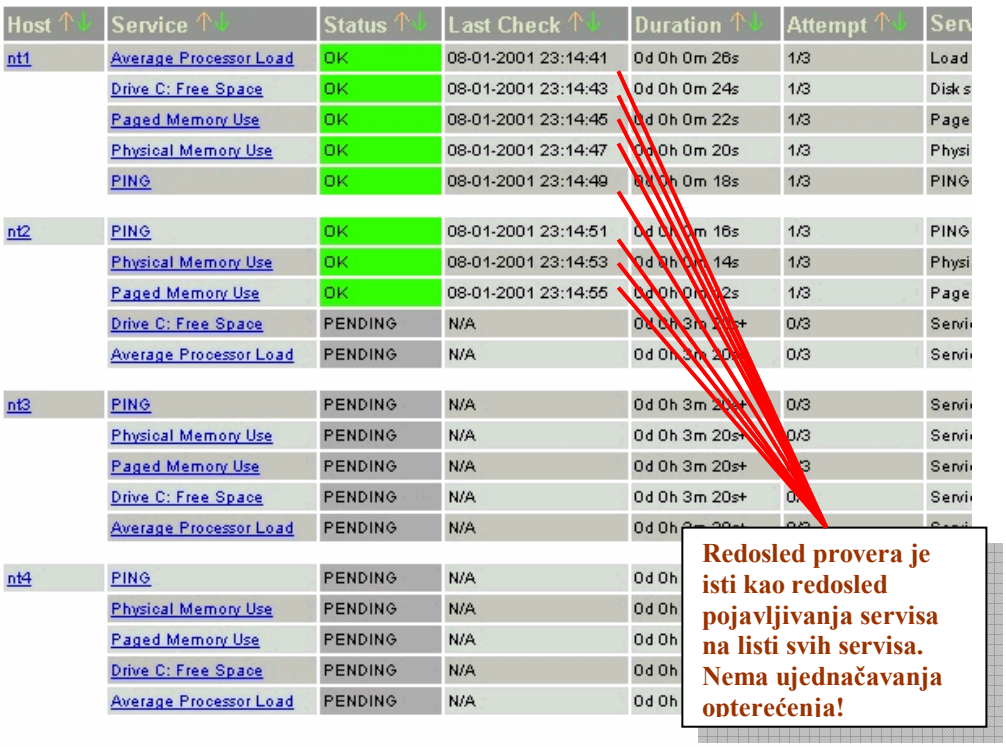
Ujednačavanje opterećenja nadgledanih hostova je naročito važno kod paralelizacije provera servisa. Ako se nadgleda ogroman broj servisa na udaljenom hostu i provere nisu raspoređene u vremenu, udaljeni host bi mogao "pomisliti" da je žrtva sinhronizovanog napada jer bi postojalo mnogo otvorenih konekcija na istom portu. Ideja ove optimizacije je da simultano proveravani servisi budu pridruženi različitiim hostovima. U tu svrhu u Nagiosu je uveden faktor učešljavanja čije će značenje biti objašnjeno u narednom primeru. Na način izračunavanja ovog faktora utiče se pomoću `service_interleave_factor` direktive u glavnoj konfiguracionoj datoteci. Sledeća formula ilustruje princip rada "*smart*" izračunavanja faktora učešljavanja provera pošto je to opcija koja se obično koristi u normalnom radu.

faktor učešljavanja = (ukupan br. servisa) / (ukupan br. hostova)

Na primer, predpostavimo da nadgledamo ukupno 1000 servisa na 150 hostova. Nagios će izračunati da faktor učešljavanja iznosi 7. To znači da će redosled inicijalnih provera servisa podrazumevati uzastopno raspoređivanje svakog sedmog servisa na listi od svih 1000 servisa. Ovaj proces se ponavlja dok sve provere servisa ne dobiju svoj raspored. Pošto su svi servisi sortirani po imenu hosta kome su pridruženi, ovo će pomoći ujednačavanju opterećenja kojem su izloženi nadgledani hostovi.

Slika 4. opisuje raspoređivanje inicijalnih provera servisa sa i bez učešljavanja, odnosno sa vrednostima faktora interlivinga 4 i 1, respektivno.

Slika 4a. Raspoređivanje provera sa faktorom učešljavanja 4



Slika 4b. Raspoređivanje provera bez učešljavanja. Faktor učešljavanja = 1

Maksimalni broj paralelnih provera servisa

Da bi se Nagios sprečio da zauzme više od zadatog dozvoljenog procesorskog vremena računara, može se zadati maksimalan broj paralelnih provera servisa tj. broj provera koje mogu simultano da se izvršavaju. Ovo je kontrolisano `max_concurrent_checks` direktivom u glavnoj konfiguracionoj datoteci.

Ideja je da se na neki način Nagiosu limitira opterećenje procesora. Pogrešno setovanje ove direktive je potencijalno opasno ako se zada previše niska vrednost ove opcije za sve pretpostavljene provere servisa. Kada nastupi trenutak da se izvrši određena provera, Nagios će se prvo uveriti da ne postoji više od *n* provera servisa koje se trenutno izvršavaju ili čekaju da se njihovi rezultati obrade (gde je *n* broj provera zadat u `max_concurrent_checks` direktivi). Ako je granica dostignuta, Nagios će odložiti izvršavanje provera dok se predhodne provere ne završe. Stoga je važno da se odredi razumna vrednost za ovu opciju. Pre svega mora da se zna sledeće:

- vrednost `inter_check_delay` koje Nagios koristi za inicijalno raspoređivanje provera servisa (ovo saznajemo pokretanjem Nagiosa sa –s opcijom).
- učestalost (u sekundama) obrade prikupljenih rezultata provera što je definisano `service_reapear_frequency` promenljivom u glavnoj konfiguracionoj datoteci.
- približno prosečno vreme potrebno proveri servisa da se izvrši (većina plaginova se nasilno prekida nakon 10s, tako da je prosek verovatno ispod ove vrednosti).

Sledeća formula ilustruje princip na osnovu kog Nagios računa maksimalan broja konkurentnih provera:

max_concurrent_checks =
max [(*service_reapear_freq*, sred.vreme izvršav. provere)] / (*inter_check_delay*)

Broj dobijen ovom formulom bi trebalo da bude razumna polazna vrednost za `max_concurrent_checks` parametar. Ovu vrednost je potrebno malo povećati ako se primeti da provere servisa i dalje kasne u odnosu na vremena za koja su raspoređene ili malo smanjiti ako Nagios i dalje uzima previše procesorskog vremena.

Na primer, pretpostavimo da se nadgleda 875 servisa sa prosečnim intervalom provere od 2 minuta. Interval između dve uzastopne provere će biti 0,137s. Ako se frekvencija obrade rezultata provera postavi na 10s, vrednost `max_concurrent_checks` parametra se može grubo proceniti na sledeći način (podrazumevano je da je prosečno vreme izvršavanja provere servisa manje od 10s):

maks. br. paralelnih provera = ceo deo (10 / 0,137)

U ovom slučaju izračunata vrednost je 73. To ima smisla pošto će Nagios (u proseku) izvršavati više od 7 novih provera servisa u sekundi pri čemu se obrada rezultata izvršava tokom 10 sek. To znači da je moguće istovremeno postojanje nešto više od 70 procesa (*threads*) provera servisa koje se izvršavaju ili čekaju da njihovi rezultati budu obrađeni. U ovom slučaju, savetuje se povećanje vrednosti `max_concurrent_checks` parametra na 80 pošto će uvek postojati neka kašnjenja kada Nagios bude obrađivao rezultate provera i u isto vreme bude vršio još neke poslove.

Očigledno, da bi se uverili da na sistemu sve radi glatko potrebno je izvršiti testiranje sa nekoliko vrednosti ove promenljive, pri čemu bi ovaj proračun trebalo da da grubu procenu tražene vrednosti.

Što se tiče ove implementacije Nagiosa, hardver servera na kom sam radio je bez problema podnosio ponuđeno opterećenje pa je `max_concurrent_checks` direktivom dozvoljen neograničen broj paralelnih provera.

Provere hostova

Za razliku od provera servisa, provere hostova se ne vrše periodično. Umesto toga one se izvršavaju samo u slučaju kada provera servisa koji je pridružen tom hostu vrati non-OK status. Nagios proverava host da bi zaključio da li je UP, DOWN ili UNREACHABLE. Ako prva provera hosta vrati non-OK status, Nagios će nastaviti sa proverom hosta:

a) dok se ne dostigne maksimalni broj ponovnih pokušaja provera hosta (definisano `max_check_attempts` direktivom u definiciji hosta) ili
b) dok provera hosta ne rezultuje OK statusom.

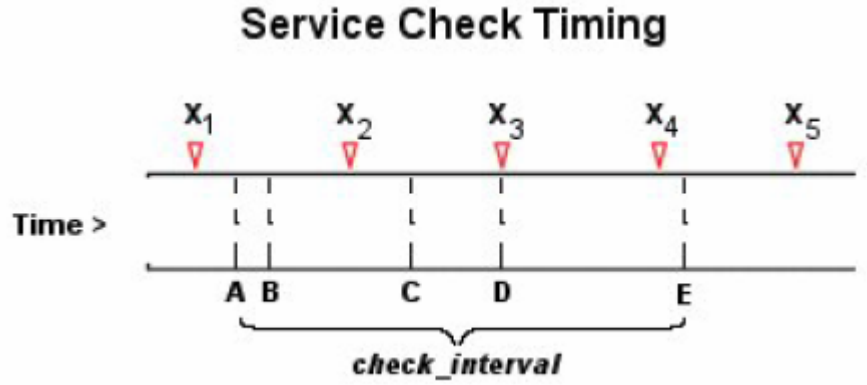
Takođe, kada Nagios proveri status hosta privremeno se obustavljaju izvršavanja novih provera servisa, obrada rezultata drugih provera servisa, itd. Ovo može malo da uspori odvijanje procesa nadgledanja i da izazove mali zastoj provera servisa, međutim da bi Nagios nastavio sa daljim proverama problematičnog servisa, neophodno je da se utvrdi status hosta.

Kašnjenje

Raspoređivanje i izvršavanje provera servisa u Nagiosu funkcioniše na *best effort* bazi. Individualne provere servisa se tretiraju kao događaji niskog prioriteta te one mogu da budu odložene ako postoji potreba za izvršavanjem događaja višeg prioriteta. Primeri događaja visokog prioriteta su: rotacija log-a, provere eksternih komandi i obrada rezultata provera. Izvršavanje i obradu provera servisa dodatno usporavaju provere hostova.

Primer raspoređivanja

Princip raspoređivanje provera servisa, njihovo izvršavanje i obrada njihovih rezultata je ilustrovan sledećim prostim primerom. Na slici 5, sa x_n su označeni trenuci u kojima se vrši prikupljanje i obrada rezultata provera. Oni su periodično raspoređeni sa učestalošću definisanom `service_reaper_frequency` direktivom u glavnoj konfiguracionoj datoteci. U ovim trenucima jezgro Nagiosa obrađuje rezultate provera koje prikupljaju plaginovi, i po potrebi generiše zahteve za proverom hostova, izvršavanjem *event* hendlera ili obaveštavanjem.



Slika 5. Raspoređivanje provera servisa u vremenu

U ovom primeru provera nekog servisa je raspoređena za izvršavanje u trenutku A. Međutim, Nagios se vratio svom *event* redu za čekanje, te provera zapravo počinje da se izvršava u trenutku B. Provera se završava u trenutku C, tako da je interval između trenutaka B i C istinito vreme izvršavanja provere.

Rezultati provere servisa se ne obrađuju odmah nakon okončanja provere, već se čuvaju do momenta kada jezgro prikuplja i obrađuje zatečene rezultate svih provera. Sledeći ovakav događaj se dešava u trenutku D tako da je to približno trenutak u kome počinje obrada rezultata (pravo vreme obrade može biti nešto posle trenutka D pošto je moguće da se prethodno obrađuju rezultati drugih provera servisa). U trenutku kada jezgro obrađuje rezultate provere posmatranog servisa, vrši se raspoređivanje njegove sledeće provere i smešta u red za čekanje. Ako pretpostavimo da je provera servisa vratila OK status, sledeća provera će biti raspoređena za trenutak E koji je udaljen od prethodno raspoređenog trenutka A tačno za normalni interval provere servisa. Ovde je važno uočiti da sledeća provera (trenutak E) nije preraspoređena na bazi trenutka u kom je stvarno (tj. sa zakašnjenjem) počela da se izvršava!

Izuzetno, ako se trenutak u kom započinje stvarno izvršavanje provere (trenutak B) desi posle trenutka sledeće provere servisa (trenutak E), Nagios će to kompenzovati raspoređivanjem sledećeg trenutka provere za još jedan normalni interval. Ovome se pribegava da Nagios ne bi uzaludno pokušavao da drži tempo sa proverama u slučaju da postane preopterećen.

Vremenski intervali

Mogućnost definisanja vremenskih intervala obezbeđuje veću kontrolu nad trenucima kada provere servisa mogu da se izvršavaju, kada obaveštenja o hostovima ili servisima mogu da se šalju i kada kontakti mogu da primaju obaveštenja. Sa ovim opcijama dolaze i potencijalni problemi o kojima će više reči biti kasnije.

Kako vremenski periodi utiču na provere servisa

Bez implementacije vremenskih perioda Nagios bi nadgledao sve definisane servise 24 h dnevno, 7 dana u nedelji (24x7). Dok je to odgovarajuće za mnoge potrebe nadgledanja, postoje servisi kojima to nije neophodno. Primera radi, ne postoji potreba za celodnevnim nadgledanjem štampača koji se inače koriste samo tokom radnog vremena. Takođe, status nekih razvojnih servera na mreži obično nije kritičan i nadgledanje njihovog statusa nije od značaja tokom vikenda. Iz ovakvih razloga u Nagios je implementirana mogućnost definisanja vremenskih intervala i perioda koji omogućavaju kontrolu nad vremenom kada će koji servisi moći da se nadgledaju.

U svakoj definiciji servisa, argument direktive `check_period` omogućava da se definiše vremenski period u kome je Nagiosu dozvoljeno da nadgleda posmatrani servis. Pre nego što Nagios pokuša da prerasporedi proveru servisa, on će se prvo uveriti da li sledeća provera upada u definisani period. Ako ne, Nagios će rasporediti sledeću proveru na početak sledećeg validnog perioda. To znači da servisi možda neće biti proveravani tokom narednog časa, dana ili nedelje, itd.

Potencijalni problemi sa vremenskim periodima

Pri korišćenju perioda koji ne pokrivaju čitav 24x7 period, mogući su izvesni problemi, naročito ako servis (ili njegov odgovarajući host) postane neoperativan u situaciji kada je sledeća provera servisa odložena za sledeći validni period provera. Evo nekih problema koji se mogu javiti:

- kontakti se ne obaveštavaju ponovno o problemima sa servisom do izvršavanja sledeće provere,
- ako se servis oporavi tokom perioda van validnog perioda za provere, kontakti neće biti obavešteni o tom oporavku,
- status servisa će se na web interfejsu i u status log datoteci prikazivati kao nepromenjen do sledeće provere,
- ako su svi servisi pridruženi određenom hostu podešeni na isti period provere, problemi sa hostom ili oporavci neće biti registrovani dok se ne izvrši jedna provera servisa (i stoga obaveštenja mogu da kasne ili da uopšte ne budu poslata).

Očigledno, potencijalni problemi se uglavnom sastoje u nepouzdanom prikazu podataka, te je na korisniku da odluči da li mu je bitnije da ima pouzdane informacije o stanju nekih manje važnih hostova ili servisa na mreži, ili da umanji opterećenje sistema sprečavanjem izvršavanja provera tokom časova van radnog vremena, vikendom i sl.

Kako vremenski periodi utiču na obaveštavanje kontakata

Verovatno najkorisnija stvar kod vremenskih perioda je kontrola trenutka kada će obaveštenje biti poslato nadležnim kontaktima. Pomoću

`service_notification_period` i `host_notification_period` direktiva u definicijama kontakata može se definisati period u toku kog je dozvoljeno obaveštavanje kontakta. Ovde treba primetiti da je omogućeno zasebno definisanje različitih periode za obaveštenja o hostovima i servisima. Ovo je, na primer, od koristi ako se želi da svakodnevno budemo obaveštavani o hostovima, a da obavešavanje o statusu servisa bude omogućeno samo radnim danima. Važno je napomenuti da bi ova dva perioda obaveštavanja trebalo da pokriju sve raspoloživo vreme tokom kog kontakt može biti obavešten. Takođe, kontrola perioda obaveštavanja može da se definiše za svaki servis i host ponaosob.

Pomoću `notification_period` direktive u definiciji hosta može se kontrolisati kada je Nagiosu dozvoljeno da šalje obaveštenje o problemima ili oporavcima hosta. Kada Nagios kreira obaveštenje on će se pre slanja uveriti da li tekući trenutak ulazi u validni period definisan `notification_period` direktivom. Ako je trenutak validan, Nagios će pokušati da obavesti sve kontakte o problemu sa hostom. Neki od kontakata možda neće primiti ovo obaveštenje ako njihov lokalno definisani (u definiciji kontakta) `host_notification_period` u datom trenutku to ne dozvoljava. Ako dati trenutak ne ulazi u `notification_period`, niko se ne obaveštava.

Periodi obaveštavanja o servisima kontrolišu se na sličan način. Definisanjem `notification_period` direktive u definiciji servisa, kontroliše se period u kome Nagios može da obaveštava kontakte o problemima ili oporavcima tog servisa. Kada Nagios kreira obaveštenje on će se pre slanja uveriti da li tekući trenutak ulazi u validni period definisan `notification_period` direktivom. Ako je trenutak validan, Nagios će pokušati da obavesti sve kontakte o problemu sa servisom. Neki kontakti možda neće biti obavešteni ako njihov lokalno definisani (u definiciji kontakta) `svc_notification_period` to ne dozvoljava u datom trenutku. Ako pak dati trenutak ne ulazi u `notification_period`, niko se ne obaveštava.

Potencijalni problemi sa obaveštavanjem kontakata

Kada se radi o obaveštavanju kontakata, ne može se pojaviti nijedan značajniji problem usled proizvoljnog definisanja perioda obaveštavanja kontakata. Jedino što se može desiti jeste da kontakti možda neće uvek biti obavešteni o problemima hostova ili servisa, ili o njihovim oporavcima. Ako tekući trenutak ne ulazi ni u period obaveštavanja servisa ili hosta, niti period obaveštavanja kontakta, obaveštenje se briše.

Zaključak

Vremenski periodi omogućavaju zaista preciznu kontrolu nad trenucima izvršavanja funkcija nadgledanja i obaveštavanja. S druge strane, ova funkcionalnost jezgra može da donese izvesne probleme. U slučaju pojavljivanja problema sa tekućom implemetacijom, korisnik uvek može da se vrati korišćenju standardnog “24x7” perioda za sve provere i sva obaveštenja.

Korišćenje validnih perioda obaveštavanja kontakata bi bilo jako važno ako bi se obaveštavanje realizovalo pomoću SMS ili pejdžing poruka. Pošto je u ovoj implementaciji Nagiosa korišćeno obaveštavanje putem elektronske pošte, konfigurisanju validnih perioda nije posvećena naročita pažnja. Validni vremenski periodi za sve provere i sva obaveštenja su standardni “24x7” period.

Obaveštavanje

Jedna od jako važnih funkcija ugrađenih u jezgro Nagiosa je funkcija alarmiranja ili obaveštavanja nadležnih kontakata u slučaju da određeni servis ili host dobije problematičan status.

Kada se podiže alarm

Odluka o slanju obaveštenja donosi se logikom provere servisa i provere hostova. Obaveštenja o hostu i servisu dešavaju se u sledećim slučajevima:

- promena HARD stanja.
- ostanak servisa ili hosta u HARD non-OK statusu tokom vremenskog intervala definisanog `notification_interval` direktivom u definiciji hosta ili servisa od trenutka slanja poslednjeg obaveštenja. Ovo se odnosi na specificirani host ili servis. Ako se ne želi periodično ponavljanje obaveštavanja, u `notification_interval` direktivi definicije hosta ili servisa treba postaviti vrednost 0 što onemogućava obaveštavanje više od jedanput za zadati problem.

Ko se obaveštava

Svaka definicija servisa ima opciju `contact_groups` koja definiše koje grupe kontakata primaju obaveštenja o tom servisu. Svaka grupa kontakata može da sadrži jedan ili više individualnih kontakata. Kada Nagios šalje obaveštenje o servisu, on će obavestiti svaki kontakt koji je član definisanih grupa kontakata u pomenutoj opciji definicije servisa. Nagios je svestan da bilo koji dati kontakt može biti član više grupa te stoga uklanja duplirana obaveštenja kontaktima pre nego što alarmira bilo koga.

Svaki host može da pripada jednoj ili više grupa hostova. Takođe, svaka grupa hostova ima opciju `contact_groups` koja definiše grupe kontakata koji primaju obaveštenja o hostovima iz date grupe hostova. Kada Nagios šalje obaveštenje o nekom hostu, on će obavestiti kontakte koji su članovi bilo koje grupe kontakata koji treba da budu obavešteni o bilo kom hostu te grupe hostova čiji je dati host član. Problem duplikata se rešava na isti način kao što je objašnjeno u prethodnom pasusu.

Filteri obaveštenja

Ako postoji potreba da se pošalje obaveštenje o nekom hostu ili servisu, to ne znači da će bilo koji kontakt biti odmah obavešten. Postoji nekoliko filtera koje potencijalno obaveštenje mora da prođe pre nego što bude smatrano za dovoljno važno da bi zaista bilo poslato. Čak i tad, određeni kontakti možda neće biti obavešteni ako lokalno definisani filteri obaveštenja (u definiciji kontakta) ne dozvoljavaju da im se pošalje obaveštenje.

Globalni filter (*programwide filter*)

Prvi filter koji obaveštenja moraju da prođu jeste opšta provera da li je opcija obaveštavanja omogućena na nivou čitave konfiguracije. To je inicijalno određeno direktivom `enable_notifications` u glavnoj konfiguracionoj datoteci, ali može biti promenjeno tokom rada programa putem web interfejsa. Ako su obaveštenja onemogućena na nivou čitave konfiguracije, niko neće biti obavešten. U suprotnom potencijalna obaveštenja se dalje obrađuju manje opštim filtrima.

Lokalni filteri obaveštenja

Prvi filter obaveštenja o hostovima ili servisima je provera da li tekući trenutak pripada intervalu predviđenom za planirano isključivanje hosta ili servisa (*downtime*). Ako pripada, niko se ne obaveštava. Ako u dati trenutak ne pripada periodu *downtime*-a, prelazi se na sledeći filter. Treba još reći da se obaveštenja o servisima pridruženim hostu za vreme njegovog *downtime* perioda takođe potiskuju.

Ako je uključena detekcija flepovanja (brza promena stanja UP i DOWN), sledeći filter koji će se primeniti na potencijalna obaveštenja je provera da li host ili servis možda flepuju. Ako servis ili host trenutno flepuju, niko se ne obaveštava. U suprotnom prelazi se na sledeći filter.

Treći filter je lokalna opcija obaveštavanja specifična za određeni host ili servis. Svaka definicija servisa sadrži opcije koje određuju da li se šalju obaveštenja za stanja WARNING, CRITICAL i RECOVERY. Slično definicijama servisa, svaka definicija hosta sadrži opcije koje određuju da li se šalju obaveštenja o stanju hosta DOWN, UNRECHEABLE ili RECOVERY. Ako potencijalno obaveštenje o servisu ili hostu ne prođe ovaj filter, niko se ne obaveštava. U suprotnom na obaveštenja se primenjuje sledeći filter.

Napomena: Obaveštenja o oporavku hosta ili servisa se šalju samo ako je poslato obaveštenje o originalnom problemu. Slanje obaveštenja o oporavku nečega što nikad nije bilo u problemu nema smisla, te se ono i ne šalje.

Četvrti filter je provera da li je u datom trenutku dozvoljeno slanje obaveštenja, tj. da li dati trenutak pripada validnom vremenskom intervalu u kom je dozvoljeno slati obaveštenja. Svaka definicija hosta ili servisa sadrži opciju `notification_period` koja definiše vremenski period u kome je dozvoljeno slati obaveštenja. Ako trenutak kreiranja obaveštenja ne ulazi u validni perod za slanje obaveštenja, niko se ne obaveštava. U suprotnom obaveštenje se predaje sledećem filtru.

Napomena: Ako se filter perioda ne prođe, Nagios će odložiti slanje obaveštenja o hostu ili servisu (ako imaju non-OK status) do početka sledećeg validnog perioda obaveštavanja. Ovim se osigurava da će kontakti biti obavešteni o problemima što je pre moguće po nastupanju prvog sledećeg validnog perioda za obaveštavanje.

Da li će se nad potencijalnim obaveštenjima izvršiti i poslednji skup filtera zavisi od dve stvari:

1. obaveštenje o problemu sa hostom ili servisom je već poslato u nekom trenutku u prošlosti,
2. host ili servis je ostao u istom non-OK statusu u kom je bio prilikom slanja poslednjeg obaveštenja.

Ako su zadovoljena oba kriterijuma, Nagios će se uveriti da je od poslednjeg obaveštavanja prošlo više od onoliko vremena koliko je specificirano `notification_interval` opcijom u definiciji hosta ili servisa. Ako nije prošlo dovoljno vremena niko se ne obaveštava. Ako je pak prošlo ili neki od dva navedena kriterijuma ovog filtra nije zadovoljen, obaveštenje će biti poslato. Da li se obaveštenje šalje individualnim kontaktima, stvar je sledećeg filtra.

Filteri obaveštenja kontakata

Kada obaveštenje prođe sve ove filtre, Nagios kreće da obaveštava sve one koji treba da saznaju za problem. To ne znači da će svaki kontakt zaista i primiti obaveštenje. Svaki kontakt ima svoj lokalni skup filtera koje obaveštenje mora da

prođe pre nego što bude poslato. Pri tom ovi filtri su definisani zasebno za svaki kontakt i ne utiču na obaveštavanje drugih kontakata.

Prvi filter koji mora da se prođe za svaki kontakt su opcije obaveštavanja. Svaka definicija kontakta sadrži opciju koja definiše tipove obaveštenja o servisu (WARNING, CRITICAL i RECOVERY) za koje je dozvoljeno slanje obaveštenja. Takođe, svaka definicija kontakta sadrži opciju kojom se definišu tipovi obaveštenja o hostu (DOWN, UNREACHABLE i RECOVERY) za koje je dozvoljeno obaveštavanje. Ako obaveštenje o hostu ili servisu ne prođe ove filtre, kontakt neće biti obavešten. U suprotnom obaveštenje se predaje sledećem filtru.

Napomena: Obaveštenja o oporavku hosta ili servisa se šalju samo ako su prethodno poslata obaveštenja o originalnom problemu.

Poslednji filter koji mora da se prođe je provera da li trenutak kreiranja obaveštenja pripada vremenskom intervalu u kom je dozvoljeno obaveštavanje datog kontakta. Svaka definicija kontakta sadrži opciju `notification_period` koja definiše ovaj vremenski period. Ako trenutak kreiranja obaveštenja ne ulazi u validni interval za obaveštavanje, kontakt neće biti obavešten. U suprotnom, kontaktu se šalje obaveštenje.

Metoda obaveštavanja koje nisu direktno ugrađene u Nagios

Shodno svojoj plagin arhitekturi, nijedna metoda obaveštavanja nije ugrađena u jezgro Nagiosa i posao obaveštavanja je prepušten spoljašnjem entitetu. Jezgro zahteva jedino da mu se u konfiguraciji definiše ime i putanja programa ili skripta kome se prosleđuju obaveštenja. Kada bi, primera radi, provere servisa bile ugrađene u jezgro Nagiosa, korisniku bi bilo jako teško da dodaje nove metode obaveštavanja, modifikuje postojeće, itd. Ovako, svaki korisnik može da u svoj sistem za nadgledanje integriše za njega najpodesniji način obaveštavanja, kako u smislu lakoće korišćenja, tako i u smislu platforme koju koristi i nivoa pouzdanosti koji želi da ostvari. Postoji zaista mnogo različitih načina da se realizuje obaveštavanje i na internetu se može naći mnogo rešenja i paketa koji to obavljaju.

Koji će se metod koristiti zavisi od konkretnog okruženja u kom se implementira Nagios. Kada se donese odluka o izabranom metodu, potrebno je da se instalira odogovarajući dodatni softver i iskonfigurišu komande obaveštavanja u konfiguracionim fajlovima. Ovo je lista samo nekoliko mogućih načina pomoću kojih Nagios može da šalje obaveštenja:

- Elektronska pošta (*email*)
- SMS (*Short Message Service*) ili MMS (*Multimedia Message Service*)
- Pejdžer
- Instant mesindžeri poput Yahoo, ICQ, MSN, Jabber i sl.
- Audio signali

U ovoj instalaciji Nagiosa obaveštavanje nadležnih kontakata realizovano je na najjednostavniji mogući način, elektronskom poštom. Na serveru `gandalf` koji hostuje Nagios podignut je mail server, *sendmail*, a u konfiguraciji Nagiosa su definisana dva jednostavna skripta, `notify-by-email` i `host-notify-by-email` koji kreiraju poruke obaveštenja za servise i hostove, respektivno, i prosleđuju ih *sendmail*-u.

Korisni izvori

Generalno, sve što može da se uradi iz komandne linije može da se upotrebi i kao komanda za obaveštavanje. Radi kompletnosti, ovde je dat kratak pregled

nekih metoda obaveštavanja koje nisu isprobane u okviru ovog rada, ali koje su preporučene u zvaničnoj dokumentaciji Nagiosa ili preko zvanične Nagiosove mejling liste.

Za alternativno korišćenje elektronske pošte za obaveštavanje preko pejdžere ili mobilnih telefona mogu se koristiti neki od sledećih paketa. Oni se mogu koristiti u kombinaciji sa Nagiosom za obaveštavanje preko modema o nastalim problemima. Pri tom ne treba se u potpunosti oslanjati na ovakvo rešenje jer i sam *email* servis može da otkaže ako postoje problemi u mreži.

- Gnoki (SMS softver za kontaktiranje Nokia GSM telefona)
- QuickPage (softver za alfanumeričke pejdžere)
- Sendpage (softver za pejdžing)
- SMS_Client (program sa komandnim interfejsom za slanje poruka na pejdžere ili mobilne telefone)

Zvučno obaveštavanje je jedna od nestandardnih metoda koja se takođe može korisititi. Za zvučno obaveštavanje na nadgledajućem serveru (sa sintetizovanim govornim porukama) preporučuje se programski paket *Festival*. Za zvučno obaveštavanje na udaljenim mašinama na kojima se ne izvršava nadgledanje vredi pogledati *Network Audio System* (NAS) i *replay* projekte.

Eskalacije obaveštenja

Nagios sadrži mogućnost definisanja eskalacije obaveštavanja kontakata za hostove i servise. Pod eskalacijom obaveštenja se podrazumeva definisanje scenarija obaveštavanja nadležnih kontakata u slučaju da problem sa servisom, hostom ili grupom hostova nastavi da perzistira. U okviru takvog jednog scenarija, posle određenog broja poslatih obaveštenja moguće je uključivanje ili isključivanje određene grupe kontakata sa liste kontakata koji se obaveštavaju, kao i promena dužine intervala između dva obaveštenja.

Eskalacija obaveštenja mogu se definisati za servis, host ili hostgrupu. Konfigurisanje eskalacija se vrši definisanjem odgovarajućih objekata za dati servis, host ili grupu hostova u konfiguracionoj datoteci `escalations.cfg`. U daljem tekstu kroz jednostavne primere ilustrativan je princip rada eskalacija.

Kada dolazi do eskalacije obaveštenja

Obaveštenja se eskaliraju ako i samo ako se jedna ili više definicija eskalacija podudari sa tekućim obaveštenjem koje se šalje. Ako se za servis, host ili grupu hostova za koje se šalje obaveštenje ne pronađe nijedna validna definicija eskalacije koja odgovara tekućem obaveštenju, Nagios će obavestiti redovne grupe kontakata koje su specificirane u definiciji servisa, hosta ili grupe hostova. Ako pak za tekuće obaveštenje pronađe validna definicija eskalacije obaveštavanje se vrši na osnovu te definicije. U sledećem primeru date su dve definicije eskalacije za nadgledani HTTP servis koji se hostuje na računaru `gandalf`:

```
define serviceescalation {
      host_name               gandalf
      service_description     HTTP
      first_notification      3
      last_notification       5
      notification_interval   90
      contact_groups          nagios-admins,linux-admins
      }
```

```
define serviceescalation{
      host_name               gandalf
      service_description     HTTP
      first_notification      6
      last_notification       10
      notification_interval   60
      contact_groups          nagios-admins,linux-admins,everyone
      }
```

Odmah se može primetiti da postoje "rupe" u datim definicijama eskalacija obaveštenja. Konkretno, prvo i drugo obaveštenje nisu obuhvaćena eskalacijama niti sva preostala posle desetog. Za sva ona obaveštenja koja nisu obuhvaćena u definicijama eskalacija, Nagios koristi default grupe kontakata definisane u definiciji servisa. U svim primerima pretpostavljeno je da je ta predefinisana grupa kontakata nagios-admins.

Ako bi se htelo da definicija eskalacije pokrije beskonačan opseg obaveštenja, u smislu od određenog obaveštenja pa nadalje, u direktivi *last_notification* treba zadati vrednost 0.

Grupe kontakata

Kada se definišu eskalacije obaveštenja, važno je imati na umu da bilo koja grupa kontakata koja je bila član "nižih" eskalacija (onih sa nižim rednim brojevima obaveštenja) bi trebalo da budu uključene u definicije "viših" eskalacija, da bi bili sigurni da će svako ko bude obavešten o problemu nastaviti da biva obaveštavan kako problem bude eskalirao.

U prethodno datom primeru, prvi (ili najniži) nivo eskalacije uključuje grupe kontakata *nagios-admins* i *linux-admins*. Poslednji (ili najviši) nivo eskalacije uključuje *nagios-admins*, *linux-admins* i *everyone* grupu kontakata.

Ovde treba uočiti da je predefinisana grupa kontakata *nagios-admins* uključena i u obe definicije eskalacije kako bi kontakti iz ove grupe bili i dalje obaveštavani ako problem nastavi da postoji posle prva dva obaveštenja o servisu. Grupa kontakata *linux-admins* se prvo pojavljuje u "nižoj" definiciji eskalacije – oni se prvi put obaveštavaju kada se pošalje treće obaveštenje o problemu sa servisom. Kada bi u konfiguraciji postojala samo prva definicija eskalacije servisa, poslednje obaveštenje koje bi primili u slučaju opstanka problema, bilo bi peto obaveštenje. Pošto se u primeru želi da ova grupa kontakata (pored grupe *everyone*) bude obaveštavana i nadalje tokom narednih 5 obaveštenja, ona je takođe uključena i u „višu" definiciju eskalacije.

Preklapanje opsega eskalacija

Definicije eskalacije obaveštenja mogu imati opsege obaveštenja koji se preklapaju. Primer:

```
define serviceescalation {
      host_name               gandalf
      service_description     HTTP
      first_notification      3
      last_notification       5
      notification_interval   20
      contact_groups          nagios-admins,linux-admins
      }
```

```
define serviceescalation{
      host_name               gandalf
      service_description     HTTP
      first_notification      4
      last_notification       0
      notification_interval   30
      contact_groups          support
      }
```

U primeru iznad, opisan je sledeći scenario obaveštavanja:

- predefinisana grupa kontakata *nagios-admins* se obaveštava prvim i drugim obaveštenjem,
- grupe kontakata *nagios-admins* i *linux-admins* se obaveštavaju trećim obaveštenjem,
- zbog preklapanja definicija, sve tri grupe kontakata se obaveštavaju četvrtim i petim obaveštenjem,
- šestim i svim daljim obaveštenjima obaveštava se samo grupa kontakata support.

Obaveštenja o oporavku

Obaveštenja o oporavku su nešto drugačija za razliku o obaveštenja o problemu kada dođe do eskalacije.

Recimo da se u prethodnom primeru oporavak HTTP servisa dogodio nakon slanja trećeg obaveštenja definisanim kontaktima. Obaveštenje o oporavku servisa bilo bi zapravo četvrto poslato obaveštenje. Međutim, mehanizam eskalacije je dovoljno inteligentan da zaključi da obaveštenje o oporavku treba da pošalje samo onim kontaktima koji su prethodno obavešteni o problemu zaključno sa trećim obaveštenjem. U ovom slučaju, o oporavku će biti obaveštene grupe kontakata *nagios-admins* i *linux-admins*, ali ne i support iz već pomenutog razloga.

Intervali obaveštavanja

Korišćenjem *notification_interval* direktive u definiciji eskalacije servisa, hosta ili grupe hostova može se podešavati učestalost slanja eskaliranih obaveštenja.

Neka je u prvom navedenom primeru predefinisani interval slanja ponovnih obaveštenja za servise 240 min. (ova vrednost je zadata u definiciji servisa). Kada obaveštenje o servisu eskalira prilikom 3., 4. i 5. obaveštenja, interval između obaveštenja će se promeniti i iznosiće 90 min. Od 6. do 10. obaveštenja, interval između obaveštenja će iznositi 60 min. kao što je definisano u drugoj definiciji eskalacije, a od 11. obaveštenja pa nadalje, sva obaveštenja će se ponovo slati u predefinisanom intervalu od 240 min.

Kako je moguće preklapanje definicija eskalacija, kao i činjenice da host može biti član više grupa hostova, Nagios mora da se odluči za jednu od preklapajućih definicija. U slučaju kada postoji više validnih definicija eskalacija za određeno obaveštenje, Nagios će "poslušati" onu sa najkraćim intervalom obaveštenja.

U poslednjem navedenom primeru vidi se da se definicije eskalacija preklapaju, tj. obe (različito) definišu učestalost obaveštavanja prilikom slanja 4. i 5. obaveštenja. Za "preklopljena" obaveštenja Nagios će koristiti interval od 45 min, pošto je to manji prisutni interval u validnim derfinicijama eskalacija za ova

obaveštenja. U opštem slučaju više od dve definicije eskalacije, koristio bi se najmanji interval.

Na kraju još treba objasniti intervale obaveštenja čija je vrednost 0. Kada se zada interval dužine 0, Nagios će poslati samo jedno obaveštenje, i to ono prvo. Sva naredna obaveštenja za ovaj servis, host ili grupu hostova biće potisnuta. Primer:

```
define serviceescalation {
      host_name               gandalf
      service_description     HTTP
      first_notification      3
      last_notification       7
      notification_interval   60
      contact_groups          nagios-admins,linux-admins
      }

define serviceescalation{
      host_name               gandalf
      service_description     HTTP
      first_notification      4
      last_notification       8
      notification_interval   0
      contact_groups          nagios-admins,linux-admins,everyone
      }

define serviceescalation{
      host_name               gandalf
      service_description     HTTP
      first_notification      9
      last_notification       0
      notification_interval   35
      contact_groups          support
      }
```

U poslednjem primeru, maksimalni broj obaveštenja o problemu koji se može poslati u vezi sa posmatranim servisom jeste 4. Ovo je slučaj zbog zadatog intervala obaveštenja od 0 u drugoj definiciji eskalacije koja znači da će se slati samo jedno obaveštenje (počev i uključujući 4-to obaveštenje) dok će sva naredna obaveštenja biti potisnuta. Zbog toga ni vrednost zadata u direktivi `last_notification` u drugoj definiciji, ni čitava treća definicija, nemaju nikakvog efekta.

Indirektne provere statusa hostova i servisa

Najveći broj servisa i hostova na mreži može se nadgledati pomoću direktnog korišćenja plaginova sa računara koji hostuje Nagios. Primeri ovih direktno proverivih servisa su dostupnost web, email, ili FTP servera, i oni se mogu direktno proveravati Nagiosovim plaginovima jer su javno dostupni resursi. Međutim, postoji mnogo servisa koje je interesantno nadgledati, ali koji nisu javno dostupni kao ostali servisi. Stanje ovakvih privatnih resursa ne može se pratiti bez korišćenja posrednog programa tzv. agenta. Ovi "privatni" resursi ili servisi uključuju informacije kao što su iskorišćenost diska, opterećenje procesora i slično, na nadgledanim hostovima.

Indirektne provere predstavljaju način da se nadgledaju svi oni servisi i hostovi koji nisu direktno dostupni plaginovima na nadgledajućem serveru. Provere servisa koje zahtevaju neku vrstu agenta posrednika koji se izvršava na udaljenom nadgledanom računaru nazivamo indirektnim proverama.

Indirektne provere su korisne za:

- nadgledanje lokalnih resursa na udaljenim hostovima
- nadgledanje servisa i hostova iza *firewall*-a
- dobijanje realističnijih rezultata iz provera servisa osetljivih na vremensko kašnjenje između udaljenih hostova (npr. vreme ping odziva između dva udaljena hosta)

Postoji više načina za izvršavanje indirektnih provera. Međutim, ovde ću se zadržati samo na principu rada `nrpe` agentskog programa, dodatka za Nagios koji je razvio autor Nagiosa, Ethan Galstad. Ovaj dodatak je napisan sa ciljem da obezbedi način za izvršavanje plaginova na udaljenom hostu. Ime **`nrpe`** je skraćeno od *Nagios Remote Plugin Executor*.

Sam program `nrpe` predstavlja samo serverski deo klijent/server programskog para za indirektne provere koji se izvršava na nadgledanom hostu. Može da se izvršava bilo kao samostalni demon, ili kao servis pod `inetd` ili `xinetd` demonom. Klijentski deo aplikacije čini `check_nrpe` plagin pomoću kog Nagios jezgro komunicira sa `nrpe` agentom.

Što se tiče sigurnosti, kada se izvršava u modu daemona, `nrpe` agent vrši autentifikaciju zahteva za izvršavanjem plaginova prostim upoređivanjem IP adrese pozivajućeg hosta sa listom dozvoljenih IP adresa koja se nalazi u njegovoj konfiguracionoj datoteteci `nrpe.cfg`. Kada se izvršava pod `inetd`-om, pristup `nrpe` agentu može da se vrši pomoću TCPwrapper-a.

Funkcionisanje indirektnih provera bi se ukratko moglo opisati na sledeći način. Na strani nadgledajućeg računara, Nagios pokreće `check_nrpe` plagin koji se koristi da pošalje zahteve `nrpe` agentu za izvršavanje određenih plaginova na udaljenom hostu. Agent `nrpe` prima zahteve, pokreće zahtevane plaginove i vraća rezultate provera `check_nrpe` plaginu na nadgledajućem hostu. Plagin `check_nrpe` preuzima izlazne rezultate udaljenih plaginova i šalje ih nazad Nagiosu kao da su njegovi. Ovim se omogućava transparentni način izvršavanja plaginova na udaljenim hostovima.

Radi demonstracije mogućnosti Nagiosa, kod nekoliko hostova sam definisao po dva ping servisa, pri čemu je proveru jednog vršio `check_ping` plagin sa servera `gandalf`, dok je drugi ping servis proveravan posredstvom `check_nrpe` plagina `check_ping` plaginom instaliranim na serveru `galeb`. Na ovaj način, ne samo da je omogućeno nadgledanje dostupnosti tih hostova sa nadgledajućeg servera, već i praćenje dostupnosti hostova i praćenje parametara veze iz perspektive nekog drugog hosta u mreži (u ovom slučaju servera `galeb`).

Indirektne provere servisa

Slika 6. ilustruje princip rada indirektnih provera na primeru provera nekih servisa onako kako su realizovane na pojedinim hostovima u okviru praktičnog dela ovog rada.

Slika 6. Indirektne provere servisa preko galeba

U konkretnom slučaju, između nadgledajućeg hosta `gandalf`-a i pojedinih nadgledanih hostova (u ovom primeru su dati `switch-el`, `switch-rc` i `el`) nalazi se *firewall* koji ne pušta ICMP saobraćaj i onemogućava proveru odziva ovih hostova na ping. Da bi se odziv na ping ovih hostova ipak proveravao, na posredničkom hostu `galeb`, koji se takođe nalazi sa druge strane *firewall*-a, podignut je `nrpe` daemon i instalirani su Nagiosovi plaginovi. U ovoj situaciji neophodno je bilo instalirati samo odgovarajući plagin za proveru ping servisa, `check_ping`.

Kada Nagios želi da proveri status ping servisa na nekom od tri prikazana hosta, preko `check_nrpe` plagina `nrpe` agentu se upućuje zahtev za izvršavanjem `check_ping` plagina. Agent `nrpe` izvršava plagin i dobijene rezultate provere vraća `check_nrpe` plaginu koji ih potom prosleđuje Nagiosu, čime se ovaj ciklus indirektne provere ping servisa završava.

Na isti način, preko `nrpe` demona Nagios pokreće lokalne Nagiosove plaginove (instalirane na galebu) i nadgleda status privatnih resursa hosta galeb. Za razliku od provera javno dostupnih servisa, za provere privatnih servisa i resursa nekog hosta neophodno je prisustvo agenta posrednika.

Čitav postupak prevođenja, instalacije i konfiguracije nrpe daemona na hostu galeb dat je u **Dodatku C.**

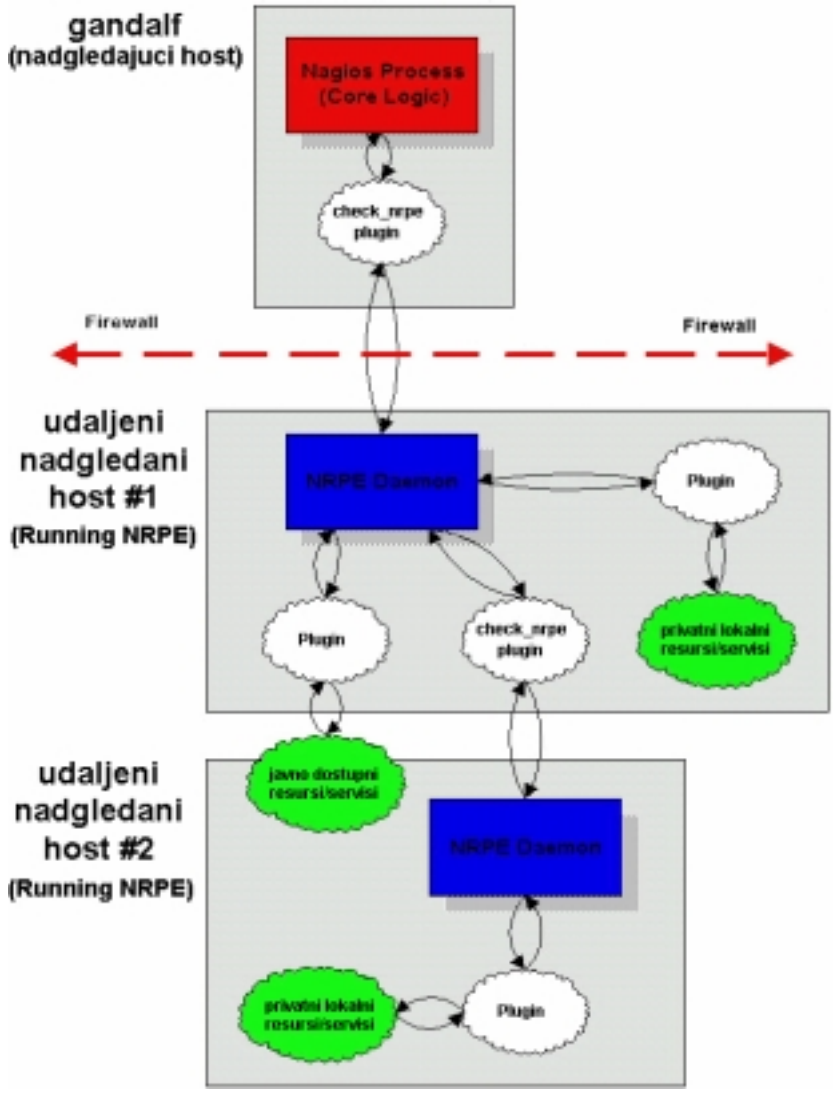
Višestruke indirektne provere servisa

Kada se vrši nadgledanje hostova koji su smešteni iza *firewall*-a u odnosu na računar koji hostuje Nagios, provere servisa na ovim mašinama mogu biti po malo komplikovane jer je obično blokirana većina dolaznog saobraćaja koji je neophodan za funkciju nadgledanja. Jedno rešenje za izvršavanje provera na

hostovima iza *firewall*-a bilo bi ostavljanje "uskog" prolaza u filtrima *firewall*-a koji bi Nagiosu omogućili da izdaje zahteve `nrpe` demonu na hostu zaštićenom *firewall*-om. Host unutar *firewall*-a bi potom mogao da bude korišćen kao posrednik u izvršavanju provera svih ostalih servera zaštićenih *firewall*-om.

Pomoću `nrpe` *daemon*-a i `check_nrpe` plagina moguće je višestruko ulančavanje indirektnih provera servisa. Iako je definisanje ovakvih provera servisa nešto komplikovanije, korisniku je ostavljena i ovakva mogućnost u slučaju da se za tako nešto ukaže potreba.

Slika 7. prikazuje princip rada višestrukih indirektnih provera servisa. Agentski program `nrpe` se izvršava na hostovima #1 i #2. Kopija koja se izvršava na hostu #2 omogućuje `nrpe` agentu na hostu #1 da izvršava provere privatnih servisa na hostu #2. Privatni servisi podrazumevaju broj procesa koji se trenutno izvršavaju na hostu, opterećenje procesora, iskorišćenost diska, itd. koji nisu direktno izloženi kao SMTP, FTP, ili HTTP servis. Podrazumeva se da su na hostovima #1 i #2 ispravno prevedeni i instalirani potrebni plaginovi.

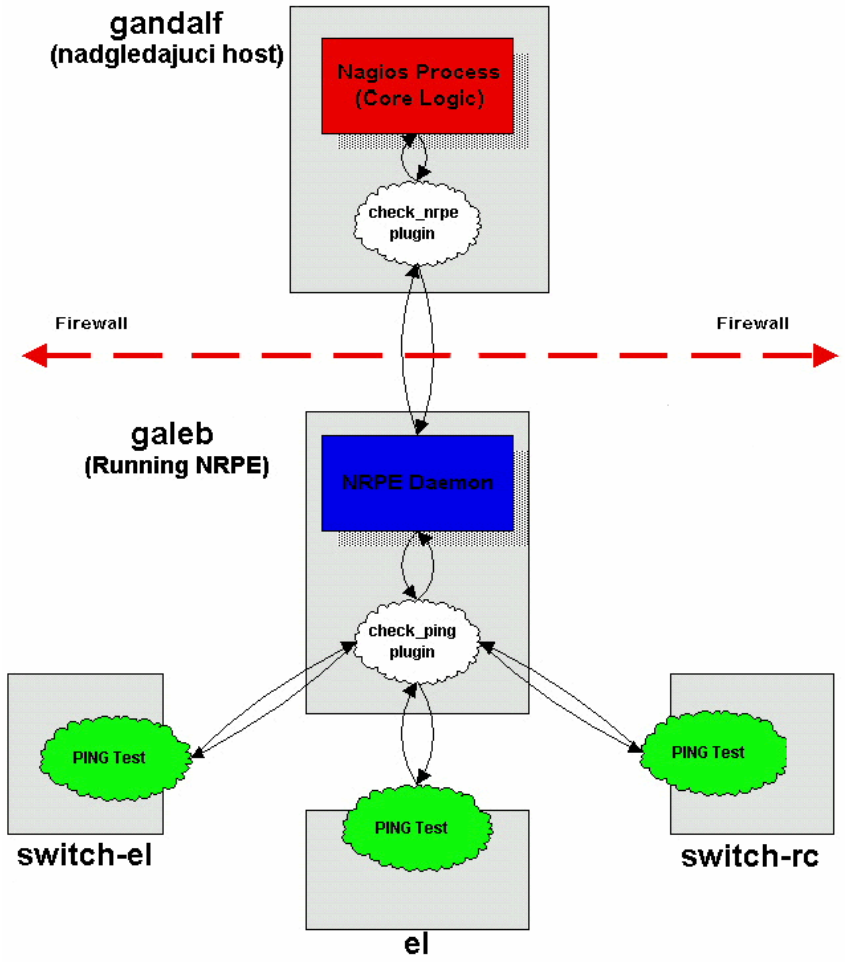


Slika 7. Višestruke indirektne provere servisa

Indirektne provere hostova

Indirektne provere hostova rade na istom principu kao i indirektne provere servisa. U osnovi, plagin koji se koristi u komandi provere hosta zahteva od agenta posrednika da izvrši proveru hosta. Indirektne provere hostova su korisne kada se udaljeni hostovi nalaza iza *firewall*-a ili kada se želi zabrana dolaznog saobraćaja nadgledanja za dotičnu mašinu. Ta mašina (udaljeni host #1 na dijagramu ispod) će izvršiti proveru hosta i vratiti rezultate nazad najstarijem `check_nrpe` plaginu (na centralnom serveru). Treba primetiti da sa ovakvom postavkom dolaze potencijalni problemi. Ako udaljeni host #1 padne, `check_nrpe` plagin neće biti u mogućnosti da kontaktira `nrpe` demona i Nagios će "verovati" da su udaljeni hostovi #2, #3 i #4 takođe pali iako to ne mora da bude slučaj. Ako je host #1 *firewall*, tada problem nije realan jer će Nagios detektovati da je on pao i obeležiti hostove #2, #3 i #4 kao nedostupne.

Dijagram ispod prikazuje primenu `nrpe` demona i `check_ping` plagina u izvršavanju indirektne provere hosta.



Slika 8. Indirektne provere hostova

Pasivne provere servisa

Generalno gledano, po načinu prikupljanja informacija od interesa, u nadgledanju računarskih mreža se koriste dva opšta modela. To su *pull* (engl. *pull* = vući) i *push* (engl. *push* = gurati) model. Ideja *pull* modela je da centralna aplikacija koja vrši nadgledanje, aktivno dovlači informacije o statusu nadgledane

mreže. U *push* modelu, informacije o statusu mreže se prikupljaju pomoću inteligentnih distribuiranih agentskih programa. Centralna aplikacija u ovom slučaju ne zahteva od agenata informacije o mreži. Prikupljene informacije se "guraju" od strane agentskih programa koji poseduju izvesnu inteligenciju da mogu samostalno da odluče kada je to potrebno.

Osnovni nedostaci prvog modela su preopterećenje servera na kom se izvršava proces nadgledanja i značajno povećanje saobraćaja na mreži. Nedostatak drugog modela je smanjena pouzdanost dobijenih informacija.

U ovom kontekstu, jedna od lepih osobina Nagiosa je da može da kombinuje upotrebu oba modela i da, pored aktivnog proveravanja servisa, može da obrađuje rezultate provere servisa koje su mu podnete od strane neke spoljne aplikacije. Provere servisa koje su izvedene i podnete Nagiosu od strane neke eksterne aplikacije nazivaju se pasivne provere. Pasivne provere se razlikuju od aktivnih po tome što ove provere servisa nisu inicijalizovane od strane Nagiosa.

Potreba za pasivnim proverama

Pasivne provere su korisne za nadgledanje servisa koji su:

- locirani iza *firewall*-a i stoga ne mogu da se proveravaju aktivno sa Nagios hosta
- asinhrone po prirodi i stoga ne mogu da se aktivno proveravaju na pouzdan način (npr. SNMP trapovi, sigurnosni alarmi, itd).

Princip rada pasivnih provera

Jedina prava razlika između aktivnih i pasivnih provera u Nagiosu je što aktivne provere inicira Nagios dok pasivne provere izvršavaju eksterne aplikacije. Kada eksterna aplikacija izvrši proveru servisa (bilo aktivno, bilo prijemom sinhronog događaja kao što je SNMP trap ili sigurnosni alarm) ona podnosi rezultate provere Nagiosu upisujući ih u datoteku eksternih komandi u propisanom formatu.

Kada Nagios sledeći put obradi sadržaj datoteke eksternih komandi, rezultati svih pasivnih provera servisa se smeštaju u red za čekanje za kasniju obradu. Red za čekanje koji se ovde koristi potpuno je isti kao onaj za smeštanje rezultata aktivnih provera.

Nagios će periodično prikuplja podnete rezultate provera servisa i uvrštava ih u red za čekanje na obradu. Svaki rezultat provere, bez obzira da li je provera bila aktivna ili pasivna, obrađuje se na isti način. Logika provere servisa je potpuno ista za obe vrste provera. Ovo obezbeđuje sličan metod rukovanja rezultatima i aktivnih i pasivnih provera servisa.

Način na koji eksterne aplikacije podnose rezultate provere servisa

Eksterne aplikacije mogu da podnesu rezultate provere servisa Nagiosu upisom PROCESS_SERVICE_CHECK_RESULT eksterne komande u datoteku eksternih komandi. Propisani format podnešenih rezultata provere je sledeći:

```
[<timestamp>] PROCESS_SERVICE_CHECK_RESULT; <host_name>;
<descriptiion>; <return_code>; <plugin_output>
```

gde je:

- `timestamp` – ternutak izvršenja (ili podnošenja) provere servisa u `time_t` formatu. Jedan blanko karakter posle desne zagrade se ne sme izostaviti!
- `host_name` – kratko ime hosta pridruženog servisu u definiciji servisa.
- `description` – opis servisa koji stoji u definiciji servisa
- `return_code` – povratni kod provere (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- `plugin_output` – tekstualni izlaz provere servisa (npr. tekst koji vraća plagin)

Drugi način podnošenja pasivnih provera servisa je preko web interfejsa korišćenjem *command* CGI skripta. Ovaj način podrazumeva da korisnik koji ima privilegiju zadavanja komandi preko web interfejsa, ručno unese rezultat pasivne provere. Na primer, ručno podnošenje provera servisa bi moglo da se primeni kod nadgledanja sigurnosti sistema. Pretpostavimo da se određenim softverom vrši detekcija sigurnosnih napada koji je tako iskonfigurisan da po detekciji nekih aktivnosti podnese CRITICAL status nadgledanog servisa Nagios serveru. Dalje, ako pretpostavimo da je problem takav da je neophodna reakcija mrežnog administratora, onda je zgodno da po otklanjanju problema, administrator ručno resetuje status servisa u OK status.

Naravno, da bi Nagios uopšte razmatrao podnete rezultate provere, servis mora biti prethodno definisan u odgovarajućoj konfiguracionoj datoteci. Rezultate provera onih servisa koji nisu bili konfigurisani pre njegove poslednje (re)inicijalizacije Nagios će prosto ignorisati.

Ako je sve što se želi dobijanje pasivnih rezultata za određeni servis (npr. aktivna provera servisa ne treba da se izvršava), u definiciji servisa treba postaviti direktivu *`active_checks_enabled`* na 0. Takođe treba se uveriti da je direktiva *`passive_checks_enabled`* u glavnoj konfiguracionoj datoteci postavljena na 1. Ako nije, Nagios neće obrađivati rezultate pasivnih provera servisa.

Primer *shell* skripta koji opisuje kako se principijelno podnose rezultati pasivne provere servisa Nagiosu dat je na kraju sledećeg poglavlja.

Podnošenje rezultata pasivnih provera servisa sa udaljenih hostova

Ako neka aplikacija koja se hostuje na istom računaru gde i Nagios podnosi rezultate pasivnih provera servisa, upisivanje rezultata provera u datoteku eksternih komandi je prilično jednostavno. Međutim, aplikacije na udaljenim hostovima ne mogu to da urade tako lako. Da bi se udaljenim hostovima omogućilo da šalju rezultate pasivnih provera servisa Nagios hostu, tvorac Nagiosa je razvio dodatni program koji obavlja ovaj posao, *Nagios Service Check Acceptor* (`nsca`).

Ovaj program se sastoji od *daemon*-a koji se izvršava na Nagios hostu po imenu `nsca`, i klijenta koji se izvršava na udaljenim hostovima, po imenu `send_nsca`. Demon osluškuje konekcije od udaljenih klijenata, vrši proveru autentičnosti poslatih rezultata i potom ih direktno upisuje u datoteku eksternih komandi. Ako na sistemu postoje instalirane `mcrypt`[3] biblioteke, komunikacija između klijenta i *daemon*-a može se kriptovati različitim algoritmima (DES, 3DES, CAST itd.). Program se konfiguriše pomoću jednostavnih konfiguracionih datoteka čiji je celokupni sadržaj dat u Dodatku D.

Korišćenjem opisanog dodatka podnošenje rezultata provera je veoma jednostavno i sa udaljenog hosta. U osnovi, jedino što eksterna aplikacija na udaljenom hostu treba da uradi jeste da klijentskom programu prenese argumente u odgovarajućem formatu.

U nastavku dat je primer upotrebe test skripta kojim sam testirao podnošenje pasivnih provera servisa sa hosta `galeb` na nadgledajući host `gandalf`.

```
/users/etf/mizoran/nagios/bin/submit_check_result galeb PING 0 “ping OK”
```

Sadržaj korišćenog probnog skripta *`submit_check_result`* je sledeći:

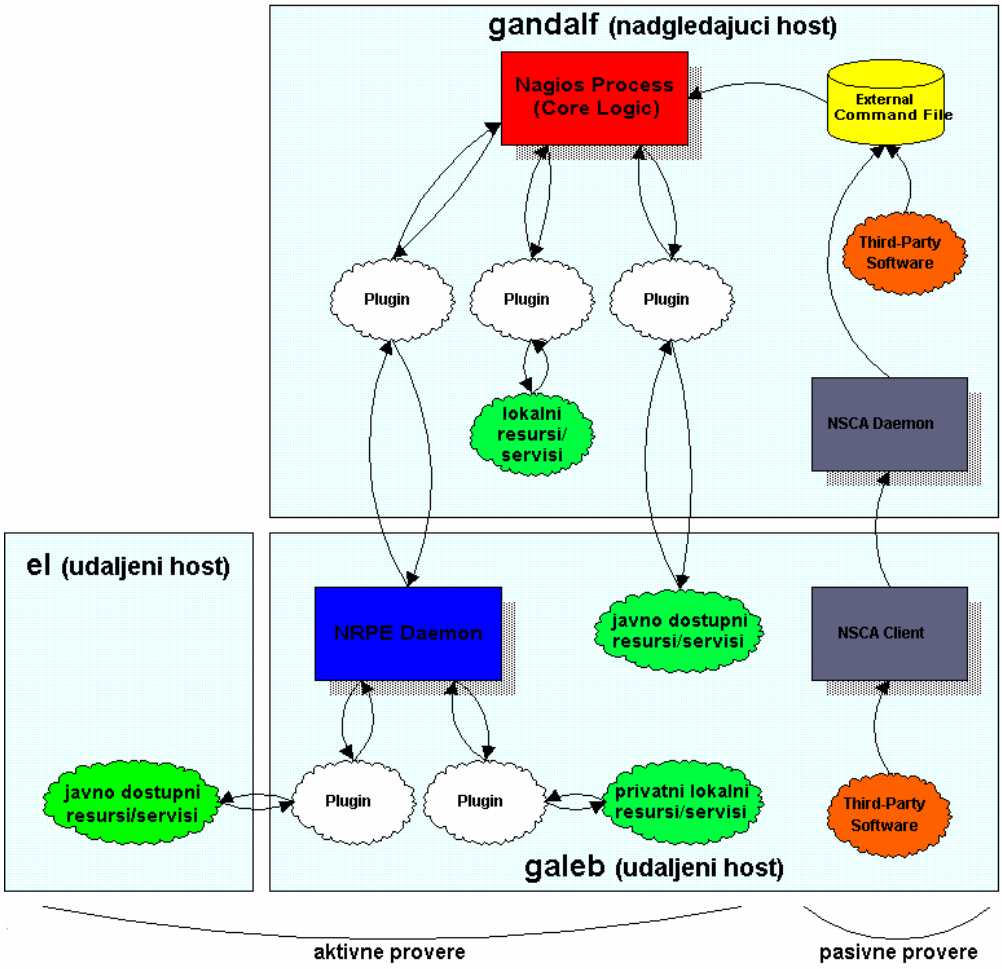
```
#!/bin/sh
# Argumenti:
# $1 = ime hosta u definiciji servisa
# $2 = ime/opis servisa u definiciji servisa
# $3 = povratni kod
# $4 = tekst sa propratnim informacijama
printf "$1\t$2\t$3\t$4\n" | /users/etf/mizoran/nagios/bin/send_nsca
–H gandalf -c /users/etf/mizoran/nagios/etc/send_nsca.cfg
```

Kombinovanje oba tipa provera servisa

Obzirom da su na raspolaganju i pasivne i aktivne provere servisa, prirodno je očekivati da se najbolji rezultati dobijaju kombinovanjem prednosti obadva tipa provera. Ovo jedino nije slučaj kada se implementira distribuirano monitoring okruženje sa centralnim serverom koji prima samo pasivne provere servisa (bez ijedne aktivne provere).

Aktivne provere su mnogo pogodnije za servise podložne periodičnim proverama (dostupnost nekog HTTP ili FTP servera i sl.), dok su pasivne provere zgodnije za rukovanje asinhronim događajima kao što su SNMP trapovi, napadi na sistem i slično.

Na slici 9. ilustrovan je primer kombinovane upotrebe aktivnih i pasivnih provera servisa. Narandžasti oblačići na desnoj strani slike su eksterne aplikacije (*Third-Party Software*) koje podnose rezultate pasivnih provera upisujući ih u Nagiosovu datoteku eksternih komandi (*External Command File*). Eksterne aplikacije predstavljene u gornjem oblačiću izvršavaju se na gandalfu, istom hostu gde je i Nagios, te one mogu direktno da upisuju svoje rezultate u komandnu datoteku.



Slika 9. Kombinovanje pasivnih i aktivnih provera servisa

Eksterne aplikacije predstavljene donjim oblačićem izvršavaju se na udaljenom hostu galebu i koriste nsca server/klijent programski par za prenos rezultata pasivnih promena Nagiosu.

Oblačići na levoj strani slike predstavljaju aktivne provere servisa koje izvršava Nagios. Prikazan je način provere lokalnih resursa (popunjenost diska na gandalfu itd.), javno dostupnih resursa na udaljenim hostovima (HTTP server, FTP server, itd.) i "privatnih" resursa na galebu (iskorišćenost diska, opterećenje procesora, broj trenutno logovanih korisnika, trenutni broj *zombie* procesa, itd.). U ovom primeru, privatni resursi na udaljenim hostovima se zapravo proveravaju korišćenjem nrpe programa koji olakšava izvršavanje plaginova na udaljenim hostovima.

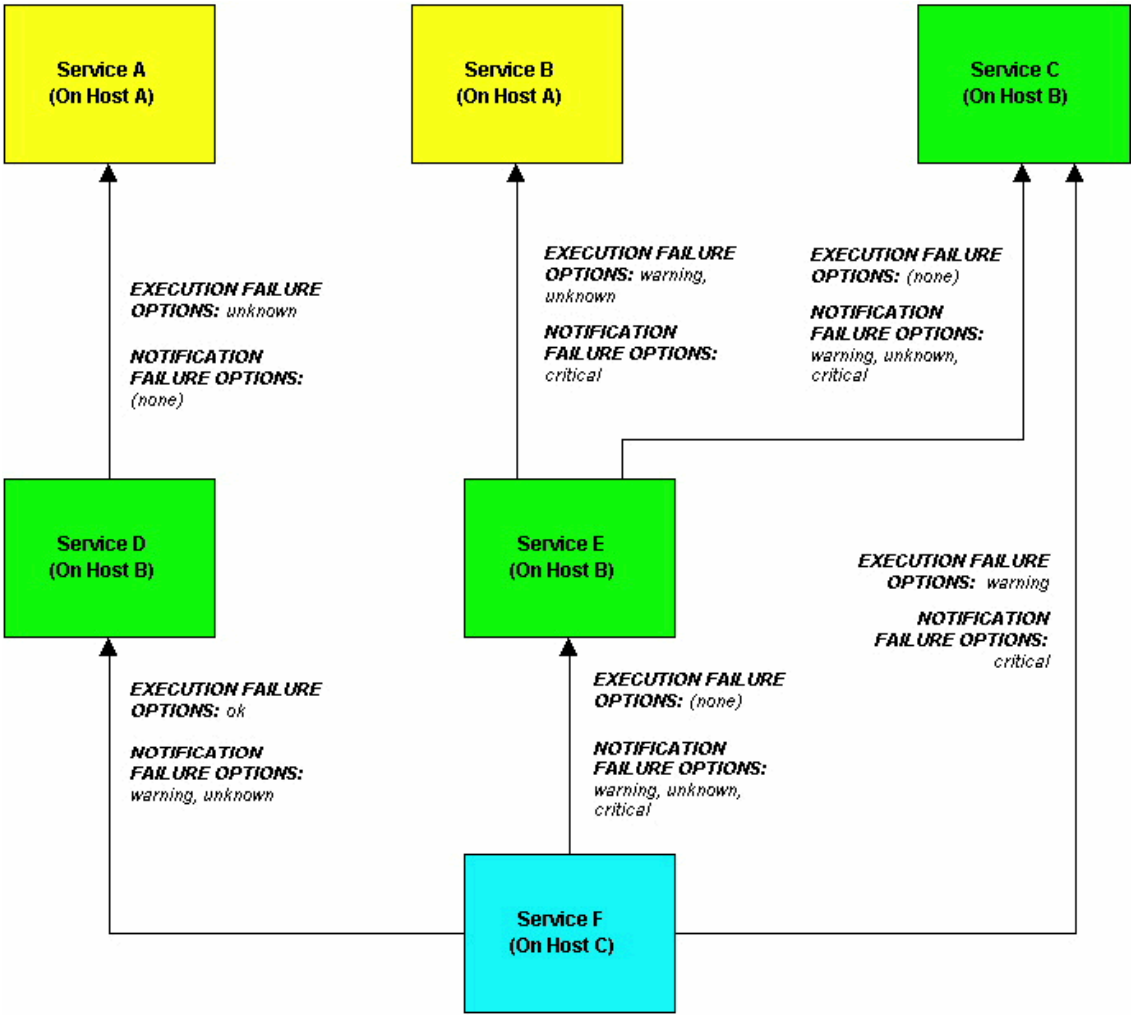
Zavisnosti hostova i servisa

Zavisnosti hostova i servisa su napredna pogodnost koja korisniku omogućava da kontroliše ponašanje hostova i servisa na osnovu statusa jednog ili više drugih hostova i servisa. Ovde će biti objašnjeno kako zavisnosti funkcionišu, zajedno sa razlikama između zavisnosti hostova i zavisnosti servisa.

Pregled zavisnosti servisa

Slika ispod prikazuje primer logičkog prikaza zavisnosti servisa. Treba uočiti nekoliko stvari:

1. servis može biti zavistan od jednog ili više drugih servisa
2. servis može biti zavistan od servisa koji nisu pridruženi istom hostu
3. zavisnosti servisa se ne nasleđuju
4. zavisnosti servisa se mogu koristiti za iniciranje izvršavanja servisa i potiskivanje obaveštenja o servisu pod različitim okolnostima (OK, WARNING, UNKNOWN i/ili CRITICAL status).



Slika 10. Zavisnosti servisa

Definisanje zavisnosti servisa

Zavisnosti servisa se definišu dodavanjem definicija zavisnosti servisa u objektnu konfiguracionu datoteku `dependencies.cfg`. U svakoj definiciji specificira se ime zavisnog servisa (*dependent*), ime servisa od koga zavisimo (*depending-on*) i kriterijum (ako uopšte postoji) koji treba da se zadovolji da bi se sprečilo izvršavanje određenih provera ili potisnula obaveštenja o određenim servisima ili hostovima.

Moguće je kreirati nekoliko zavisnosti za dati servis, ali potrebno je dodati zasebnu definiciju zavisnosti servisa za svaku zavisnost.

Na slici iznad definicija zavisnosti za servis F na hostu C bio bi definisan na sledeći način:

```
define servicedependency {
      host_name                          Host B
```

```
        service_description                     Service D
        dependent_host_name                     Host C
        dependent_service_description           Service F
        execution_failure_criteria              o
        notification_failure_criteria           n
        }

define servicedependency {
        host_name                               Host B
        service_description                     Service E
        dependent_host_name                     Host C
        dependent_service_description           Service F
        execution_failure_criteria              n
        notification_failure_criteria           w,u,c
        }

define servicedependency {
        host_name                               Host B
        service_description                     Service C
        dependent_host_name                     Host C
        dependent_service_description           Service F
        execution_failure_criteria              w
        notification_failure_criteria           c
        }
```

Kako se testiraju zavisnosti servisa

Pre nego što Nagios izvrši proveru servisa ili o njemu pošalje obaveštenje, prvo će proveriti da li taj servis zavisi od nekog drugog servisa. Ako ne, provera se izvršava ili se šalje obaveštenje kao što bi normalno i trebalo da bude. Ali ako servis ima jednu ili više zavisnosti, Nagios će proveriti svaku zavisnost na sledeći način:

1. proverava se tekući status servisa od koga dati servis zavisi
2. tekući status servisa od kog zavisi posmatrani servis se upoređuje sa zadatim kriterijumima sprečavanja izvršavanja ili obaveštavanja u definiciji zavisnosti (koja god da je relevantna u tom trenutku)
3. ako tekući status servisa od koga zavisi posmatrani servis zadovoljava jedan od kriterijuma sprečavanja, kaže se da zavisnost nije prošla i provera daljih zavisnosti se obustavlja
4. ako tekući status servisa od koga zavisi posmatrani servis ne zadovoljava ni jedan od kriterijuma sprečavanja, kaže se da je zavisnost prošla i provera sledećih zavisnosti (ako ih ima) će se nastaviti.

Ovaj ciklus provera zavisnosti se nastavlja dok se ne provere sve zavisnosti ili do prve neuspele provere.

Zavisnosti izvršavanja servisa

Ako se svi testovi zavisnosti izvršavanja za određeni servis uspešno prođu, Nagios će izvršiti proveru servisa kao što je normalno predviđeno. Ako jedan od testova zavisnosti izvršavanja servisa ne prođe, Nagios će privremeno sprečiti izvršavanje provere ovog (zavisnog) servisa. U nekom trenutku u budućnosti testovi zavisnosti izvršavanja servisa će se uspešno proći. Ako se to dogodi, Nagios će ponovo započeti proveru servisa kao što je i ranije normalno činio.

U prethodno pomenutom primeru, servis E ne bi prošao test zavisnosti izvršavanja ako bi servis B imao WARNING ili UNKNOWN status. Ako bi to bio

slučaj, provera servisa se ne bi izvršila i bila bi preraspoređena za potencijalno izvršavanje u nekom kasnijem trenutku.

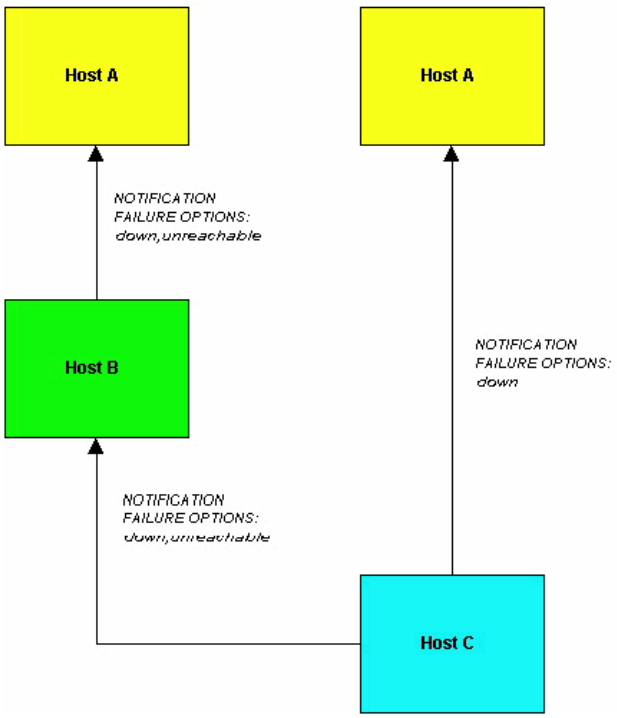
Zavisnosti obaveštavanja o servisu

Kao što je pomenuto ranije, zavisnosti servisa se ne nasleđuju. U primeru se takođe može videti da servis F zavisi od servisa E. Međutim on ne nasleđuje automatski zavisnost servisa E od servisa B i C. Da bi servis F učinili zavisnim od servisa C moramo dodati novu definiciju zavisnosti servisa. Pošto ne postoji definicija zavisnosti od servisa B, servis F neće zavisiti od servisa B. U nekim slučajevima nedostatak svojstva nasleđivanja će podrazumevati dodatne definicije zavisnosti u konfiguracionoj datoteci, ali ovaj nedostatak je ustvari prednost jer zavisnost čini fleksibilnijim. Na primer, u pomenutom primeru mogli smo da imamo dobar razlog da ne činimo servis F zavisnim od servisa B. Ako bi zavisnost bila automatski nasleđena, to ne bi bilo moguće.

Zavisnosti hostova

Zavisnosti hostova funkcionišu na isti način kao zavisnosti servisa. Razlika je samo u tome što se radi o hostovima. Druga razlika je što zavisnosti hostova vrše samo potiskivanje obaveštenja, ali ne i sprečavanje izvršavanja provera hostova.

Slika 11. ilustruje primer logičkog prikaza zavisnosti hostova.



Slika 11. Zavisnosti hostova

Na ovoj slici, definicije zavisnosti za host C bi izgledala na sledeći način:

```
define hostdependency {
      host_name                       Host A
      dependent_host_name             Host C
      notification_failure_criteria   d
      }

define hostdependency {
```

```
host_name                       Host B
dependent_host_name             Host C
notification_failure_criteria   d,u
}
```

Kao kod zavisnosti servisa, zavisnosti hostova se ne nasleđuju. Može se videti da host C ne nasleđuje zavisnosti hosta B. Da bi host C zavisio od hosta A, mora se dodati nova definicija zavisnosti.

Zavisnosti obaveštenja hostu funkcionišu slično kao zavisnosti servisa. Ako se prođu svi testovi zavisnosti za određeni host, Nagios normalno šalje obaveštenje o hostu. Ako se jedan od kriterijuma ne zadovolji, privremeno se potiskuju obaveštenja o zavisnom hostu. U nekom trenutku u budućnosti testovi zavisnosti će možda biti zadovoljeni. Ako se to dogodi, ponovo će početi normalno obaveštavanje o ovom hostu.

Praćenje stanja (state stalking)

Praćenje stanja je pogodnost koju većina korisnika verovatno neće koristiti. Ako se uključi, ona će omogućiti logovanje promena kroz koje prolazi servis ili host čak i kada se stanje servisa ili hosta ne menja. Tada Nagios pažljivo posmatra dati servis i loguje svaku promenu koju vidi. Kao što se može pretpostaviti, ovo može biti od velike koristi kod naknadne analize log-a.

Princip rada

Pod normalnim okolnostima, rezultati provera hosta ili servisa se loguju samo pri promeni stanja od poslednje provere. Postoji nekoliko izuzetaka od ovog pravila, ali ono važi u većini slučajeva.

Ako se omogući praćenje jednog ili više stanja određenog hosta ili servisa, Nagios će logovati rezultate provera ako se izlaz provere razlikuje od izlaza predhodne provere. Pogledajte sledeći primer od 8 uzastopnih provera servisa:

Service Check	Service Status	Service Check Output
x	OK	RAID array optimal
x+1	OK	RAID array optimal
x+2	WARNING	RAID array degraded (1 drive bad, 1 hot spare rebuilding)
x+3	CRITICAL	RAID array degraded (2 drives bad, 1 host spare online, 1 hot spare rebuilding)
x+4	CRITICAL	RAID array degraded (3 drives bad, 2 hot spares online)
x+5	CRITICAL	RAID array failed
x+6	CRITICAL	RAID array failed
x+7	CRITICAL	RAID array failed

Iz date sekvence provera vidi se da bi normalno videli samo 2 unosa u logu za ovu katastrofu. Prvi bi se dogodilo pri proveri servisa x+2 kada se staus menja iz OK u WARNING (prelazak iz HARD u SOFT stanje). Drugi unos bi se zabeležio kod x+3 provere servisa kada mu se status menja iz WARNING u CRITICAL.

Bez obzira na razlog, korisnik bi svakako bilo od koristi da ima kompletnu istoriju ovakve katastrofe u svojim log datotekama. Ako ništa drugo, možda da bi lakše objasnio svom menadžeru koliko brzo je situacija izmakla kontroli i sl.

Ako bi se opcija praćenja stanja ovog servisa omogućila za CRITICAL stanja, u log datoteci bi imali logovane događaje kod x+4 i x+5 provere uz predhodne x+2 i x+3.

Sličan primer bio bi praćenje stanja servisa koji proverava status HTTP servera. Ako `check_http` plagin prvo vrati WARNING status zbog greške 404, a potom vrati ista stanja od naknadnih provera zbog nepronalaženja određenog teksta (*pattern*) u vraćenoj strani, web administrator bi možda želeo da zna za to. U slučaju kada ova opcija nije uključena za WARNING stanja servisa, bilo bi logovano samo prvo stanje (greška 404) dok o ostalim ne bi bilo nikakvih tragova kao ni to da problem nije u greški 404, već u nedostajućem *pattern*-u u vraćenoj HTTP stranici.

Da li treba omogućiti praćenje stanja

Prvo treba oceniti da li zaista postoji potrebu da se analiziraju arhivirani logovani podaci u cilju pronalaženja pravog uzroka problema. Obično, uvek se radi o nekolicini hostova i servisa. Nikada za sve. Takođe, može se zahtevati praćenje samo nekih stanja hostova ili servisa. Na primer, može se omogućiti praćenje samo WARNING i CRITICAL stanja servisa, a ne i OK i UNKNOWN stanja.

Odluka da li omogućiti praćenje statusa odeđenog hosta ili servisa takođe zavisi od plagina koji se koristi za njihovu proveru. Ako plagin uvek vraća isti tekstualni izlaz za određeno stanje, nema razloga da se uključuje praćenje stanja ovakvog servisa.

Konfigurisanje praćenja

Ova opcija se omogućava pomoću `stalking_options` direktive u definicijama hostova i servisa. Trenutno, ova direktiva je podržana samo u konfiguracionim datotekama u formatu baziranom na obrascima (*template based method*).

Potencijalni problemi

Trebalo bi se čuvati potencijalnih zamki koje dolaze sa uključivanjem praćenja stanja. Ovo je povezano sa funkcijama sadržanim u različitim CGI skriptovima (*histogram*, *alertsummary*, itd). Zbog praćenja stanja koje izaziva dodatno logovanje *alert* stanja, izveštaji koje generišu ovi CGI skriptovi će rezultovati nerealno povećanim brojem alert stanja.

Generalno, uključivanje opcije praćenja stanja se ne preporučuje pre nego što se dobro promisli o njegovim posledicama. Međutim, ako je tako nešto ipak potrebno, ono je tu da pomogne.

Event hendleri

Event hendleri su opcione komande koje se izvršavaju kad god se dogodi promena stanja hosta ili servisa. Očigledan primer upotrebe *event* hendlera (naročito kod servisa) je mogućnost proaktivnog rešavanja problema pre nego što iko o tome bude obavešten. Druga potencijalna upotreba *event* hendlera mogla bi da bude logovanje događaja (*event*) hostova ili servisa u centralnu eksternu bazu podataka.

Tipovi *event* hendlera

Postoje dva glavna tipa *event* hendlera koja se mogu definisati - *event* hendleri servisa i *event* hendleri hostova. Komande *event* hendlera se opciono definišu u svakoj definiciji hosta ili servisa. Pošto su ovi hendleri pridruženi samo određenim servisima ili hostovima možemo ih nazvati "lokalnim" *event* hendlerima. Ako je lokali *event* hendler definisan za host ili servis, on će se izvršavati kad god taj servis ili host promeni svoje stanje.

Takođe, mogu se definisati i "globalni" *event* hendleri koji se izvršavaju pri svakoj promeni stanja bilo kog hosta ili servisa. Oni se definišu `global_host_event_handler` i `global_service_event_handler` direktivama u glavnoj konfiguracionoj datoteci. Globalni hendleri se uvek izvršavaju neposredno pre lokalnih.

Kada se izvršavaju komande *event* hendlera

Komande *event* hendlera se izvršavaju kada servis ili host:

- nađe se u SOFT stanju greške
- po prvi put uđe u HARD stanje greške
- oporavi se iz SOFT ili HARD stanja greške

Redosled izvršavanja *event* hendlera

Globalni hendleri se izvršavaju pre bilo kog lokalnog hendlera definisanog za odeđeni host ili servis.

Pisanje komandi *event* hendlera

U većini slučajeva, komande *event* hendlera će biti *shell* ili *perl* skriptovi. U najmanju ruku, skriptovi bi kao argumente trebalo da koriste sledeće makroe

- makroi *event* hendlera servisa: `$SERVICESTATE$, $STATETYPE$, $SERVICEATTEMPT$`
- makroi *event* hendlera hostova: `$HOSTSTATE$, $STATETYPE$, $HOSTATTEMPT$`

Skriptovi bi trebalo da ispitaju vrednosti zadatih argumenata i na osnovu njih preduzmu neophodne akcije. Najbolji način za razumevanje principa rada *event* hendlera je da se pogleda neki primer koji se može naći u `/eventhandlers` poddirektorijumu distribucije Nagiosa. Neki od ovih primera prikazuju upotrebu eksternih komandi pomoću kojih se implementiraju redundantni nadgledajući serveri.

Privilegije korisnika koji izdaje komande *event* hendlera

Svaka komanda *event* hendlera koja se zada izvršavaće se sa istim dozvolama koje poseduje korisnik pod čijim imenom se izvršava Nagios proces. Ovo predstavlja problem onim skriptovima koji pokušavaju da kontrolišu sistemske servise za čiji su pristup obično potrebne administratorske privilegije.

U idealnom slučaju trebalo bi odrediti sve tipove *event* hendlera koji će se implementirati tako da se Nagios korisniku definišu samo neophodne dozvole kako bi mogao da izvršava sve neophodne sistemske komande. Da bi se ovo obezbedilo može da se koristi `sudo` komanda. Način korišćenja `sudo` komande izlazi iz okvira ovog rada, ali se čitanje njene dokumentacije svakako preporučuje.

Debagovanje komandi *event* hendlera

Kada se vrši debagovanje neke komande *event* hendlera, toplo se preporučuje omogućavanje opcije logovanja ponovnih pokušaja provera servisa, provera hostova i pokretanje hendlera pomoću `service_retries`, `host_retries` i `event_handler_commands` direktiva u glavnoj konfiguracionoj datoteci. Ovo omogućuje precizan uvid u vreme kada se i zašto izvršavaju određene komande *event* hendlera.

Kada se proces debagovanja završi, poželjno je da se onemoguće pomenute opcije jer vrlo brzo mogu da uvećaju log datoteku, što opet ne mora da bude problem ako je uključena rotacija logova.

Primer *event* hendlera servisa

Prikazani primer predpostavlja da se nadgleda HTTP server na lokalnom hostu na kome se izvršava Nagios. Dalje, pretpostavlja se da je u definiciji HTTP servisa definisana komanda event hendlera, recimo po imenu `restart-httpd`. Takođe, neka je vrednost `max_check_attempts` direktive postavljena na 4 ili više (servis se ponovno proverava 4 ili više puta pre nego što se proglasi da zaista ima problem). Primer definicije servisa bi mogao da izgleda ovako:

```
define service{
      host_name                       gandalf
      service_description             HTTP
      max_check_attempts              4
      event_handler                   restart-httpd
      ...ostale direktive u definiciji servisa...
      }
```

Kada se servis definiše sa određenim event hendlerom, potrebno je taj *event* hendler definisati kao komandu u konfiguracionoj datoteci `checkcommands.cfg`. Treba obratiti pažnju na makroe u komandnoj liniji kojim se prenose potrebni argumenti skriptu *event* hendlera.

```
define command{
      command_name  restart-httpd
      command_line  /usr/nagios/libexec/eventhandlers/restart-httpd
      $SERVICESTATE$ $STATETYPE$ $SERVICEATTEMPT$
}
```

U daljem tekstu dat je primer skripta *event* hendlera `restart-httpd` koji se nalazi u direktorijumu `/usr/nagios/libexec/eventhandlers/`

```
#!/bin/sh
## Event handler skript koji restartuje Apache HTTP server na lokalnom
## računaru. Ovaj skript će restartovati Apache server samo ako se posle
## 3 uzastopne provere (u SOFT stanju) vrati non-OK status HTTP servisa
## ili ako servis na neki način uspe da uđe u HARD stanje greške.

# Koji je status HTTP servisa?
case "$1" in
OK)
      # Servis se upravo opravio, te ne preduzimamo ništa...
      ;;
WARNING)
      # Ne brine nas ni status upozorenja, pošto servis verovatno još
      # uvek funkcioniše...
```

```
        ;;
UNKNOWN)
        # Ne znamo zašto bi servis mogao da se nađe u ovom statusu, pa ne
        # preduzimamo ništa...
        ;;
CRITICAL)
        # izgleda da HTTP servis ima problem! – probaćemo da restartujemo
        # Apache server...

                # Da li je servis u  SOFT ili HARD stanju?
        case "$2" in
        SOFT)
                # Servis je u SOFT stanju, što znači da Nagios upravo
                # pokušava da izvrši ponovne provere servisa, pre nego što
                # zaključi da servis zaista ima problem, prebaci ga u HARD
                # stanje i obavesti nadležne kontakte...

                        # Koji pokušaj ponovne provere je u toku? Ne želimo da
                        # restartujemo Apache server tokom prve provere, jer
                        # je uzbuna možda lažna!
                case "$3" in
                        # Čekamo dok Nagios ne izvrši tri ponovne provere pre
                        # nego što krenemo da restartujemo web server. Ako i
                        # četvrta provera vrati non-OK status (dakle, pošto
                        # smo već restartovali Apache server) stanje servisa
                        # prelazi u HARD stanje i kontakti će biti obavešteni
                        # o ovom problemu. Međutim, nadamo se da će ovo
                        # uspešno restartovati server, tako da će četvrta
                        # provera rezultovati SOFT oporavkom. Ako se to
                        # dogodi, niko neće biti obavešten jer je problem
                        # rešen!
                3)
                        echo -n "Restartovanje HTTP servisa..."
                                #Poziv init skripta koji restartuje web server
                        /etc/rc.d/init.d/httpd restart
                ;;
                esac
        ;;
                # HTTP servis je nekako uspeo da uđe u HARD stanje greške bez
                # pokušaja restartovanja. Problem bi trebalo da je rešen
                # pomoću komande iznad, ali iz nekog razloga to nije slučaj.
                # Dajemo mu poslednju šansu!
                # Napomena: Kontakti su već obavešteni o problemu sa servisom
                # u ovoj tački (osim ako za ovaj servis nije onemogućeno
                # obaveštavanje)
        HARD)
                echo -n "Restartovanje HTTP servera..."
                        # Poziv init skripta da restartuje HTTPD server
                /etc/rc.d/init.d/httpd restart
        ;;
        esac
;;
esac
exit 0
```

Ovaj skript će pokušati da restartuje HTTP server na lokalnoj mašini na dva različita načina: pošto HTTP servis bude ponovno proveren po treći put (u SOFT stanju greške) i nakon što servis pređe u HARD stanje. HARD stanje ne bi smelo da se dogodi, pošto skript restartuje servis još dok je u SOFT stanju (npr. prilikom treće ponovne provere), ali je tako ostavljeno za svaki slučaj, ako ipak na neki način servis dospe u njega.

Treba još primetiti da će se *event* hendler servisa izvršiti samo prilikom prvog prelaska servisa u HARD stanje. Ovim se spečava da Nagios uzaludno izvršava skript restartovanja HTTP servera u slučaju da oporavak servisa nije

moguć na ovaj način, a servis je u HARD stanju.

Eksterne komande

Nagios može da izvršava komande eksternih aplikacija (uključujući i CGI skriptove) i da na različite načine menja funkcije nadgledanja na osnovu primljenih komandi.

Omogućavanje eksternih komandi

Po *default* konfiguraciji, Nagios niti proverava, niti izvršava eksterne komande. Ako je to potrebno, u glavnoj konfiguracionoj datoteci treba:

- omogućiti `check_external_commands` direktivu,
- zadati odgovarajuću vrednost u `command_check_interval` direktivi,
- zadati lokaciju komandnog datoteke direktivom `command_file` i
- postaviti odgovarajuće dozvole nad direktorijumom koji sadrži datoteku sa eksternim komandama.

Kada Nagios proverava eksterne komande

- U regularnim intervalima definisanim `command_check_interval` direktivom u glavnoj konfiguracionoj datoteci.
- Odmah po izvršavanju *event* hendlera. Ovo je dodatna provera regularnom ciklusu provera eksternih komandi koja se vrši da bi se potencijalna akcija odmah preduzela, ako *event* hendler zada komandu Nagiosu.

Upotreba eksternih komandi

Eksterne komande se mogu koristiti za završavanje različitih zadataka u vreme izvršavanja Nagiosa. Primer mogu biti privremeno onemogućavanje obaveštenja o servisima i hostovima, privremeno onemogućavanje provera servisa, forsiranje provera servisa u datom trenutku, dodavanje komentara hostovima i servisima itd.

Format komande

Eksterne komande u komandnoj datoteci upisuju se u sledećem formatu.

```
[vreme] ime_komande;argumenti_komande
```

gde je vreme izraženo u `time_t` formatu i predstavlja trenutak u kom eksterna aplikacija ili CGI skript izvršavaju eksternu komandu iz datoteke eksternih komandi. Kompletna tabela svih eksternih komandi ugrađenih u Nagios sa opisom njihovih argumenata data je u Dodatku E

Nagiosovi plaginovi

Već u pregledu mogućnosti jezgra Nagiosa može se naslututi da se jezgro u potpunosti oslanja na funkcionalnost plaginova i eventualno još nekih eksternih aplikacija. Drugim rečima, Nagios je beskoristan bez svojih plaginova i oni se ne mogu izbeći.

Osnovni zadatak svakog plagina je da proveri tekući status određenog hosta ili servisa i vrati jezgru formatirane rezultate provere. Svaki plagin je praktično nezavistan programski modul koji se može pokretati iz komandne linije. Obzirom da se pod pojam servisa može podvesti mnogo toga, i plaginovi, kao programi specijalizovani za provere statusa tih mnogobrojnih servisa, mogu da se razlikuju po implementaciji, kompleksnosti ili platformi za koju su ciljno razvijani.

Implementacija

Većina plaginova je napisana u programskim jezicima C i C++, a dobar deo ostalih plaginova je razvijen u Perl-u ili *shell* skriptu. Plaginovi napisani u C/C++ programskom jeziku su tipično brzi i jako pogodni za nadgledanje ogromnog broja servisa. S druge strane, *shell* i Perl plaginovi se za oko red veličine sporije izvršavaju jer su interpretirani, ali razvijanje i dopunjavanje ovakvih plaginova je daleko jednostavnije i brže. Kod značajnijeg korišćenja plaginova napisanih u Perlu, oni se donekle mogu ubrzati upotrebom *Embedded* Perl interpretera.

Verzije plaginova

Standardni plaginovi za Nagios su javno dostupni na zvaničnom web sajtu Nagios projekta. U ovom trenutku, u zvaničnoj distribuciji je uključeno oko pedesetak plaginova koji se zbog lakšeg instaliranja objavljuju u obliku jedinstvene *tarball* arhive sa izvornim kodovima, uputstvom za instalaciju i instalacionim skriptom. Svaki plagin program je označen brojem svoje verzije, ali zbog lakšeg praćenja, za identifikaciju plaginova se najčešće koristi jedinstvena verzija paketa u okviru kog je plagin objavljen. Trenutno, aktuelna je beta verzija plaginova 1.3.1, i 1.4.0 u alfa verziji.

Neretko, lista i format argumenata koje primaju određeni plaginovi se menjaju iz verzije u verziju. Stoga je poznavanje verzije korišćenih plaginova jako važno jer promena verzije plaginova može da iziskuje odgovarajuće ispravke u konfiguracionim datotekama.

Do sada je bilo reči samo o zvaničnim plaginovima čiji je programski kod u potpunosti pregledan i testiran od strane autora Nagiosa. Pored njih, na internetu se mogu naći i brojni drugi plaginovi za proveru različitih servisa koje su za svoje specifične potrebe razvili brojni mrežni administratori i podelili ih sa *open source* zajednicom. Zbirka sa više od šezdeset ovakvih plaginova koji nisu usaglašeni po broju verzije, distribuira se zajedno sa zvaničnim plaginovima u `/contrib` poddirektorijumu distribucije plaginova. Većina njih je napisana u Perlu, i sa malo ili bez imalo prepravki, spremna je za upotrebu.

Trenutno postojeći *ebuild* u *Gentoo Portage* stablu za Nagios koristi plaginove verzije 1.3.1. te su oni i korišćeni u implementaciji Nagiosa na fakultetskoj mreži (videti glavu Instalacija).

Dokumentacija

Dokumentacija o načinu korišćenja individualnih plaginova se ne dobija uz Nagios program. Ona dolazi sa plaginovima. Svi plaginovi koji su napisani u skladu sa minimalnim razvojnim smernicama ovog projekta uključuju internu dokumentaciju. Dokumentacija se može čitati izvršavanjem plagina sa `-h` opcijom. Ako ova opcija ne radi, radi se o bagu.

Na primer, informacije o tome kakve i koje argumente i opcije prima plagin za provere servisa pomoću SNMP protokola, *`check_snmp`*, kao i o njegovom autoru i licenci, mogu se dobiti na sledeći način:

```
/usr/nagios/libexec/check_snmp -h
```

U slučaju da ova dokumentacija nije dovoljno jasna, uvek je dostupan otvoren kod plaginova u kome se tačno može videti njegov princip rada.

Korišćeni plaginovi

Namera ovog rada bila je pre da se paket Nagios iskoristi za nadzor računarske mreže Elektrotehničkog fakulteta, nego da se ispitaju sva svojstva i osobine koje ispoljavaju brojni plaginovi za Nagios. Iako ima veoma mnogo interesantnih plaginova za Nagios, u ovom poglavlju dat je detaljniji pregled samo onih plaginova koje sam zaista koristio za nadgledanje fakultetske mreže.

Uz opis svakog plagina dati su format i primer upotrebe iz komandne linije. Obeležavanje opcionih flegova razlikuje se od obaveznih po tome što su prvi dati u uglastim zagradama. Na primer, sa "`[-t <vreme_izvš>]`" je označen jedan neobavezni fleg. Izbor između više opcija obeležen je uspravnom crtom, "`|`". Zbog veće preglednosti, u svim primerima korišćenja komandi plaginova podrazumevano je da se one na ovaj način mogu pokretati iz njihovog matičnog direktorijuma `/usr/nagios/libexec/`.

Da bih izbegao nepotrebno ponavljanje, odmah ću objasniti sve flegove koji su zajednički za sve ili bar većinu plaginova.

Flegom `-H` zadaje se IP adresa hosta koji pruža proveravani servis. Opciono, umesto IP adrese može se zadati ime hosta. Međutim, ovakva praksa nije dobra budući da plagin pre provere mora prvo DNS upitom da razreši ime hosta u njegovu IP adresu. Ovim se nepotrebno opterećuju DNS serveri i povećava mrežni saobraćaj, a provera servisa postaje zavisna od statusa DNS servera i sporije se izvršava. Host se kao argument pojavljuje u svim plaginovima osim onih koji vrše provere servisa koji se hostuju na lokalnom računaru.

Maksimalno dozvoljeno vreme izvršavanja plagina kontroliše se flegom `-t`. Ako se ovo vreme prekorači, plagin obustavlja proveru i vraća status CRITICAL sa tipičnim tekstualnim izlazom "*Plugin time out*". Izostavljanjem ovog parametra, plagin koristi neku podrazumevanu vrednost, tipično 10 sekundi.

Minimalna dokumentacija i opis svih opcija plagina dobija se pozivanjem sa flegom `-h`, dok se verzija plagina i verzija plagin distribucije u okviru koje je plagin objavljen dobija pomoću flega `-V`. Takođe je uobičajeno da plaginovi primaju fleg `-v` (ili `--verbose`) kojim se zahteva detaljan tekstualni izlaz plagina. Ova opcija se može koristiti samo iz komandne linije, obično kod debagovanja komandi provera.

check_dns

Format: `./check_dns -H <ime_hosta> [-s <IP_adresa>] [-a <IP_adresa] [-A <IP_adresa>] [-t <vreme_izvš>]`

Primeri:
```
./check_dns -H galeb.etf.bg.ac.yu
./check_dns -H galeb.etf.bg.ac.yu –a 147.91.8.64
./check_dns -H 147.91.8.64 –a galeb.etf.bg.ac.yu
–A 147.91.8.6
```

Ovo je jedan od dva plagina koje sam koristio za nadgledanje servisa koji pružaju DNS serveri. Plagin je baziran na programu *nslookup* koji za zadato ime hosta vraća IP adresu. Ako u mreži postoji sekundarni DNS server, on se opciono može definisati pomoću `-s` flega. Ako se pak ne definiše ni jedan DNS server, plagin koristi server ili servere definisane na nadgledajućem serveru, tipično u `/etc/resolv.conf`.

Plagin vraća OK status ako DNS uspešno odgovori na upit, status WARNING u slučaju nepotpunog odgovora na upit, dok se u slučaju greške ili neodogovaranja vraća CRITICAL status.

U prvom primeru *check_dns* plagin proverava da li DNS server uredno odgovara na upit. U drugom primeru se proverava i to da li DNS ispravno rezolvuje ime `galeb.etf.bg.ac.yu` (moglo je i samo `galeb`) u njegovu IP adresu 147.91.8.64. U oba slučaja se koristi jedan od DNS servera definisanih na sistemu. U trećem primeru, zahteva se da na upit odgovori specificirani DNS server, koji je u ovom slučaju 147.91.8.6 (`ns.etf.bg.ac.yu`).

check_dig

Format: `./check_dig -H <IP_adresa> -l <ime_hosta> [-t <vreme_izvš>] [-v]`

Primeri:
```
./check_dns -H 147.91.8.6 -l www.google.com
./check_dns -H 147.91.8.62 –l www.gentoo.org
```

Plagin *check_dig* takođe služi za proveru ispravnosti rada jednog od najvažnijih servisa na mreži, servisa koji pružaju DNS serveri. Za razliku od prethodnog, baziran je na programu `dig`. Plagin kao argumente uzima IP adresu DNS servera i ime računara za koje DNS treba da vrati IP adresu.

Suštinska razlika u odnosu na prethodni plagin je u tome što se ovim plaginom direktno obraćamo DNS serveru čija je adresa eksplicitno definisana.

U prvom primeru *check_dig* plagin proverava da li DNS server 147.91.8.6 ispravno rezolvuje ime web servera najvećeg svetskog pretraživača u jednu od njegovih IP adresa. U drugom primeru, proverava se isto to (traži se IP adresa `www.gentoo.org`) pomoću drugog fakultetskog DNS servera.

check_disk

Format: `./check_disk -w <w_prag> -c <c_prag> [-p <putanja> | -x <particija>] [-t <vreme_izvš>] [-m] [-e] [-v]`

Primeri:
```
./check_disk -w 20% -c 10% -p /dev/sda3
./check_disk -w 409600 -c 204800 -p /dev/sda5
```

Plagin *check_disk* služi za proveru popunjenosti svih montiranih particija ili tačno zadate particije hard diska na lokalnom sistemu. Plaginu se ili u procentima ili u kilobajtima zadaju vrednosti pragova za status WARNING i

CRITICAL. Vrednost pragova može biti zadata u procentima ili celobrojnim vrednostima, kada označavaju kapacitet u kilobajtima. Kada slobodan memorijski prostor padne ispod definisanih pragova, plagin će podići odgovarajući alarm. Ako se ne navede nijedna particija, plagin će proveriti slobodan prostor na svim montiranim particijama. Opcioni flegovi –x i –e pružaju mogućnost isključivanja određene particije i prikazivanje samo particija sa greškama, respektivno.

Ovo je jedan od plaginova koji zahteva da bude instaliran na računaru čije stanje hard diskova proverava. Budući da sam posedovao nalog samo na serverima `gandalf` i `galeb`, imao sam mogućnost provere stanja hard diskova samo na ovim računarima. Na nadgledajućem serveru, nadgledano je svih šest particija. Swap particija je nadgledana posebnim plaginom koji će kasnije biti objašnjen. Nadgledanje stanja particija na hard diskovima servera `galeb`, vršeno je pomoću indirektnih provera. Naravno, zato je na njemu prethodno bilo potrebno da se instaliraju plaginovi i posrednički program `nrpe`.

check_dummy

Format: `./check_dummy <celobr_status_kod>`

Primeri:
```
./check_dummy 0
./check_dummy 2
```

Ovaj plagin jednostavno vraća status koji odgovara brojnoj vrednosti njegovog jedinog argumenta. Drugim rečima, *`check_dummy`* vrši veštačku proveru i vraća rezulat koji mu se zada. Uporedna tabela brojnih vrednosti i odgovarajućih statusa je sledeća:

br.	status
0	OK
1	WARNING
2	CRITICAL
3	UNKNOWN

Ovaj plagin definitivno nije neophodan, ali može koristiti za elegantno rešavanje nekih problema. U mojoj implemetaciji Nagiosa, ovaj plagin je našao primenu kod definisanja virtuelnih hostova (odatle njihovo ime, *dummy hosts*) čija je svrha objašnjena na strani 11. Formalno, definicija virtuelnih hostova zahteva da se definiše komande provere hosta kao i da se tom hostu pridruži bar jedan servis. Kako je status ovakvih hostova uvek UP, jedno od rešenja za komandu provere ovakvog hosta bilo je korišćenje *`check_ping`* plagina pri čemu bi kao adresa hosta bila navedena *loopback* IP adresa 127.0.0.1. Očigledno, *`check_dummy`* plagin je odabran kao mnogo jednostavnije rešenje.

check_ftp

Format:
```
./check_ftp -H <IP_adresa> -p <port> [-w <w_vreme>]
[-c <c_vreme>] [-s <string>] [-e <string>] [-q <string>]
[-r <ok|warn|crit>] [-m <br_bajtova>] [-d <vreme>]
[-t vreme_izvrš] [-v]
```

Primer: `./check_ftp -H 147.91.8.64`

Plagin *`check_ftp`* služi za proveru ispravnosti rada FTP servera na mreži. U osnovi ovog plagina krije se *`check_tcp`* plagin. Iz formata komande se može videti da je podržano bogatstvo opcija. Omogućeno je definisanje pragova za dozvoljena

vremena odziva FTP servera u sekundama, koji za dato prekoračenje podižu odgovarajući alarm. Takođe, podržano je i slanje definisanog stringa serveru i upoređivanje odgovora sa očekivanim stringom odgovora. Opcionim flegom `-d` može se definisati vreme čekanja u sekundama (*delay*) od trenutka slanja stringa do početka poliranja odziva. Opcija `-m` definiše broj primljenih bajtova nakon čega se smatra da je povera bila uspešna. Uključivanjem flega `-r` omogućeno je vraćanje OK statusa servisa čak i za slučaj kada je očekivano da server odbije pokušaj konekcije (slučaj `-r ok`).

Moja upotreba ovog plagina se svodila na prosto otvaranje i zatvaranje konekcije na fakultetskim serverima na kojima je podignut FTP servis, kao u navedenom primeru. U ovom slučaju, plagin vraća OK status kada se sa serverom uspostavi veza, WARNING kada server odbije konekciju i CRITICAL u slučaju da server uopšte ne odgovori.

check_hpjd

Format: `./check_hpjd -H <IP_adresa> [-C <community>]`

Primer: `./check_hpjd -H hplaser4v.etf.bg.ac.yu -C public`

Plagin *`check_hpjd`* je napisan za proveru statusa veoma rasprostranjenih *Hewlett Packard*-ovih mrežnih laserskih štampača sa *JetDirect* mrežnom kartom. Plagin se zasniva na SNMP protokolu a od argumenata prima *community* string koji je podešen na štampaču. Ukoliko se plagin pozove bez argumenata, koristi se *default community* string "public".

Plagin vraća status OK kada je štampač operativan i dostupan preko mreže. U slučaju neke od niza mogućih grešaka štampača (*Out of Paper*, *Power Save On*, *Peripheral Error*, *Intervention Required*, *Toner Low*, *Insufficient Memory*, *Output Tray is Full*, *Unknown Paper Error* i sl.), plagin vraća WARNING status zajedno sa tekstualnom porukom koja sadrži informacije o uzroku greške. Ukoliko je štampač nedostupan, plagin vraća status CRITICAL.

U datom primeru, prikazana je upotreba ovog specijalizovanog plagina na jednom fakultetskom mrežnom laserskom štampaču sa *JetDirect* karticom.

check_load

Format: `./check_load -w <opterećenje_1min,opterećenje_5min,opterećenje_15min>`
`-c <opterećenje_1min,opterećenje_5min,opterećenje_15min>`

Primer: `./check_load -w 15,10,5 -c 25,20,15`

Plagin *`check_load`* vrši proveru prosečnog opterećenja procesora tokom proteklih 1, 5 i 15 minuta. Vrednosti koje vraća plagin su po formatu iste kao vredosti koje vraćaju standardni Linux programski alati *`uptime`* i *`w`*.

Flegovima `-w` i `-c` definišu se pragovi za u formatu tripleta zapetom razdvojenih realnih brojeva. Po prekoračenju bilo kog od prva tri praga, vraća se status WARNING, a po prekoračenju bilo kog od druga tri praga, vraća se status CRITICAL.

Ovo je još jedan iz grupe plaginova kojim se mogu pratiti samo stanja privatnih lokalnih servisa na računaru na kom su instalirani. Potpuno analogno sa *`check_disk`* plaginom, i ovim plaginom sam vršio nadgledanje opterećenja procesora samo na serverima `gandalf` i `galeb` Na prvom serveru direktno, na drugom indirektno pomoću agentskog programa *`nrpe`*.

check_nagios

Format: `./check_nagios -F <status_datoteka> -e <br_minuta> -C <procesni_string>`

Primer: `./check_nagios –F /var/nagios/status.log -e 5 -C /usr/nagios/bin/nagios`

Plagin *check_nagios* proverava da li se Nagios proces izvršava. Provera se vrši tako što se plagin uverava da statusna datoteka Nagiosa, `status.log`, nije starija od zadatog broja minuta. Ova datoteka je indikator izvršavanja Nagiosa jer u normalnim okolnostima postoji samo u toku njegovog izvršavanja. Ako se nekom greškom dogodi da se statusna datoteka ne izbriše po zaustavljanju Nagiosa, plagin proverava i njegovu starost. Da bi bili potpuno "sigurni" da se Nagios izvršava, plagin koristi standardni Linux program *ps* koji izlistava sve tekuće procese na sistemu. Ako u izlazu programa *ps* pronađe očekivani string komande Nagiosa, plagin će zaključiti da se Nagios proces izvršava.

Flegom `–F` zadaje se ime Nagiosove statusne datoteke sa njenom apsolutnom putanjom. Flegom `-e` zadaje se prag starosti statusne datoteke. Na kraju, flegom `-C` definiše se očekivani string komande koji se javlja u izlazu programa *ps*. Ako se bilo koji od ova dva kriterijuma ne zadovolje, plagin vraća CRITICAL status. Ukoliko se na standardnom izlazu za greške (*stderr*) pojavi bilo kakav izlaz, plagin će vratiti WARNING status.

Plagin je karakterističan po tome što ga koriste CGI skriptovi. Komanda provere statusa Nagios procesa se opciono definiše *nagios_check_command* direktivom u konfiguracionoj datoteci CGI skriptova. Međutim, da bi *command* CGI skript ispravno funkcionisao, ova komanda provere mora da se definiše i da ima OK status. Pored ove namene, plagin se koristi kod distribuiranog nadgledanja ili nadgledanja redundantnim Nagios serverima. Primera radi, kod redundantnog nadgledanja, uz ovu proveru može da se definiše *event* hendler koji kada provera procesa na glavnom nadgledajućem (*master*) serveru vrati CRITICAL status, aktivira provere na redundantnom (*slave*) Nagios serveru.

U mojoj implemetaciji Nagiosa, ovaj plagin sam koristio samo za potrebe CGI skriptova. Pri tom sam primetio da je zbog provere stringa plagin jako osetljiv na način na koji je pokrenut. Na primer, kada bi Nagios bio pokrenut ručno iz njegovog `/bin` direktorijuma, a plagin pokretan sa argumentima kao u datom primeru, plagin bi vraćao CRITICAL status i onemogućavao rad *command* CGI skripta, iako se proces normalno izvršavao. Tada, način da se utvrdi ispravan string koji treba zadati plaginu bio je izlaz sledeće komande:

```
ps aux | grep nagios
```

check_nrpe

Format: `./check_nrpe -H <IP_adresa> [-p <port>] -c <komanda> [-to <vreme_izvrš>]`

Primer: `./check_nrpe –H 147.91.8.64 –c check_ping_el`

Plagin *check_nrpe* je zapravo klijentski deo `nrpe` dodatka za Nagios koji se koristi za pokretanje Nagiosovih plaginova na udaljenim računarima. Ovaj program je praktično jedan od načina izvršavanja indirektnog provera koje su detaljno opisane na str. 29 – 31.

Kada Nagios izda komandu provere pomoću ovog plagina, plagin prvo pokušava da preko određenog porta uspostavi konekciju sa `nrpe` *daemon*-om na

definisanom udaljenom računaru i izdaje mu zahtev za izvršavanjem određene komande provere. Demonski program `nrpe` izvršava zadatu komandu i rezultate provere vraća `check_nrpe` plaginu, da bi ih on konačno podneo Nagiosu. Ovim se omogućava da Nagios izvršava provere pomoću plaginova na udaljenim hostovima kao da su lokalni.

Plagin za konekciju koristi podrazumevani port 5666, ali pomoću flega `-p` dozvoljeno je opciono definisanje proizvoljnog neprivilegovanog porta (većeg od 1024) koji će se koristiti za komunikaciju. Ova opcija može biti jako korisna ako se između nadgledajućeg hosta i udaljenog računara nalazi *firewall*.

Komanda provere se zadaje iza obligatornog flega `-c`, pri čemu je ona zapravo samo kratko ime komande koja je u potpunosti definisana u konfiguraciji `nrpe` demona na udaljenom hostu. Komanda iz primera je definisana na sledeći način:

```
command[check_ping_el]=/users/etf/mizoran/nagios/libexec/check_ping
-H 147.91.10.51 -w 100.0,40% -c 300.0,80% -p 5
```

U datom primeru, prikazana je komanda kojom se proverava PING servis na hostu `el` (`147.91.10.51`) posredstvom `check_ping` plagina na serveru `galeb`. Naravno, da bi plagin ispravno radio, na udaljenom hostu mora biti pokrenut ispravno iskonfigurisani `nrpe` daemon. Detalji o postupku instalacije i konfiguracije dati su u **Dodatku C**.

check_ping

Format: `./check_ping -H <IP_adresa_hosta> -w <wrta>,<wpl>% -c <crta>,<cpl>% [-p br_paketa] [-t <vreme_izvš>] [-L]`

Primer: `./check_ping -H 147.91.8.64 -w 100.0,80% -c 300.0,60% -p 5`

Ovo je jedan od plaginova koje sam najviše koristio. Plagin *`check_ping`* pomoću ICMP eho zahteva proverava dostupnost mrežnih hostova i uređaja. Takođe, meri prosek vremena u milisekundama za koje se ICMP paket vrati nazad (RTT – *round trip time*) i procentualni gubitak poslatih paketa (PL – *packet loss*). Obaveznim flegovima `-H`, `-w` i `-c` zadaju se IP adresa proveravanog hosta ili mrežnog interfejsa, vrednosti pragova za WARNING i CRITICAL status, respektivno. Kada se neki prag prekorači, plagin vraća odgovarajući status. Opcionim flegom `-p` definiše se broj ICMP paketa koji se šalje prilikom provere.

Plagin je dobio ime po programu `ping` koji je standardna alatka za proveru konektivnosti hosta u mreži. Gde god je bilo moguće, pomoću njega sam definisao proveru statusa hosta. Takođe, svakom hostu koji je mogao da bude pingovan sa servera `gandalf` pridružio sam PING servis radi praćenja RTT i PL parametara.

Pored ovog, postoji i *`check_fping`* plagin iste namene koji je napisan tako da vrši brže provere dostupnosti hosta u okruženjima sa ogromnog brojem hostova koji se proveravaju. Kako to u našem slučaju nije bilo potrebno, ovaj plagin nije uvršten u krajnju konfiguraciju.

check_pop

Format: `./check_pop -H <IP_adresa> -p <port> [-w <w_vreme>] [-c <c_vreme>] [-s <send>] [-e <expect>] [-W <wait>] [-t <vreme_izvrš>] [-v]`

Primer: `./check_pop -H 147.91.8.64`

Plagin *check_pop* služi za proveru ispravnosti rada POP3 *email* servera na mreži. U osnovi ovog plagina krije se *check_tcp* plagin koji otvara i zatvara konekciju na portu 110 udaljenog servera (ako nije definisano drugačije). Iz formata komande se može videti da je podržano još nekoliko interesantnih opcija. Omogućeno je definisanje pragova za dozvoljena vremena odziva POP3 servera u sekundama, koji za dato prekoračenje podižu odgovarajući alarm. Takođe, podržano je i slanje definisanog stringa serveru i upoređivanje odgovora sa očekivanim stringom odogovora. Opcionim flegom `-W` može se definisati vreme čekanja (*wait*) u sekundama od trenutka slanja stringa do početka poliranja odziva.

Moja upotreba ovog plagina se svodila na prosto otvaranje i zatvaranje konekcije na fakultetskim *email* serverima bez korišćenja dodatnih opcija kao što je navedeno u primeru.

check_procs

Format: `./check_procs -w <w_vreme> -c <c_vreme> [-s <stanje_procesa>] [-p <ppid>] [-u <korisnik>] [-a <niz_argumenata>] [-C <string_komande>]`

Primeri:
```
./check_procs -w 70 -c 100 -s RSZDT
./check_procs -w 1:10 -c :50 -u nagios
```

Plagin *check_procs* služi za proveru broja tekućih procesa u određenom stanju na lokalnom sistemu. Može se koristiti za nadgledanje ispravnosti rada nekog bitnog procesa na sistemu i u kombinaciji sa odgovarajućim *event* hendlerom koji, primera radi, pre nego što podigne uzbunu, pokušava da restartuje proces ili na neki drugi način reši problem.

Baziran je na standardnom Linux programu `ps` koji daje listu tekućih procesa na sistemu. Plagin radi tako što parsira izlaz programa *ps* i prebrojava tekuće proces sa nekim zadatim osobinama. Kada taj broj prevaziđe prag, iskoči iz zadatog opsega ili bude ispod zadatog minimuma procesa, plagin će podići odgovarajući alarm. Opsezi se zadaju u formatu '`min:max`' ili '`min:`' ili '`:max`' (ili '`max`').

Korišćenjem opcionog flega `-s` mogu se posebno prebrojavati procesi u određenom stanju, gde se sa R označava proces koji se izvršava (*running*), sa S proces koji je zaustavljen (*stopped*), sa Z proces koji se zaglavio (*zombie*), itd. Takođe, plagin daje mnoštvo mogućnosti obzirom da se mogu filtrirati i prebrojavati tekući procesi nekog zadatog korisnika odnosno programa, procesi sa zadatim PID-om ili sa zadatim stringom komande.

Ovo je jedan od plaginova koji zahteva da bude instaliran na računaru čije broj procesa proverava. Budući da sam posedovao nalog samo na serverima `gandalf` i `galeb`, imao sam mogućnost provere broja tekućih procesa samo na ovim računarima. Na nadgledajućem serveru, nadgledani su svi tekući procesi korišćenjem svih oznaka posle flega `-s`, `RSZDT`, (vidi prvi primer) kao što je dato u primeru.

Nadgledanje broja procesa na serveru `galeb` vršeno je pomoću indirektnih provera. Međutim, broj procesa koje je plagin prikazivao nije bio istinit što je bila posledica izvršavanja plagina sa dozvolama običnog korisnika. Problem se sastoji u tome da program *ps* kada se pokrene od strane običnog korisnika, daje samo listu njegovih procesa.

check_smtp

Format:
```
./check_smtp -H <IP_adresa> [-p <port>] [-e <string>]
[-f <from addr>] [-w <w_vreme>] [-c <c_vreme>]
[-t <vreme_izvrš>] [-v]
```

Primer:
```
./check_smtp –H 147.91.8.62
```

Plagin *check_smtp* služi za proveru ispravnosti rada SMTP *email* servera na mreži. U osnovi ovog plagina krije se *check_tcp* plagin koji otvara i zatvara konekciju na udaljenom serveru na standardnom portu 25 (ako nije definisano drugačije).

Iz formata se može videti da je podržano još nekoliko interesantnih opcija provere. Omogućeno je definisanje pragova za dozvoljena vremena odziva SMTP servera u sekundama, koja kada se prekorače podižu odgovarajući alarm. Takođe, opcionim flegom `-e` podržana je provera slaganja povratnog stringa po uspostavi konekcije sa očekivanim stringom odogovora. Ako se ovaj fleg ne navede, plagin podrazumevano očekuje da server na "HELO" SMTP komandu odgovori stringom "220" (konekcija uspešna). Flegom `-f` zadaje se adresa pozivajućeg računara koja se uključuje u MAIL komandu. Njegova upotreba je obavezna samo kod nadgledanja MS *Exchange email* servera. Bez navođenja `-f` flega, vrednost ovog argumenta je NULL.

Moja upotreba ovog plagina se svodila na prosto otvaranje i zatvaranje konekcije na standardnom portu SMTP protokola na fakultetskim *outgoing email* serverima bez korišćenja dodatnih opcija kao što je navedeno u primeru.

check_snmp

Format:
```
./check_snmp -H <IP_adresa> -o <OID> [-w <w_opseg>]
[-c <c_opseg>] [-C <community>] [-s <string>]
[-r <regexp>] [-R <regexi>] [-t <vreme_izvrš>]
[-l <labela>] [-u <jedinica>] [-p <port>]
[-d <delimiter>] [-D <output_delimiter>]
[-m <MIB_lista>] [-P <SNMP_verzija>] [-L <nivo_sigurn>]
[-U <korisničko_ime>] [-a <protokol_autorizac>]
[-A <autorizac_lozinka>] [-X <priv_lozinka>]
```

Primeri:
```
./check_snmp –H 147.91.10.51 –o .sysUpTime.0 –u 'ticks'

./check_snmp -H 147.91.8.1
-o .1.3.6.1.4.1.9.2.1.57.0,.1.3.6.1.4.1.9.2.1.58.0
-w 60:,70: -c 60:,70: -C xxxxx -l 'CPU_load'
-D '1min/5 min'

./check_snmp -H 147.91.8.1 -o ifOutErrors.19 -C xxxxx
-m IF-MIB -w 400 -c 500 -l 'ifOutErrors rcub->ETF eth'
```

Plagin *check_snmp* služi za nadgledanje sistemskih informacija na hostovima i mrežnim uređajima pomoću SNMP protokola koji je prihvaćen kao

osnovni metod upravljanja i nagledanja u TCP/IP mrežama. Zbog jako zgodnih mogućnosti koje pruža ovaj protokol, ovim plaginom se može pokriti nadgledanje jako velikog broja različitih servisa. Praktično, može se nadgledati bilo koja promenljiva u *Management Information Base* (MIB) stablu nadgledanog uređaja. MIB-ovi su pak skupovi statističkih i kontrolnih vrednosti organizovanih u strukturu stabla. Jedini uslov za nadgledanje ovim plaginom jeste da na nadgledanim hostovima ili mrežnim uređajima postoji implementiran SNMP protokol. Takođe, plagin zahteva da se na nadgledajućem hostu instalira programski alat iz NET-SNMP paketa[4] koji je zapravo jedna *open source* implementacija ovog protokola.

Plagin koristi program *snmpget* uključen u NET-SNMP paket kako bi dovukao vrednost zadatog identifikatora promeljive (tzv. *Object Identifier* - OID) u MIB stablu i potom obrađuje njen izlaz. U prvom navedenom primeru, plagin dovlači informaciju o *uptime* vremenu hosta `el.etf.bg.ac.yu` korišćenjem *snmpget* komande na sledeći način:

```
/usr/bin/snmpget -t 1 -r 9 -m ALL -v 1 -C public el:161 .sysUpTime.0
```

U okviru jedne komande plagina može da se dovuče do osam zapetom ili prazninom razdvojenih OID-a. Jedan ili lista OID-a se navode iza flega `-o` pri čemu se ravnopravno može koristiti brojna ili tekstualna notacija. U razmatranom primeru, umesto (skraćenog) tekstualnog zapisa OID-a koji čuva podatak o proteklom vremenu od početka rada sisema (`sysUpTime.0`) može se koristiti puna notacija `.1.3.6.1.2.1.1.3.0`. Zbog veće preglednosti, u konfiguracionim datotekam sam koristio tekstualnu notaciju gde god je to bilo moguće.

Izlaskom dovučene ili dovučenih vrednosti OID-a van zadatih opsega celih brojeva (definisano flegovima `-w` i `-c`), plagin vraća non-OK status. Opsezi se notiraju dvotačkom (npr. `min:max`), dok se obični celi brojevi tumače kao gornje granice, odnosno pragovi. Beskonačna vrednost granice opsega predstavlja se izostavljanjem vrednosti za tu granicu (npr. `min:`). Ako je u komandi provere zadata lista OID-a, kao u drugom primeru, iza flegova `-w` i `-c` je potrebno navesti liste zapetom odvojenih opsega. Plagin vraća status OK samo ako se sve nadgledane vrednosti nalaze unutar odgovarajućih definisanih opsega.

Flegom `-C` (veliko slovo 'C'!) definiše se *community* string koji predstavlja bezbednosni mehanizam korišćen u većini verzija SNMP protokola. Ovaj string predstavlja neku vrstu lozinke sadržane u SNMP poruci kojom program menadžer (u ovom slučaju program *snmpget*) zahteva neku informaciju ili upravljačku operaciju od agentskog programa na nadgledanom uređaju. Ukoliko zahtev sadrži agentu poznati *community* string, na ovaj zahtev se odgovara. Ako se u plaginu ovaj fleg ne navede, za SNMP upit se koristi podrazumevani *community* string 'public'. To je obično *default* vrednost stringa sa *read only* pravom pristupa u većini implementacija koja zadovoljava potrebe nadgledanja. Na primer, sve mašine na mreži fakulteta osim rutera i svičeva su mogle da se nadgledaju pomoću ovog *default community* stringa. Iz sigurnosnih razloga, postavljeni *community* stringovi na važnijim ruterima i svičevima su obično tajni, te u poslednja dva primera za glavni fakultetski ruter string nije eksplicitno naveden, već na njegovom mestu stoji 'xxxxx'.

Verzija SNMP protokola koja se može koristiti za nadgledanje zavisi pre svega od verzije koju koristi agent na nadgledanom hostu. Ovo se kontroliše flegom `-P`. SNMPv3 protokol ima dobro razrađene sigurnosne mehanizme, međutim danas je još uvek malo uređaja koji ga podržavaju. Ukoliko se ipak koristi

ova verzija, na raspolaganju stoji nekoliko flegova koji su posvećeni samo kontroli sigurnosnih mehanizama implementiranih u SNMPv3 protokol. Podrška za tzv. "*User based Security Model for version 3 of SNMP*" je sadržana u sledećim opcijama:

- `-L` ili `--seclevel=[noAuthNoPriv|authNoPriv|authPriv]` – kontroliše nivo sigurnosti u smislu da li se SNMP komunikacija vrši bez enkripcije (`noAuthNoPriv`), samo sa autorizacijom (`authNoPriv`) ili i sa autorizacijom i sa privatnošću (`authPriv`).
- `-U` ili `--secname=USERNAME` – definiše korisničko ime za autentifikaciju
- `-a` ili `--authproto=[MD5|SHA]` – definiše korišćeni algoritam enkripcije autentifikacije, MD5 ili SHA
- `-A` ili `--authpassword=PASSWORD` – definiše korisničku lozinku za autentifikaciju ukoliko se koristi samo autorizacija
- `-X` ili `--privpasswd=PASSWORD` – definiše korisničku lozinku za autentifikaciju ukoliko se koristi i autorizacija i privatnost (enkriptovano DES algoritmom).

Za komunikaciju SNMP protokolom obično se koristi standardni port 161, ali pomoću opcionog flega `-p` sa udaljenim agentima moguće je ostvarivati komunikaciju na proizvojno definisanom portu. Ako se ovaj fleg ne navede, plagin će podrazumevano koristiti standardni SNMP port 161.

Flegovima `-u` i `-l` omogućeno je definisanje jedinice za vraćenu vrednost plagina i labele koja će se pojaviti na početku tekstulanog izlaza plagina, respektivno. U prvom primeru, u izlazu plagina iza vraćene vrednosti *uptime* stajaće jedinica *ticks* što doprinosi većoj razumljivosti. Slično, u poslednjem primeru, na početku tekstualnog izlaza plagina kojim se nadgleda broj grešaka na jednom od interfejsova glavnog fakultetskog rutera stajaće labela `` `ifOutErrors rcub->ETF eth` ``. Uključivanjem ovakvog kratkog opisa nadgledanog sevisa u tekstualni izlaz plagina značajno se može pojasniti sadržaj poruke, naročito u situaciji kada se nadgleda ogroman broj sličnih servisa (veliki broj interfejsa na ruteru) ili kada se informacija o CRITICAL statusu ovog servisa šalje putem SMS poruke.

Način obrade izlaza korišćenog *snmpget* programa takođe se može prilagoditi specifičnim potrebama korišćenjem sledećih flegova:

- `-d` ili `--delimiter=STRING` – definiše delimiter koji se koristi pri parsiranju tekstualnog izlaza *snmpget* komande. U obzir se uzimaju sve vrednosti koje se nađu sa desne strane znaka delimitera. Ako se fleg ne definiše, plagin kao delimiter koristi podrazumevano znak jednakosti ("=").
- `-D` ili `--output-delimiter=STRING` – definiše delimiter koji odvaja izlaz snmpget komande kada se zahteva dovlačenje vrednosti više OID-a.
- `-s` ili `--string=STRING` – definiše string za određeni OID koji se u tačno istom obliku očekuje u izlazu provere i u tom slučaju (za taj OID) vraća OK status.
- `-r` ili `--ereg=REGEX` – definiše regularni izraz (*regular expression* ili skraćeno *regex*) kojim se parsira izlaz provere i u slučaju uspeha (za taj OID) vraća OK status.
- `-R` ili `--eregi=REGEX` – definiše takođe regularni izraz kao prethodni fleg s jedinom razlikom da se u ovom slučaju **ne** razlikuju mala i velika slova (*case insensitive*).

Na kraju, flegom `-m` definiše se lista MIB stabala koje je potrebno učitati da bi *snmpget* mogao da pronađe i vrati vrednost određene promenljive OID-a. Ako se fleg ne navede, *snmpget* će se izvršavati kao da je u plaginu navedeno `-m ALL`, odnosno kao da su učitane sve raspoložive MIB liste. U poslednjem navedenom primeru u kome je prikazana provera broja grešaka na interfejsu rutera, ovim flegom je učitano samo IF-MIB stablo u kome se nalaze OID-i sa parametrima koji se prate na mrežnim interfejsima.

U mojoj implementaciji Nagiosa, oko polovine svih provera servisa nadgleda se pomoću ovog plagina. Njime je uglavnom vršeno nadgledanje stanja interfejsa rutera i glavnih *layer 3* svičeva na fakultetskoj mreži. Pored toga, ovim plaginom su nadgledana temperatura šasije rutera i svičeva, i dostupnost nekih hostova koji nisu mogli da se pinguju sa servera `gandalf`, a imali su implementiran SNMP protokol.

check_spop

Format:
```
./check_spop -H <IP_adresa> -p <port> [-w <w_vreme>]
[-c <c_vreme>] [-s <send_string>] [-e <očekiv_string>]
[-q <quit_string>] [-m <br_bajtova>] [-d <vreme_ček>]
[-t <vreme_izvrš>] [-v]
```

Primer:
```
./check_spop -H 147.91.8.8 -w 2 -c 8
```

Plagin *check_spop* služi za proveru ispravnosti rada *Secure* POP3 *email* servera na mreži. Iz formata komande se može videti da je podržano još nekoliko interesantnih opcija. Omogućeno je definisanje pragova za dozvoljena vremena odziva SPOP servera u sekundama, koji za dato prekoračenje podižu odgovarajući alarm. Takođe, podržano je i slanje definisanog stringa serveru, upoređivanje odgovora sa očekivanim stringom odgovora i zadavanje stringa za čisto zatvaranje konekcije sa serverom. Opcionim flegom `-d` može se definisati vreme čekanja (*delay*) u sekundama od trenutka slanja stringa do početka poliranja odziva. Takođe, opcionim flegom `-m` može se definisati broj bajtova nakon čijeg primanja plagin proglašava proveru uspešnom.

Moja upotreba ovog plagina se svodila na proveru uspostave veze i brzine odziva na jedinom fakultetskom SPOP *email* serveru `kondor` (`147.91.8.8`) kao što je prikazano u primeru.

check_swap

Format:
```
./check_swap [-a] -w <procenat_zauz% | br_slob_bajtova>
-c <procenat_zauz% | br_slob_bajtova>
```

Primer:
```
./check_swap -w 50% -c 90%
```

Plagin *check_swap* služi za proveru popunjenosti *swap* particija na lokalnoj *nix ili Linux platformi. Vrednost pragova za status WARNING i CRITICAL može biti zadata u procentima zauzetog memorijskog *swap* prostora ili celobrojnim vrednostima slobodnih bajtova. Kada slobodan memorijski prostor padne ispod definisanih pragova, plagin će podići odgovarajući alarm. Ako se navede opcioni plagin `-a`, plagin će sa zadatim pragovima uporediti svaku *swap* particiju ponaosob.

Ovo je jedan od plaginova koji zahteva da bude instaliran na računaru čije se *swap* particije proveravaju. Moja upotreba ovog plagina se svodila na proveru swap particije na nadgledajućem serveru `gandalf`, kao što je dato u primeru.

check_users

Format: `./check_users -w <br_korisnika> -c <br_korisnika>`

Primer: `./check_users –w 15 -c 30`

Plagin *check_users* služi za proveru broja logovanih korisnika na lokalnoj *nix ili Linux platformi. Plagin je baziran na standardnoj Unix/Linux programskoj alatki `who` koji izlistava sve ulogovane korisnike na sistemu.

Vrednost pragova za status WARNING i CRITICAL se zadaje celim brojevima logovanih korisnika. Kada broj logovanih korisnika pređe zadatu vrednost praga, plagin će podići odgovarajući alarm.

Ovo je jedan od plaginova koji zahteva da bude instaliran na računaru na kome se nadgleda broj ulogovanih korisnika. Budući da sam posedovao naloge samo na serverima `gandalf` i `galeb`, imao sam mogućnost nadgledanja ulogovanih korisnika samo na ovim serverima. Naročito je bilo interesantno nadgledati broj ulogovanih korisnika na serveru `galeb` budući da ovaj server radi kao studentski terminal server i da je na njemu često ulogovano više od 15 korisnika.

check_tcp

Format:
```
./check_tcp -H <IP_adresa> -p <port> [-w <w_vreme>]
[-c <c_vreme>] [-s <send_string>] [-e <očekiv_string>]
[-q <quit_string>] [-m <br_bajtova>] [-d <vreme_ček>]
[-t <vreme_izvrš>] [-v]
```

Primer:
```
./check_tcp –H 147.91.8.64 -p 23
./check_tcp –H 147.91.8.8 -p 443
```

Plagin *check_tcp* služi za proveru raspoloživosti bilo kog TCP porta na nadgledanom hostu. Kao što je ranije već napominjano, iz ovog plagina je izvedeno nekoliko specijalizovanih plaginova kao što su *check_smtp*, *check_ftp*, *check_pop*, itd.

Iz formata se može videti da je ovaj plagin takođe opremljen svim potrebnim opcijama koje poseduju *check_ftp*, *check_pop* i sl., tako da se provere rada FTP, POP3, HTTP ili HTTPS servera mogu veoma lako da se realizuju i pomoću ovog plagina. Opcije neće biti objašnjavane pošto je to već učinjeno u izvedenim plaginovima.

Ovaj plagin je korišćen za proveru dostupnosti telnet servisa na serveru galeb. Naravno, flegom –p naznačavan je standardni port 23 za ovaj servis. Na sličan način vršeno je nadgledanje porta 443 na fakultetskim serverima koji nude HTTPS servis.

Instalacija

Sistemski zahtevi

Osnovni zahtev koji se mora ispuniti da bi Nagios radio jeste da mašina radi pod Linux-om (ili nekom Unix platformom) i da ima C prevodilac. Potreban je i iskonfigurisan TCP/IP stek, budući da se većina provera servisa u mreži vrši upravo njegovim posredstvom.

Hardverski zahtevi značajno zavise od veličine mreže čije se nadgledanje planira i zahtevnosti plaginova kojima se izvršavaju provere servisa, pa se minimalni hardverski zahtevi nigde ne navode eksplicitno. Hardverski zahtevi za implementaciju Nagiosa sa standardnim plaginovima na računarskoj mreži sa stotinak hostova i nekoliko stotina servisa su sledeći:

- najmanje 256MB sistemske memorije,
- IDE ili SCSI hard disk kapaciteta od bar 10GB
- statička IP adresa

CGI skriptovi koji su takođe uključeni u Nagios distribuciju nisu neophodni za rad jezgra programa, ali znatno poboljšavaju njegovu funkcionalnost. Za ispravan rad CGI skriptova potrebno je još instalirati sledeći softver:

1. HTTP server (preporučeno Apache)
2. GD biblioteke Tomasa Butela, verzija 1.6.3 ili novija (zahtevaju je *statusmap* i *trends* CGI skriptovi)

Instalacija Nagiosa u ovoj implementaciji izvedena je na razvojnom fakultetskom serveru `gandalf`, domen `etf.bg.ac.yu`, sa statičkom IP adresom 147.91.13.30. Server je raspolagao sa dva Intelova Pentium II procesora, 352900kB sistemske memorije i bzim SCSI hard diskovima.

Postupak prevođenja i instalacije

Postupak instalacije Nagiosa generalno može da se razlikuje u zavisnostu od ciljne platforme. U ovom radu konkretno je opisan postupak instalacije na *Gentoo* distribuciji GNU/Linux-a, kernel 2.4.20, koji je zbog automatizovanosti dosta jednostavniji u odnosu na postupak koji bi se izvodio na klasičnim distribucijama Linux-a ili UNIX-a. Sam postupak instalacije, naravno, ne utiče na kasniji rad programa, međutim radi kompletnosti, u **Dodatku A** dat je i detaljan postupak ručne instalacije Nagiosa.

Kao što je već pomenuto, *Gentoo* Linux distribucija poseduje jednostavan i napredan alat za menadžment paketa koji se zasniva na *Gentoo Portage* stablu. Umesto rada sa uobičajenim Makefile-ovima i *make* komandom, *Portage* koristi preimućstva *Pyton*-a i lakoću korišćenja *bash* skripta sa određenim objektno orjentisanim karakteristikama.

Ideja na kojoj počiva *Portage* sistem u mnogome liči na tradicionalni BSD-ov *ports* sistem. Oba dovlače izvorni kod paketa sa javnih servera na internetu, prevode (kompajliraju) pakete iz izvornog koda i korisniku omogućavaju da na svom sistemu bezbedno instalira, održava ili deinstalira softver. Takođe, oba automatski rešavaju eventualne međuzavisnosti paketa. *Gentoo*-ovo *Portage* stablo

se sastoji od velikog broja setova recepata (više od 4000) za prevođenje (bildovanje) paketa, tzv. *ebuilds*. Ovi *ebuild*-ovi bazirani na *bash* skriptu praktično kazuju Portage endžinu kako da prevede i instalira određeni softverski paket. Upotrebom *emerge* alata iz komandne linije, *Portage* se može koristiti za automatizovano instaliranje i održavanje svih paketa koji čine sam operativni sistem i aplikacije na sistemu.

Zahvaljujući postojanju odgovarajućeg *ebuild*-a u *Portage* stablu, **čitav** postupak prevođenja i instalacije Nagiosa i biblioteka i/ili paketa neophodnih za njegov rad se jednostavno pokreću sledećom komandom.

```
emerge nagios
```

Ovim se pokreće *Portage* mehanizam koji prevodi i instalira Nagios i sve zavisne pakete prema "uputstvima" iz odgovarajućeg Nagios-ove *ebuild* datoteke (`/usr/portage/net-analzyer/nagios/nagios-1.1.ebuild`).

Posle uspešnog prevođenja i instalacije svih potrebnih paketa i pod pretpostavkom da je uneta validna konfiguracija, Nagios je spreman za rad. Automatsko pokretanje Nagiosa kao daemonskog procesa pri podizanju sistema se obezbeđuje komandom,

```
rc-update add nagios default
```

U `/etc/init.d/` direktorijumu je instaliran inicijalizacioni skript `nagios` kojim se po potrebi program može i ručno kontrolisati njegovim pozivanjem sa željenim opcijama (start, stop, restart, itd.) na sledeći način

```
/etc/init.d/nagios start|stop|restart|status|reload|force-reload
```

Konfiguracija

U ovom poglavlju, dat je detaljan pregled sa objašnjenjima svih Nagiosovih konfiguracionih datoteka koje su korišćene za nadgledanje fakultetske računarske mreže. U pregledu sam težio da na što potpuniji način objasnim značenje svake direktive i da za svaku dam odgovarajući primer.

Pre bilo kakvih objašnjenja još ću navesti tri jednostavna pravila koja su poštovana prilikom pisanja konfiguracionih datoteka kako bi ih Nagios ispravno isparsirao.

1. linije koje počinju karakterom "#" tumače se kao komentari i nemaju nikakav uticaj na konfiguraciju,
2. imena promenljivih obavezno počinju od početka linije bez umetnutih belina ispred i
3. imena promenljivih razlikuju mala i velika slova (*case sensitive*)

Glavna konfiguraciona datoteka – nagios.cfg

Podrazumevani naziv glavne konfiguracione datoteke je `nagios.cfg`. U ovoj datoteci se definišu parametri kojim se globalno utiče na rad jezgra programa.

Definisanje log datoteke

Format: `log_file=<ime_datoteke>`

Primer: `log_file=/var/nagios/nagios.log`

Ova direktiva određuje gde će Nagios da kreira svoju glavnu log datoteku. To bi ujedno trebalo da bude i prva direktiva koja se definiše u konfiguracionoj datoteci, pošto će Nagios pokušati da u ovu datoteku upiše sve greške na koje naiđe u preostalim konfiguracionim datotekama. Ako se omogući rotacija logova, ova datoteka će automatski biti rotirana svakog časa, dana, nedelje ili meseca.

Imena i putanje konfiguracionih datoteka objekata

Format: `cfg_file=<ime_datoteke>`

Primeri:
```
cfg_file=/etc/nagios/hosts.cfg
cfg_file=/etc/nagios/aditional_hosts.cfg
cfg_file=/etc/nagios/services.cfg
cfg_file=/etc/nagios/commands.cfg
```

Ova direktiva se koristi za specifikaciju konfiguracionih datoteka objekata koje Nagios treba da koristi za nadgledanje. Konfiguracione datoteke objekata sadrže definicije hostova, servisa, grupa hostova, kotakata, grupa kontakata, komandi itd.

Kada je broj objekata u konfiguraciji jako veliki, konfiguracione daoteke se mogu podeliti u više datoteka korišćenjem više `cfg_file=` direktiva. Nagios će ih sve pročitati.

Primera radi, ovo može biti jako zgodno ukoliko postoji potreba za isključivanjem ili uključivanjem u proces nadgledanja određene grupe objekata čije se definicije nalaze u zasebnoj datoteci. U datom primeru, isključivanje čitave grupe hostova definisanih u datoteci `aditional_hosts.cfg` iz procesa

nadgledanja bi se jednostavno izvršilo dodavanjem znaka za komentar '#' ispred njegove direktive i reinicijalizacijom programa.

Direktorijum sa konfiguracionim datotekama objekta

Format: `cfg_dir=<directory_name>`

Primeri:
```
cfg_dir=/etc/nagios/hosts
cfg_dir=/etc/nagios/services
```

Ako je broj konfiguracionih datoteka toliko veliki da ni prethodno opisana direktiva nije zgodna za upotrebu, onda se može koristiti `cfg_dir` direktiva. Povećanje preglednosti konfiguracije se postiže definisanjem čitavog direktorijuma koji sadrži konfiguracione datoteke objekata.

Direktiva primorava Nagios da procesira sve konfiguracione datoteke u ovako navedenom direktorijumu sa ekstenzijom `.cfg` bez eksplicitnog navođenja njihovih imena. Na ovaj način konfiguracione datoteke mogu da se rasporede na željeni način u različite direktorijume pri čemu se za svaki konfiguracioni direktorijum ponaosob mora navesti direktiva `cfg_dir`.

Definisanje *resource* datoteke

Format: `resource_file=<ime_datoteke>`

Primer: `resource_file=/etc/nagios/resource.cfg`

Ova direktiva se koristi za definisanje opcione risors datoteke koja može da sadrži `$USERn$` definicije makroa. `$USER$` makroi se koriste za čuvanje korisničkih imena, lozinki i uopšte stvari koje se često koriste u definicijama komandi (kao npr. putanje do direktorijuma plaginova ili event hendlera). Pošto **nije** predviđeno da CGI skriptovi čitaju risors datoteke, iz sigurnosnih razloga poželjno je da se nad ovakvim datotekama postave veoma restriktivne dozvole (na primer, 600 ili 660).

Moguće je definisati više risors datoteka višestrukim navođenjem *`resource_file`* direktive u glavnoj konfiguracionoj datoteci. Nagios će sve da ih procesira. Primer `resource.cfg` datoteke u kojoj je prikazano kako se definišu `$USERn$` makroi takođe je dat u okviru ovog poglavlja.

Definisanje *temp* datoteke

Format: `temp_file=<ime_datoteke>`

Primer: `temp_file=/var/nagios/nagios.tmp`

Ovo je privremena (*temp*) datoteka koju Nagios povremeno kreira i upotrebljava prilikom osvežavanja podataka o statusu, komentarima itd. Temp datoteka se briše kada više nije potrebna.

Definisanje statusne datoteke

Format: `status_file=<ime_datoteke>`

Primer: `status_file=/var/nagios/status.log`

Ovu datoteku Nagios koristi za čuvanje informacija o statusu svih servisa koji se nadgledaju. Takođe, ovde se čuvaju i statusne informacije o svim hostovima koji su pridruženi nadgledanim servisima. Ovu datoteku koriste CGI skriptovi kako bi preko web interfejsa mogli da prikažu trenutni status nadgledanih objekata. Iz tog

razloga, CGI skriptovi moraju da imaju dozvole čitanja ove datoteke. Inače, Nagios briše ovu datoteku kad god se zaustavi i ponovo je kreira pri (re)startovanju.

Aggregate Status Update direktiva

Format: `aggregate_status_updates=<0/1>`

Primer: `aggregate_status_updates=1`

Ovom direktivom se određuje da li će Nagios da sakuplja (agregira) sveže informacije o statusu hostova, servisa i programa. Ako se ova opcija onemogući, statusne informacije će biti dovlačene pri svakom izvršavanju provera servisa ili hosta. Iako se ovim postiže praktično trenutno ažuriranje (*update*) statusnih informacija, može doći do velikog opterećenje procesora i I/O uređaja na sistemu, pogotovu ako se nadgleda veliki broj servisa. Ako se pak želi da Nagios dovlači samo promene statusnih informacija (koje se nalaze u status datoteci) svakih par sekundi (što je definisano `status_update_interval` direktivom), onda treba uključiti ovu opciju.

Omogućavanje ove opcije se toplo preporučuje (čak i za kratke intervale) osim ako ne postoji dobar razlog za njeno onemogućavanje.

0 = isključena opcija agregiranja *update*-ova

1 = uključena opcija agregiranja *update*-ova (default)

Direktiva *Aggregated Status Interval*

Format: `status_update_interval=<br_sekundi>`

Primer: `status_update_interval=15`

Ovde se kontroliše koliko često (u sekundama) će Nagios vršiti ažuriranje statusnih informacija u status datoteci. Minimalni interval je 5 sekundi. Ako je prethodno onemogućena opcija agregiranja statusa, ova direktiva nema efekta.

Korisničko ime pod kojim se izvršava Nagios

Format: `nagios_user=<korisničko_ime/korisnički_UID>`

Primer: `nagios_user=nagios`

Ovde se podešava korisničko ime pod kojim će se Nagios proces izvršavati. Nakon inicijalnog podizanja programa i pre početka bilo kakvog nadgledanja, Nagios će odbaciti dozvole pod kojim je pokrenut i nastaviti rad pod ovim korisničkim nalogom i njemu pridruženim dozvolama. Opciono, mogu se navesti ili korisničko ime ili korisnički UID broj. Sasvim je svejedno.

Grupa korisnika kojoj pripada korisnik `nagios`

Format: `nagios_group=<ime_grupe/grupni_GID>`

Primer: `nagios_group=nagios`

Potpuno analogni komentari kao o prethodno objašnjenoj direktivi za definisanje korisničkog imena pod kojim se izvršava Nagios.

Kontrola obaveštavanja

Format: `enable_notifications=<0/1>`

Primer: `enable_notifications=1`

Ovom direktivom se na nivou čitavog programa kontroliše mogućnost da Nagios šalje bilo kakva obaveštenja. Ako se ova opcija isključi, Nagios neće poslati ni jedno obaveštenje ni za jedan host ili servis. Opcije su:

0 = isključena opcija obaveštavanja na nivou čitave konfiguracije

1 = uključena opcija obaveštavanja na nivou čitave konfiguracije (default)

Kontrola provere servisa

Format: `execute_service_checks=<0/1>`

Primer: `execute_service_checks=1`

Ovom direktivom određuje se da li je Nagiosu dozvoljeno da izvršava provere servisa (*service checks*). Ako je ova opcija deaktivirana, Nagios neće vršiti nikakve provere servisa i ostaće u nekoj vrsti "uspavanog" stanja (pri čemu još uvek može da prima pasivne provere, ako i one nisu isključene). Ova direktiva se najčešće koristi kada se konfigurišu redundantni *back up* monitoring serveri ili kada se konfiguriše distribuirano monitoring okruženje. Opcije su sledeće:

0 = provere servisa zabranjene na nivou čitave konfiguracije

1 = provere servisa dozvoljene na nivou čitave konfiguracije (default)

Kontrola prihvatanja pasivnih provera servisa

Format: `accept_passive_service_checks=<0/1>`

Primer: `accept_passive_service_checks=1`

Ovom direktivom se kontroliše mogućnost Nagiosa da prihvata pasivne provere servisa na nivou čitave konfiguracije. Ako je opcija isključena, Nagios neće prihvatiti nijednu pasivnu proveru servisa bez obzira ako su, na primer, one omogućene lokalnim direktivama u definicijama servisa. Opcije su sledeće:

0 = zabrana prihvatanja pasivnih provera servisa na nivou čitave konfiguracije

1 = dozvoljeno prihvatanja pasivnih provera servisa na nivou čitave konfiguracije (default)

Kontrola rukovanja događajima (*event handling*)

Format: `enable_event_handlers=<0/1>`

Primer: `enable_event_handlers=1`

Ova direktiva globalno određuje da li se omogućava rukovanje događajima (*event handling*) na nivou čitave konfiguracije. Ako se ova opcija isključi, Nagios neće pokretati ni jedan *event handler*.

0 = zabranjeno pokretanje *event* hendlera na nivou čitave konfiguracije

1 = dozvoljeno pokretanje *event* hendlera na nivou čitave konfiguracije (default)

Metod rotacije log datoteka

Format: `log_rotation_method=<n/h/d/w/m>`

Primer: `log_rotation_method=d`

Nagios nudi nekoliko metoda za rotaciju log datoteke. Opcione vrednosti su:

n = bez rotacije (log datoteka se nikada ne rotira) (default)

h = na sat (log datoteka se rotira na početku svakog punog časa)

d = dnevno (log datoteka se rotira svakog dana u ponoć)

w = nedeljno (log datoteka se rotira subotom u ponoć)

m = mesečno (log datoteka se rotira u ponoć svakog poslednjeg dana u mesecu)

Putanja arhive logova

Format: `log_archive_path=<putanja>`

Primer: `log_archive_path=/var/nagios/archives/`

Ovom direktivom se definiše ime direktorijuma u koji će Nagios da odlaže rotirane log datoteke. Direktiva se ignoriše ako nije odabran nikakav metod rotacije log , tj. `log_rotation_method=n`.

Kontrola provere eksternih komandi

Format: `check_external_commands=<0/1>`

Primer: `check_external_commands=1`

Ovom direktivom se kontroliše da li će Nagios proveravati sadržaj datoteke eksternih komandi. Ova opcija treba da bude uključena ako se planira korišćenje komandnih CGI skriptova za izdavanje komandi putem web interfejsa. Isto je neophodno ukoliko treba dozvoliti da eksterne aplikacije (*third party software*) budu u mogućnosti da izdaju komande Nagiosu upisivanjem u komandnu datoteku.

Treba imati na umu i to da omogućavanje ove opcije nosi izvesne sigurnosne rizike, budući da svako ko može da upiše komandu u komandnu datoteku, može da kontroliše i čitav Nagios proces. S tim u vezi, neophodno je obezbediti pravilno setovanje dozvola nad ovom datotekom kao što je detaljno objašnjeno na kraju Dodatka A.

0 = zabranjena provera datoteke eksternih komandi (default)

1 = dozvoljena provera datoteke eksternih komandi

Kontrola intervala između provera datoteke eksternih komandi

Format: `command_check_interval=<xxx>[s]`

Primeri:
```
command_check_interval=2
command_check_interval=30s
command_check_interval=-1
```

Ovom direktivom se kontroliše dužina vremeskog intervala između dve provere sadržaja datoteke sa eksternim komandama. Interval se zadaje celim brojem koji označava broj "vremenskih jedinica". Ukoliko nije menjana podrazumevana vrednost promenljive *`interval_lenght`* (dužina "jediničnog" intervala) od 60, ova vrednost podrazumeva minute. Ako se iza zadatog broja doda slovo "**s**" (npr. 30s), Nagios će interval tumačiti kao interval u sekundama.

Zadavanjem vrednosti **-1**, Nagios će vršiti provere sadržaja datoteke eksternih komandi što je češće moguće. Svaki put kada izvrši proveru eksterne komande, on će, pre nego što nastavi sa svojim ostalim zadacima, pročitati i obraditi sve komande koje se nalaze u komandnoj datoteci.

Definisanje datoteke eksternih komandi

Format: `command_file=<ime_datoteke>`

Primer: `command_file=/var/nagios/rw/nagios.cmd`

Ovo je direktiva kojom se definiše putanja i ime datoetke eksternih komandi. Nagios će proveriti sadržaj ovog datoteke u potrazi za eksternim komandama koje treba da se izvrše. Za upisivanje komandi u ovu datoteku kroz web interfejs zadužen je *command* CGI skript. Ako su nad ovim fajlom valjano definisane dozvole upisa, i ostali *third party programi* mogu u njega da upisuju svoje komande. Datoteka sa eksternim komandama implementirna je kao *named pipe* (FIFO), koji se kreira pri inicijalizaciji Nagiosa, i briše pri njegovom zaustavljanju. Ako pri inicijalizaciji Nagiosa datoteka već postoji, Nagios će se zaustaviti i vratiti poruku o greški.

Downtime datoteka

Format: `downtime_file=<ime_datoteke>`

Primer: `downtime_file=/var/nagios/downtime.log`

Ovom direktivom definiše se *downtime* . U ovoj datoteci Nagios smešta informacije o planiranom gašenju određenih hostova i servisa zbog redovnog održavanja. Komentari o hostovima i servisima mogu se pregledavati ili dodavati putem *Extended informations* CGI skripta.

Datoteka za smeštanje komentara

Format: `comment_file=<ime_datoteke>`

Primer: `comment_file=/var/nagios/comment.log`

Ovom direktivom definiše sedatotekasa komentarima. U ovom datoteci Nagios čuva tekstualne komentare o hostovima i servisima. Komentari se mogu videti ili dodavati bilo za hostove ili servise kroz *Extended information* CGI skript.

Lock datoteka

Format: `lock_file=<ime_datoteke>`

Primer: `lock_file=/var/nagios/nagios.lock`

Ovom direktivom specificira se lokacija *lock* datoteke koju će Nagios kreirati ako se pokrene kao daemon. Ovadatotekasadrži identifikacioni broj Nagios procesa (PID).

Opcija pamćenja stanja hostova i servisa

Format: `retain_state_information=<0/1>`

Primer: `retain_state_information=1`

Ova direktiva kontroliše da li će Nagios da sačuva informacije o stanjima hostova i servisa između zaustavljanja i reinicijalizcije programa. Ako se ova opcija omogući, još je neophodno definisati ime *state retention* datoteke `state_retenton_file` direktivom. U tom slučaju, Nagios će sačuvati sve informacije o stanjima hostova i servisa pre nego što se zaustavi ili reinicijalizuje, ili ako se ponovo bude inicijlizovao, iz pomenute datoteke pročitaće prethodno sačuvane informacije o stanjima hostova i servisa. Opcije su:

0 = onemogućeno pamćenje informacija o statusu hostova i servisa

1 = omogućeno pamćenje informacija o statusu hostova i servisa (default)

State retention datoteka

Format: `state_retention_file=<ime_datoteke>`

Primer: `state_retention_file=/var/nagios/status.sav`

U ovoj datoteci Nagios "pamti" informacije o stanjima servisa i hostova pre nego što se zaustavi. Prilikom procesa reinicijalizacije pre nego što počne sa nadgledanjem, iskoristiće informacije sačuvane u ovoj datoteci za inicijalizaciju svih tekućih stanja hostova i servisa. Kada Nagios pročita informacije o inicijalnim stanjima, ovadatotekase briše. Da bi primorali Nagios da sačuva informacije o stanjima do svog sledećeg pokretanja, potrebno je prethodno opisanom direktivom uključiti opciju pamćenja stanja.

Kontrola učestalosti automatskog ažuriranja *state retention* datoteke

Format: `retention_update_interval=<br_minuta>`

Primer: `retention_update_interval=60`

Ovom direktivom se kontroliše koliko često (izraženo u minutima) će Nagios automatski ažurirati datoteku sa sačuvanim informacijama o stanjima hostova i servisa tokom njegovog normalnog rada. Ako se zada vrednost **0**, Nagios neće čuvati ove informacije u regularnim intervalima, već samo prilikom zaustavljanja ili reinicijalizacije. Ako je opcija pamćenja stanja isključena, ova direktiva nema efekta.

Kontrola čuvanja stanja programa

Format: `use_retained_program_state=<0/1>`

Primer: `use_retained_program_state=1`

Ovim se kontroliše da li će Nagios pri reinicijalizaciji koristiti vrednosti nekih globalnih konfiguracionih parametara na osnovu njihovih vrednosti iz datoteke definisane *state_retention_file* direktivom. Neki od ovih globalnih konfiguracionih parametara koji se obično čuvaju prilikom reinicijalizacije programa su parametri zadati direktivama *enable_notifications*, *enable_flap_detection*, *enable_event_handlers*, *execute_service_checks* i *accept_passive_service_checks*. Ako je opcija pamćenja stanja onemogućena, ova direktiva nema efekta. Opcije su:

0 = globalni konfiguracioni parametri se **ne** pamte u *state retention* datoteci

1 = globalni konfiguracioni parametri se pamte u *state retention* datoteci

Kontrola logovanja u sistemsku log datoteku (*syslog*)

Format: `use_syslog=<0/1>`

Primer: `use_syslog=1`

Ovom direktivom se definiše da li se poruke koje proizvodi Nagios upisuju u sistemsku log datoteku na lokalnom hostu.

0 = ne

1 = da

Kontrola logovanja notifikacija

Format: `log_notifications=<0/1>`

Primer: `log_notifications=1`

Određuje da li se u Nagiosovoj log datoteci vodi evidencija o odaslatim obaveštenjima. Ako je definisan veći broj kontakata ili se javlja veći broj regularnih grešaka servisa, logdatotekamože relativno brzo da se uveća.

0 = obaveštenja se ne loguju

1 = obaveštenja se loguju

Kontrola logovanja ponovnih pokušaja provere servisa

Format: `log_service_retries=<0/1>`

Primer: `log_service_retries=1`

Određuje da li se loguju ponovni pokušaji provere servisa. Ponovna provera servisa dešava se kada provera vrati non-OK status, a Nagios je konfigurisan tako da izvestan broj puta ponovi proveru pre konačnog obaveštavanja o problemu. U ovoj situaciji se kaže da je servis u SOFT stanju greške. Logovanje ponovnih provera servisa je najkorisnija kada se testira ispravost rada *event* hendlera servisa. Opcije:

0 = ponovni pokušaji provere servisa se ne loguju

1 = ponovni pokušaji provere servisa se loguju

Kontrola logovanja ponovnih pokušaja provere hostova

Format: `log_host_retries=<0/1>`

Primer: `log_host_retries=1`

Određuje da li se loguju ponovni pokušaji provere hostova. Ponovna provera hostova dešava se kada provera rezultuje non-OK statusom, a Nagios je konfigurisan tako da izvestan broj puta ponovi proveru pre konačnog obaveštavanja o problemu sa hostom. U ovoj situaciji se kaže da je host u SOFT stanju greške. Logovanje ponovnih provera hostova je najkorisnija kada se testira ispravost rada *event* hendlera hostova.

0 = ponovni pokušaji provere hostova se ne loguju

1 = ponovni pokušaji provere hostova se loguju

Kontrola logovanja *event* hendlera

Format: `log_event_handlers=<0/1>`

Primer: `log_event_handlers=1`

Event hendleri su opcione komande koje se izvršavaju kad god servisi ili hostovi promene svoje stanje. Logovanje *event* hendlera je najkorisnije kada se testira ispravnost rada skripta *event* hendler. Ova opcija se kontroliše upravo ovom direktivom.

0 = izvršavanje *event* hendlera se ne loguje

1 = izvršavanje *event* hendlera se loguje

Kontrola logovanja inicijalnih stanja

Format: `log_initial_states=<0/1>`

Primer: `log_initial_states=0`

Ova direktiva određuje da li se Nagios primorava da loguje sva inicijalna stanja hostova i servisa, čak i ako su u OK stanju. Inicijalna stanja servisa i hostova se normalno loguju samo kada postoji problem kod prve provere. Uključivanje ove

opcije je korisno ako se koristi neka aplikacija ili skript koji skenira log datoteku i utvrđuje dugoročnu statistiku grešaka servisa i hostova.

0 = inicijalna stanja se ne loguju (default)

1 = inicijalna stanja se loguju

Kontrola logovanja eksternih komandi

Format: `log_external_commands=<0/1>`

Primer: `log_external_commands=1`

Ova direktiva određuje da li će Nagios zapisivati u svoju log datoteku svako izvršavanje eksternih komandi, a koje prima kroz datoteku eksternih komandi.

Ovde se mora napomenuti da ova direktiva ne kontroliše logovanje pasivnih provere servisa (koje jesu tip eksterne komande). Logovanje pasivnih provera se kontroliše narednom direktivom.

0 = izvršavanje eksternih komandi se ne loguje

1 = izvršavanje eksternih komandi se loguje (default)

Kontrola logovanja pasivnih provera servisa

Format: `log_passive_service_checks=<0/1>`

Primer: `log_passive_service_checks=1`

Ovom direktivom se određuje da li se u Nagiosov log zapisuju pasivne provere servisa koje se primaju posredstvom datoteke eksternih komandi. Ako se konfiguriše distribuirano monitoring okruženje ili se planira rukovanje ogromnim brojem pasivnih provera na regularnoj bazi, preporučuje se isključivanje ove opciju kako logdatotekane bi previše brzo narasla.

0 = ne loguje

1 = loguje (default)

Definisanje globalnog *event* hendlera hostova

Format: `global_host_event_handler=<komanda>`

Primer: `global_host_event_handler=log-host-event-to-db`

Ova direktiva dozvoljava da se definiše komanda globalnog *event* hendlera hostova koja treba da se izvrši prilikom promene stanja svakog hosta. Globalni *event* hendler se izvršava neposredno pre ostalih (lokalnih) *event* hendlera koji se (opciono) definišu u svakoj definiciji hosta ponaosob. Argument direktive je kratko ime komande koja se u celini definiše u objektnoj konfiguracionom datoteci `checkcommands.cfg`. Maksimalno dozvoljeno vreme izvršavanja globalnog event hendlera kontroliše se direktivom `event_handler_timeout`.

Definisanje globalnog *event* hendlera servisa

Format: `global_service_event_handler=<komanda>`

Primer: `global_service_event_handler=log-service-event-to-db`

Ova direktiva dozvoljava da se definiše komanda globalnog *event* hendlera servisa koja treba da se izvrši prilikom svake promene stanja servisa. Globalni *event* hendler se izvršava neposredno pre ostalih (lokalnih) *event* hendlera koji se (opciono) definišu u svakoj definiciji servisa ponaosob. Argument direktive je

kratko ime komande koja je definisana u objektnoj konfiguracionoj datoteci `checkcommands.cfg`. Maksimalno dozvoljeno vreme izvršavanja komande hendlera kotroliše se direktivom `event_handler_timeout`.

Definisanje intervala sna između provera

Format: `sleep_time=<br_sekundi>`

Primer: `sleep_time=1`

Ovom direktivom definiše se broj sekundi tokom kojih Nagios "spava" (pravi pauzu) pre nego što proveri da li u redu za čekanje postoji sledeća provera servisa za izvršavanje. Treba reći da će Nagios da spava samo ako u redu za čekanje postoje zaostale provere servisa (ako red nije prazan).

Metod vremenskog raspoređivanja izvršavanja provera

Format: `inter_check_delay_method=<n/d/s/x.xx>`

Primer: `inter_check_delay_method=s`

Ova direktiva dozvoljava kontrolisanje načina inicijalnog vremenskog raspoređivanja provera servisa u redu za njihovo izvršavanje. Upotrebom "pametnog" izračunavanja raspoređivanja (default), Nagios će izračunati srednji interval između provera i ravnomerno rasporediti trenutke izvršavanja inicijalnih provera svih servisa unutar tog intervala i tako će eliminisati neravnomerno opterećenje procesora nadgledajućeg hosta. Nekorišćenje inicijalnog raspoređivanja provera je generalno loša ideja, osim ako se ne testira funkcionalnost paralelizacije provera servisa. U tom slučaju, sve inicijalne provere će biti raspoređene za izvršavanje u istom trenutku! Postojeće opcije su:

n = ne koristi se vremensko razmeštanje provera (sve se izvršavaju paralelno u vremenu)

d = koristi se "glupo" vremensko razmeštanje od 1s između provera nezavisno od broja provera i sl.

s = koristi se pametno rapoređivanje provera u vremenu (default)

x.xx = koristi se korisnički definisano raspoređivanje sa intervalom od x.xx sekundi između dve susedne provere

Definisanje faktora učešljavanja servisa (*service interleave factor*)

Format: `service_interleave_factor=<s/x>`

Primer: `service_interleave_factor=s`

Ovom direktivom se određuje način na koji se provere servisa učešljavaju. Učešljavanjem (tj. interlivingom) se obezbeđuje mnogo ravnomerniji raspored provera servisa stim da se smanjuje opterećenje udaljenih (nadgledanih) hostova i ubrzava ukupni proces detekcije problema na hostovima. Postavljanje vrednosti faktora na **1** jednako je isključenju opcije učešljavanja provera. Korišćenje paralelnih provera servisa sa isključenom opcijom učešljavanja provera može dovesti do situacije da hostovi budu bombardovani istovremenim proverama što kao rezultat može imati neuspele provere ili dobijanje netačnih rezultata od hostova preopterećenih drugim zahtevima za proverom servisa. Preporučuje se postavljanje vrednosti na **s** (*smart*) za "pametno" izračunavanje faktora učešljavanja osim ako ne postoji dobar razlog za njegovu promenu. Ponuđene opcije su:

x = korisnički definisana vrednost faktora interlivinga servisa (≥1).

s = koristi se pametni interliving (default)

Definisanje maksimalnog broja paralelnih provera servisa

Format: `max_concurrent_checks=<maks_broj>`

Primer: `max_concurrent_checks=20`

Ova direktiva omogućava definisanje maksimalnog broja provera servisa koje se mogu izvršavati paralelno u vremenu. Postavljanjem vrednosti na 1 mogućnost konkurentnog izvršavanja provera se isključuje. Postavljanjem vrednosti na 0 (*default*) daje se mogućnost neograničenog broja konkurentnih provera. U zavisnosti od raspoloživih resursa ova promenljiva se postavlja na odgovarajuću vrednost pošto direktno utiče na maksimalno opterećenje kojem će sistem na kome se izvršava Nagios biti izložen.

Definisanje učestalosti obrade rezultata provera servisa

Format: `service_reaper_frequency=<br_sekundi>`

Primer: `service_reaper_frequency=10`

Ova direktiva definiše učestalost obrade rezultata provera servisa u sekundama. U zadatom inervalu vremena vrši se obrada rezultata izvršenih paralelnih provera servisa.

Definisanje dužine jediničnog vremenskog intervala

Format: `interval_lenght=<br_sekundi>`

Primer: `interval_lenght=60`

Ova direktiva definiše trajanje "jediničnog intervala" u sekundama. Jedinični interval se koristi za merenje vremena u redovima za čekanje, kod ponovnih obaveštenja, itd. Takođe, koriste se u konfiguracionoj datoteci hostova za definisanje učestalosti izvršavanja provera servisa, ponovnog obaveštavanja kontakta, itd.

Podrazumevana vrednost jediničnog intervala je **60**, što znači da jedinica u konfiguracionoj datoteci hostova "vredi" 60 sekundi, odnosno 1 minut. Neoprezna promena ovog parametra nosi izvestan rizik!

Kontrola agresivne provere hostova

Format: `use_aggressive_host_checks=<0/1>`

Primer: `use_aggressive_host_checks=0`

Ovom direktivom kontroliše se način provere statusa hostova. Nagios se trudi da pametno odabere trenutak provere statusa hostova. Generalno, onemogućavanje opcije agresivnog proveravanja statusa hostova dozvoljava Nagiosu da donosi pametnije odluke i proverava hostove nešto brže. S druge strane, omogućavanjem ove opcije vreme potrebno za proveru hostova će se uvećati, ali će se pouzdanost vraćenih podataka neznatno popraviti. Osim ako ne postoji problem prepoznavanja oporavka hostova, preporučuje se onemogućavanje ove opcije.

0 = hostovi se ne proveravaju agresivno (*default*)

1 = hostovi se agresivno proveravaju

Kontrola detekcije "flepovanja"

Format: `eable_flap_detection=<0/1>`

Primer: `enable_flap_detection=0`

Ovim se određuje da li će Nagios pokušati da detektuje hostove i servise koji "flepuju" (engl. *flapping*). Za neki host ili servis kažemo da "flepuje" kada menja status suviše često. Detekcija flepovanja je bitna jer se na taj način izbegavaju baraži poslatih obaveštenja. Kada Nagios detektuje da host ili servis flepuje, privremeno će isključiti opciju obaveštavanja za taj host ili servis dok flepovanje ne prestane. U ovom trenutku detekcija flepovanja je u eksperimentalnoj fazi tako da ovu opciju treba koristiti sa izvesnom rezervom!

0 = isključena detekcija flepovanja (*default*)

1 = uključena detekcija flepovanja

Definisanje donjeg praga flepovanja servisa

Format: `low_service_flap_treshold=<procenat>`

Primer: `low_service_flap_treshold=25.0`

Ovom direktivom se definiše donji prag za detekciju flepovanja servisa.

Definisanje gornjeg praga flepovanja servisa

Format: `high_service_flap_treshold=<procenat>`

Primer: `high_service_flap_treshold=50.0`

Ovom direktivom se definiše gornji prag za detekciju flepovanja servisa.

Definisanje donjeg praga flepovanja hosta

Format: `low_host_flap_treshold=<procenat>`

Primer: `low_host_flap_treshold=25.0`

Ovom direktivom se definiše donji prag za detekciju flepovanja hosta.

Definisanje gornjeg praga flepovanja hosta

Format: `high_host_flap_treshold=<procenat>`

Primer: `high_host_flap_treshold=50.0`

Ovom direktivom se definiše gornji prag za detekciju flepovanja hosta.

Kontrola provere zavisnost servisa u SOFT stanju

Format: `soft_state_dependencies=<0/1>`

Primer: `soft_state_dependencies=0`

Ova direktiva određuje da li Nagios koristi informaciju o SOFT stanju servisa kada proverava zavisnosti servisa. U normalnim uslovima Nagios će koristiti samo poslednje HARD stanje servisa prilikom provere zavisnosti.

0 = ne koriste se SOFT stanja prilikom provere zavinosti (*default*)

1 = koriste se SOFT stanja prilikom provere zavinosti

Vremenska kontrola provere servisa (*service check timeout*)

Format: `service_check_timeout=<br_sekundi>`

Primer: `service_check_timeout=60`

Ovom direktivom se definiše maksimalno dozvoljeno vreme u sekundama izvršavanja provera servisa. Ako provera probije ovaj interval, ona se ubija, vraća se CRITICAL status servisa i loguje se poruka o greški probijanja intervala.

Često postoji velika konfuzija oko toga šta ova opcija zapravo radi. Zamišljena je kao mehanizam poslednjeg leka za otklanjanje plaginova koji se ne ponašaju kako bi trebalo niti se završavaju u predviđenom roku. Ovaj parametar treba podesiti na neku visoku vrednost (kao što je 60 ili više sekundi), tako da svaka provera servisa ima dovoljno vremena da se izvrši unutar ovog intervala. Ako provera bude trajala duže, Nagios će je ubiti misleći da se radi o zaglavljenom procesu.

Vremenska kontrola provere hosta (*host check timeout*)

Format: `host_check_timeout=<br_sekundi>`

Primer: `host_check_timeout=60`

Ovom direktivom se definiše maksimalno dozvoljeno vreme u sekundama izvršavanja provera hostova. Ako provera probije ovaj interval, ona se "ubija" i vraća se CRITICAL status, a za host se podrazumeva da je u DOWN statusu. Takođe, poruka greške o probijanju intervala će se logovati.

Ova opcija je zamišljena je kao mehanizam poslednjeg rešenja za otklanjanje plaginova koji se ne ponašaju kako bi trebalo niti se završavaju u predviđenom roku. Ovaj parametar treba podesiti na neku visoku vrednost (kao što je 60 ili više sekundi), tako da svaka provera hosta ima dovoljno vremena da se izvrši unutar ovog intervala. Ako provera bude trajala duže, Nagios će je ubiti podrazumevajući da se radi o procesu koji se zaglavio.

Vremenska kontrola *event* hendlera (*event handler check timeout*)

Format: `event_handler_timeout=<br_sekundi>`

Primer: `event_handler_timeout=60`

Ova direktiva definiše maksimalni dozvoljeni vremenski interval u sekundama izvršavanja *event* hendlera. Ako bilo koji hendler probije ovaj interval, biće ubijen, a poruka o greški probijanja intervala će biti logovana.

Kao i kod vremenske kontrole provera hostova, ova opcija je zamišljena kao mehanizam poslednjeg rešenja za ubijanje komandi *event* hendlera koje se ne ponašaju kako bi trebalo ili se ne završavaju u predviđenom roku. Ovu promenljivu treba podesiti na neku visoku vrednost (kao što je 60 ili više sekundi), tako da svaka komanda ima dovoljno vremena da se izvrši unutar ovog intervala. Ako komanda bude trajala duže, Nagios će je ubiti misleći da se radi o zaglavljenom procesu.

Vremenska kontrola opsesivno kompulsivne komande procesora servisa

Format: `ocsp_timeout=<br_sekundi>`

Primer: `ocsp_timeout=5`

Ova direktiva definiše maksimalnu dužinu intervala u sekundama koju će Nagios dati komandi opsesivno kompulsivnog procesora provera servisa da se izvršava. Ako se interval prekorači, komanda će biti ubijena, a upozorenje logovano.

Vremenska kontrola komande procesora podataka o performansama

Format: `perfdata_timeout=<br_sekundi>`

Primer: `perfdata_timeout=5`

Podešava maksimalnu dužinu intervala u sekundama koju će Nagios dati komandi procesora podataka o performansama da se izvršava. Ako se interval prekorači, komanda će biti ubijena, a upozorenje logovano.

Direktiva *Obsess Over Services*

Format: `obsess_over_services=<0/1>`

Primer: `obsess_over_services=0`

Ova direktiva određuje da li će Nagios "opsesivno" proveravati rezultate provera servisa i pokretati prethodno definisane komande opsesivno kompulsivne obrade servisa. Ime direktive je dosta nesrećno, međutim autor nije smislio ništa pametnije povodom toga. Uključivanje ove opcije je korisno kod distribuiranog nadgledanja. Ako se ne vrši distribuirano nadgledanje, toplo se preporučuje njeno isključivanje.

0 = ne vrši se opsesivna provera rezultata provera servisa (*default*)

1 = vrši se opsesivna provera rezultata provera servisa

Definisanje komande opsesivno kompulsivnog procesora servisa

Format: `ocsp_command=<komanda>`

Primer: `ocsp_command=obsessive_service_handler`

Ovim se definiše komanda koja koja se izvršava nakon svake provere servisa, što može biti koriso kod distribuiranog monitoringa. Ova komanda izvršava se posle bilo koje komande *event* hendlera ili obaveštavanja. Argument komande je krako ime komande definisane u datoteci gde stoje definicije hostova. Maksimalna dužina intervala u kom je dozvoljeno izvršavanje ove komande zadato je `ocsp_timeout` opcijom.

Direktiva procesora podataka o performansama

Format: `process_performance_data=<0/1>`

Primer: `process_performance_data=0`

Direktiva određuje da li se vrši obrada podataka o performansama izvršenih provera.

0 = ne (*default*)

1 = da

Kontrola napuštenih provera servisa (*Orphaned service checks*)

Format: `check_for_orphaned_services=<0/1>`

Primer: `check_for_orphaned_services=0`

Ova direktiva daje mogućnost detekcije napuštenih provera servisa. Napuštene provere servisa su one provere koje su izvršene, ali tokom dužeg perioda nisu vratile nikakav rezultat. Pošto se nisu vratili nikakvi rezultati, ovakva provera servisa će biti vraćena i preraspoređena u red za čekanje. Ovo može da izazove zaustavljanje izvršavanja provera servisa. U normalnim uslovima to će se jako retko dešavati – može da se desi kada neki eksterni proces ili korisnik ubije proces

čiji se servis u tom trenutku proveravao. Ako se ova opcija uključi i utvrdi se da se rezultati za tu proveru nisu vratili, generisaće se poruka greške i izvršiti preraspoređivanje provere servisa. Ako se primete provere servisa za koje izgleda da nikad nisu preraspoređene, treba omogućiti ovu opciju i pogledati da li postoji neki zapis u log datoteci o napuštenim servisima.

0 = ne proverava se postojanje napuštenih provera servisa (*default*)

1 = proverava se postojanje napuštenih provera servisa

Kontrola "svežine" rezulta provera servisa

Format: `check_service_freshness=<0/1>`

Primer: `check_service_freshness=1`

Ova direktiva kontroliše da li Nagios periodično proverava "svežinu" vraćenih rezultata provera servisa. Ova opcija je korisna kada se želi da Nagios proverava da li su rezultati pasivnih provera servisa primljeni u zadatom vremenskom intervalu. Ukoliko to nije slučaj, rezultati se smatraju zastarelim, a servis bez obzira na rezultat dobija CRITICAL status.

0 = svežina rezultata provera se ne proverava

1 = svežina rezultata provera se proverava (*default*)

Definisanje itervala provere svežine servisa

Format: `freshness_check_interval=<br_sekundi>`

Primer: `freshness_check_interval=60`

Ovom direktivom podešava se učestalost proveravanja svežine rezultata provere servisa. Ako je kontrola svežine rezultata provera onemogućena, ova direktiva nema efekta.

Definisanje ilegalnih karaktera imena objekata

Format: `illegal_object_name_chars=<karakteri>`

Primer: `illegal_object_name_chars=~!$%^&*"|'<>?,()`

Ovom direktivom se dozvoljava definisanje ilegalnih karaktera tj. karaktera čije se korišćenje zabranjuje u imenima hostova, opisima servisa ili imenima drugih tipova objekata. Nagios dozvoljava većinu karaktera u definicijama objekta, ali upotreba karaktera iz prikazanog primera se ne preporučuje zato što mogu da stvore probleme u web interfejsu, komandama obaveštenja itd.

Definisanje ilegalnih karaktera izlaza makroa

Format: `illegal_macro_output_chars=<karakteri>`

Primer: `illegal_macro_output_chars=~$^&"|'<>`

Ova direktiva dozvoljava da se definišu ilegalni karakteri koji treba da se izbace iz makroa pre nego što se iskoriste u obaveštenjima, *event* hendlerima i ostalim komandma. Ovo **ne** utiče na makroe koji se koriste u komandama provere servisa ili hostova. Može se izabrati da se karakteri prikazani u gornjem primeru ne izbacuju, ali se savetuje da se to ne čini. Neki od ovih karaktera se mogu interpretirati u konzoli i predstavljaju potencijalne siguronosne probleme. Sledeći makroi su oslobođeni definisanih karaktera: `$OUTPUT$`, `$PERFDATA$`

Definisanje email adrese administratora Nagiosa

Format: `admin_email=<email_adresa>`

Primer: `admin_email=root@gandalf.etf.bg.ac.yu`

Ovom direktivom se definiše email adresa administratora Nagiosa, tj. lokalne mašine. Umesto eksplicitnog navođenja u komandama za obaveštavanje, ova adresa se može koristiti pomoću `$ADMINEMAIL$` makroa.

Definisanje pejdžera administratora

Format: `admin_pager=<br_pejdžera_ili_pejdžer_email_gateway>`

Primer: `admin_pager=pager_root@gaandalf.etf.bg.ac.yu`

Ovom direktivom se definiše broj pejdžera administratora Nagiosa ili email adresa naloga na pejdžing gejtveju. Umesto eksplicitnog navođenja u komandama za obaveštavanje, ovaj broj/adresa se može koristiti pomoću `$ADMINEPAGER$` makroa.

Konfiguraciona datoteka CGI skriptova – cgi.cfg

Podrazumevani naziv konfiguracione datoteke CGI skriptova je `cgi.cfg`. U ovoj datoteci se definišu parametri kojim se utiče na rad CGI skriptova.

Lokacija glavne konfiguracione datoteke

Format: `main_config_file=<ime_datoteke>`

Primer: `main_config_file=/etc/nagios/nagios.cfg`

Ova direktiva definiše lokaciju glavne konfiguracione datoteke. CGI skriptovi moraju da znaju gde se nalazi ova datoteka kako bi došli do konfiguracionih informacija, tekućeg statusa hostova i servisa, itd.

Fizička HTML putanja

Format: `physical_html_path=<putanja>`

Primer: `physical_html_path=/usr/nagios/share`

Ova direktiva definiše fizičku putanju do direktorijuma na lokalnoj mašini u kome su smeštene Nagiosove HTML stranice. Nagios podrazumeva da su dokumentacija i datoteke slika (koje koriste CGI skriptovi) smešteni u poddirektorijumima ove putanje, `docs/` i `images/` respektivno.

URL HTML putanja

Format: `url_html_path=<putanja>`

Primer: `url_html_path=/nagios`

Ako se prilikom pristupanja Nagiosu preko web pretraživača zada URL adresa `http://imeračunara/nagios/` kao parametar HTML putanje u ovoj direktivi bi trebalo zadati putanju `/nagios`. U osnovi, to je deo URL adrese koji se koristi za pristup Nagiosovim HTML stranicama.

Definisanje komanda provere Nagios procesa

Format: `nagios_check_command=<komanda>`

Primeri: `nagios_check_command=/usr/nagios/libexec/check_nagios /var/nagios/status.log 5 '/usr/nagios/bin/nagios -d /etc/nagios/nagios.cfg'`

```
nagios_check_command=/usr/local/nagios/libexec/check_nagios
/usr/local/nagios/var/status.log 5 '/usr/local/nagios/bin/nagios -d
/usr/local/nagios/etc/nagios.cfg'
```

Ovo je opciona direktiva koja definiše komadu za proveru statusa Nagios procesa. Koriste je CGI skriptovi pomoću koje su "svesni" da li se Nagios još uvek izvršava ili je iz nekog razloga zaustavljen. Ako se u konfiguracionoj datoteci ne pojavi ova direktiva, CGI skriptovi će podrazumevati da se Nagios proces izvršava, pri čemu zadavanje komandi preko web interfejsa neće funkcionisati. Ako status Nagios procesa ipak treba da se prati, prilikom definisanja komande provere treba imati na umu da je ona u osnovi bazirana na `check_nagios` plaginu sa kojim se postupa isto kao i sa ostalim standardnim plaginovima. U slučaju da komanda vrati non-OK status, CGI skriptovi će smatrati da se Nagios proces ne izvršava i odbiće svaki pokušaj izvršavanja komandi preko *command* CGI skripta.

Kontrola autentifikacije korisnika

Format: `use_authentication=<0/1>`

Primer: `use_authentication=1`

Ova direktiva kontroliše autorizovanu upotrebu CGI skriptova i funkcionalnost autorizacije kada se određuje kojim informacijama i komandama korisnici imaju pravo da pristupe. Toplo se preporučuje korišćenje autentifikacije za CGI skriptove. Ako se autentifikacija ipak isključi, obavezno treba ukloniti *command* CGI skript kako bi se neautorizovani korisnici sprečili da izdaju komande Nagiosu. CGI skriptovi neće izdavati komande Nagiosu ako je autentifikacija isključena, ali zbog potpune sigurnosti savetuje se njihovo potpuno uklanjanje.

Definisanje *default* korisničkog imena

Format: `default_user_name=<username>`

Primer: `default_user_name=guest`

Definisanjem ove direktive definiše se *default* korisnik sa pravom pristupa CGI skriptovima. Ovo omogućava korisnicima iz sigurnog domena (npr. iza *firewall*-a) da pristupaju CGI skriptovima bez neophodne autentifikacije kroz web server. Ovo se može koristiti kako bi se izbegla bazična autentifikacija u slučaju korišćenja neosiguranog servera, budući da ovaj način autentifikacije prenosi lozinku kao običan tekst, bez enkripcije.

Važno je napomenuti da se definisanje *default* korisnika ne preporučuje osim ako se web interfejsu ne pristupa sa sigurnog web servera i ako je sigurno da su svi koji imaju pristup CGI skriptovima prethodno autentifikovani na neki način! Definisanjem ove direktive, svako ko nije autentifikovan kroz web server prosto će naslediti sva prava koja poseduje ovaj korisnik!

Kontrola pristupa informacijama o sistemu/procesu

Format: `authorized_for_system_information=<user1>,...<usern>`

Primer: `authorized_for_system_information=nagiosadmin,rt,zoran`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji po autentifikovanju krpz web interfejs mogu da vide informacije o sistemu/procesu pomoću *Extended information* CGI skripta. Korisnici na ovoj listi nisu automatski autorizovani za izdavanje sistem/proces komandi. Ako se želi da ovi korisnici to mogu, neophodno je dodati ih u direktivi `authorized_for_system_commands`.

Kontrola pristupa komandama sistem/procesa

Format: `authorized_for_system_commands=<user1>,...<usern>`

Primer: `authorized_for_system_command3=nagiosadmin,rt,zoran`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji mogu da izdaju komande sistem/procesu putem *command* CGI skripta. Korisnici na ovoj listi nisu automatski autorizovani za uvid u sistem/proces informacije. Ako se želi da im se to omogući neophodno je dodati ih na listu prethodno objašnjene direktive.

Kontrola pristupu konfiguracionim informacijama

Format: `authorized_for_configuration_information=<user1>,...<usern>`

Primer: `authorized_for_configuration_information=nagiosadmin,rt`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji imaju pravo uvida u konfiguracione informacije iz *configuration* CGI skripta. Korisnici sa ove liste mogu da pristupe informacijama o svim konfigurisanim hostovima, grupama hostova, servisima, kontaktima, grupama kontakta, periodima vremena i komandama.

Kontrola pristupa globalnim informacijama o hostovima

Format: `authorized_for_all_hosts=<user1>,...<usern>`

Primer: `authorized_for_all_hosts=nagiosadmin,zoran,krajko`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji imaju uvid u status i konfiguracione informacije svih hostova. Korisnici sa ove liste automatski su autorizovani za uvid u informacije o svim servisima. Korisnici sa ove liste nemaju automatski pravo da izdaju komande za sve hostove ili servise.

Kontrola pristupa globalnim komandama za hostove

Format: `authorized_for_all_host_commands=<user1>,...<usern>`

Primer: `authorized_for_all_host_commands=nagiosadmin,krajko`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji mogu da izdaju komande za sve hostove putem *command* CGI skripta. Korisnici sa ove liste su automatski autorizovani za izdavanje komandi za sve servise. Pri tom nisu autorizovani za uvid u konfiguracione informacije za sve servise ili hostove.

Kontrola pristupa globalnim komandama za servise

Format: `authorized_for_all_services=<user1>,...<usern>`

Primer: `authorized_for_all_services=nagiosadmin,zoran`

Ovom direktivom se definiše lista zapetom razdvojenih imena korisnika koji imaju uvid u status i konfiguracione informacije o svim servisima. Korisnici sa ove liste nisu automatski autorizovani za uvid u status i konfiguracione informacije o svim hostovima.

Definisanje pozadine za *statusmap* CGI skript

Format: `statusmap_background_image=<gd2_slika>`

Primer: `statusmap_background_image=statusmapbg.gd2`

Ovom direktivom se omogućava definisanja datoteke slike koja se koristi kao pozadina u *statusmap* CGI skriptu. Podrazumeva se da datoteka ove slike zaista postoji na putanji HTML slika (npr. `/usr/nagios/share/images`). Putanja se automatski određuje dodavanjem "`/images`" putanji definisanoj direktivom za definisanje fizičke putanje do HTML stranica.

Definisana slika mora biti u GD2 formatu (poželjno bez kompresije)!

Definisanje podrazumevanog stila prikaza statusmape

Format: `default_statusmap_layout=<br_stila>`

Primer: `default_statusmap_layout=4`

Ova direktiva omogućava definisanje podrazumevanog (*default*) stila koji koristi *statusmap* CGI skript. Važeće opcije su date u sledećoj tabeli:

vrednost *<layout_number>*	stil prikaza
0	korisnički definisane koordinate
1	nivoi dubine
2	nerazgranato drvo
3	razgranato drvo
4	cirkularno
5	cirkularno (*marked up*)
6	cirkularno (*baloon*)

Kontrola uključenja korisničkih objekata pomoću statuswrl CGI-a

Format: `statuswrl_include=<vrml_datoteka>`

Primer: `statuswrl_include=myworld.vrl`

Ova direktiva omogućava uključivanje korisnički definisanih objekata u generisani VRML svet. Podrazumeva se da zadata datoteka postoji na putanji definisanoj fizičkom HTML putanjom.

Napomena: Ovaj datoteka mora da bude potpuno definisani VRML svet (koji se može videti pomoću VRML brauzera).

Definisanje podrazumevanog stila prikaza statuswrl CGI skriptom

Format: `default_statuswrl_layout=<br_stila>`

Primer: `default_statuswrl_layout=4`

Ova direktiva omogućava definisanje podrazumevanog stila prikaza koji koristi *statuswrl* CGI skript. Važeće opcije su date u sledećoj tabeli:

<layout_number>	stil prikaza
0	korisnički definisane koordinate
2	nerazgranato drvo
3	razgranato drvo
4	cirkularno

Kontrola brzina osvežavanja stranica generisanih CGI skriptovima

Format: `refresh_rate=<br_sekundi>`

Primer: `refresh_rate=90`

Ova direktiva definiše vremenski interval u sekundama nakon kog se stranica generisana *status*, *statusmap* i *extinfo* CGI skriptovima ponovo učitava, tj. osvežava se.

Definisanje zvučnih alarma

Formati:
```
host_unreachable_sound=<zvučna_datoteka>
host_down_sound=<zvučna_datoteka>
service_critical_sound=<zvučna_datoteka>
service_warning_sound=<zvučna_datoteka>
service_unknown_sound=<zvučna_datoteka>
```

Primeri:
```
host_unreachable_sound=hostu.wav
host_down_sound=hostd.wav
service_critical_sound=critical.wav
service_warning_sound=warning.wav
service_unknown_sound=unknown.wav
```

Predstavljene direktive dozvoljavaju definisanje audio datoteka koje se interpretiraju u web brauzeru ako se dogodi neki problem dok se posmatra stranica generisana pomoću *status* CGI skripta. Podrzumeva se da su audio datoteke smeštene u `media/` poddirektorijumu na fizičkoj HTML putanji.

Definisanje sintakse komande ping preko WAP interfejsa

Format: `ping_syntax=<komanda>`

Primeri: `ping=/bin/ping -n -U -c 5 $HOSTADDRESS$`

Definiše sintaksu komande koja bi trebalo da se koristi pri pokušaju pingovanja hosta kroz WAP interfejs (korišćenjem statuswrml CGI skripta neophodno je uključivanje potpune putanje do izvršnog koda komande ping zajedno sa svim zahtevanim opcijama). `$HOSTADDRESS$` makro je zamenjen adresom hosta pre izvršavanja komande.

Risors konfiguraciona datoteka – resource.cfg

Podrazumevani naziv risors konfiguracione datoteke je `resource.cfg`. U ovoj datoteci se čuvaju osetljive konfiguracione informacije u formi korisnički definisanih makroa. U glavnoj konfiguracionoj datoteci može se definisati i više njih ako je tako nešto potrebno.

Definisanje $USERn$ korisničkih makroa

Format: `$USER<n>$=<vrednost_makroa>`

Primeri:
```
$USER1$=/usr/nagios/libexec
$USER2$=/usr/nagios/libexec/eventhandlers
$USER3$=n4gi0s
```

U risors datoteci može se definisati do 32 korisnički definisana makroa (`$USER1$` do `$USER32$`). Ovi makroi se u konfiguracionim datotekama mogu koristiti u razne svrhe. Jedna namena ovih makroa je kraće notiranje nekih često korišćenih putanja

u konfiguracionim datotekama. Takođe, ako bi u nekom trenutku odlučili da promenimo putanju izvršnih datoteka plaginova ili *event* hendlera, dovoljno bi bilo ažurirati samo putanje u risors datoteci.

U sledećem primeru prikazan je makro `$USER2$` onako kako sam koristio makroe u definicijama objekata komandi umesto eksplicitnog navođenja putanja do direktorijuma sa plaginovima ili *event* hendlerima. Umesto sledeće definicije komande *event* hendlera `restart-httpd`

```
define command{
      command_name  restart-httpd
      command_line  /usr/nagios/libexec/eventhandlers/restart-httpd
      $SERVICESTATE$ $STATETYPE$ $SERVICEATTEMPT$
}
```

koristio sam definiciju sa korisnički definisanim makroom (boldovano u primeru)

```
define command{
      command_name  restart-httpd
      command_line  $USER2$/restart-httpd
      $SERVICESTATE$ $STATETYPE$ $SERVICEATTEMPT$
}
```

Međutim, pravi razlog za korišćenje ovih makroa nije kraća notacija komandi već zaštita osetljivih sistemskih informacija od neautorizovanog pristupa. Korisničkim makroima može se izbeći čuvanje korisničkih imena i lozinki u konfiguracionim datotekama kojima CGI skriptovi imaju pravo pristupa. Ovo je važno zato što za razliku od svih ostalih konfiguracionih datoteka, nije predviđeno da CGI skriptovi pristupaju risors datoteci pa se nad njom mogu postaviti strožije dozvole pristupa. Nagios neposredno pre izvršavanja komande sa makroom zamenjuje makro njegovom vrednošću iz risors datoteke.

Definisanje parametara za pristup bazi podataka

Format: `<direktiva_za_bazu_podataka>=<vrednost>`

Primer:

```
# DB STATUS DATA
xsddb_host=<host>
xsddb_port=<port>
xsddb_database=<baza_podataka>
xsddb_username=<korisničko_ime>
xsddb_password=<lozinka>
xsddb_optimize_data=1
xsddb_optimize_interval=3600
```

Ukoliko je Nagios preveden sa podrškom za skladištenje konfiguracionih direktiva u bazi podataka, u ovoj datoteci se mogu naći i direktive sa korisničkim imenima i lozinkama za pristup određenim tabelama jedne ili više baza podataka.

Pri tom korisničko ime koje se ovde specificira mora da ima SELECT, INSERT, UPDATE i DELETE privilegije nad 'programstatus', 'hoststatus' i 'servicestatus' tabelama u bazi.

U mojoj realizaciji, Nagios nije prveden sa podrškom za skladištenje konfiguracionih podataka u bazi podataka, te se ove direktive nisu našle u mojoj konfiguracionoj datoteci.

Konfiguracione datoteke objekata

Kao što je pomenuto u uvodnom pregledu konfiguracije Nagiosa, svi podaci o nadgledanoj mreži se smeštaju ili u bazu podataka ili u konfiguracione datoteke objekata. U ovom poglavlju biće opisan način definisanja svih dvanaest tipova objekata *template based* metodom. Takođe biće opisan način organizacije datoteka na primerima koje sam koristio za nadgledanje fakultetske mreže.

Formati definicija objekata

Definicija objekta komande (*checkcommands*)

Format:
```
define command{
        [template       <ime_obrasca_koji_se_koristi>]
        [name           <ime_obrasca_za_dalje_korišćenje>]
        command_name    <ime_komande>
        command_line    <komanda_u_komandnoj_liniji>
        }
```

Primeri:
```
define command{
        command_name    check_pop3
        command_line    $USER1$/check_pop3 -H $HOSTADDRESS$
        }

   define command{
        command_name    notify-by-email
        command_line    /usr/bin/printf "%b"
"***** Nagios  *****\n\n
Notification Type: $NOTIFICATIONTYPE$\n\n
Service: $SERVICEDESC$\n
Host: $HOSTALIAS$\n
Address: $HOSTADDRESS$\n
State: $SERVICESTATE$\n\n
Date/Time:$DATETIME$\n\n
Additional Info:\n\n $OUTPUT$"
| /bin/mail -s "** $NOTIFICATIONTYPE$ alert -
$HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
        }
```

Definicijom objekta komande pojedinačno se definišu komande koje Nagios može da prepozna i koristiti kao komande provere servisa, hostova ili komande *event* hendlera.

U definicijama hostova i servisa, komande se notiraju kratkim imenom objekta komande (`command_name`), a umesto njih Nagios izvršava komande definisane `command_line` direktivom. Takođe, pri navođenju komande iz komandne linije, u primeru se može videti preimućstvo korišćenja makroa. U prvom primeru, ako se komanda `check_pop3` izvršava kao komanda provere servisa pridruženog hostu `galeb`, Nagios će neposredno pre njenog izvršavanja, umesto makroa `$HOSTADDRESS$` staviti ime hosta `galeb`.

U drugom primeru data je definicija komande obaveštavanja, `notify-by-email`, sa nešto dužom komandom iz komandne linije i brojnim makroima. Ovde se jasno vidi ogromno pojednostavljenje konfiguracije korišćenjem ovako skraćenih zapisa komandi poput ove koja se inače koristi u svakoj definiciji

kontakta.

Direktive `template` i `name` su opcione i koriste se kod nasleđivanja osobina postojećeg objekta komande i kreiranja objekta koji će nekom drugom objektu komande poslužiti kao obrazac, respektivno. Princip nasleđivanja će naknadno biti detaljno objašnjen.

Definicija objekta nadležnog kontakta (*contact*)

Format:
```
define contact{
        contact_name                    <korisničko_ime>
        alias                           <alias>
        service_notification_period     <period_obavešt_za_servise>
        host_notification_period        <period_obavešt_za_hostove>
        service_notification_options    <vrste_obavešt_za_servise>
        host_notification_options       <vrste_obavešt_za_hostove>
        service_notification_commands   <ime_komande>
        host_notification_commands      <ime_komande>
        email                           <email_adresa>
        }
```

Primer:
```
define contact{
        contact_name                    zoran
        alias                           Zoran Milenkovic
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,u,r
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                   mizoran@galeb.etf.bg.ac.yu
        }
```

Definicijom objekta kontakta se pojedinačno definišu nadležni kontakti koji će biti obaveštavani u slučaju prelaska hostova ili servisa u određeni non-OK status.

Iz formata i primera mogu se videti značenja svih raspoloživih direktiva. Za svaki kontakt se posebno definišu filtri obaveštenja o problemima u određenom statusu, filtri validnih perioda obaveštavanja, komande obaveštavanja, i sve to, posebno za hostove, posebno za servise. Filtri obaveštenja se zadaju navođenjem početnih slova statusa hosta ili servisa za koji obaveštenje može da se pošalje.

U datom primeru, definisani kontakt `zoran` može da prima sva obaveštenja o događajima sa hostovima u svim statusima budući da su pod direktivom *`host_notification_options`* navedene sve moguće opcije koje zapravo predstavljaju početna slova statusa za koja je obaveštavanje konatakta dozovljeno (d=DOWN, u=UNREACHABLE, r=RECOVERY).

Definicija objekta grupe kontakata (*contact groups*)

Format:
```
define contactgroup{
        contactgroup_name <ime_grupe>
        alias             <alias_grupe>
        members           <lista_kontakata_članova_grupe>
```

```
        }
```

Primer:
```
define contact{
        contactgroup_name router-admins
        alias             administratori rutera
        members           krajko,zoran,nagios-admin
        }
```

Definicijom objekta grupe kontakta se definiše grupa nadležnih kontakata koji će biti obaveštavani u slučaju prelaska hostova i servisa u non-OK status. Lista kontakata članova grupe se zadaje listom zapetom odvojenih imena prethodno definisanih objekata kontakta. Ime ovako definisane grupe kontakata se dalje koristi u definicijama hostova i servisa.

Definicija objekta zavisnosti servisa (*service dependency*)

Format:
```
define servicedependency{
        host_name                       <host_od_kog_zavisimo>
        hostgroup_name                  <grp_host_od_koje_zavisimo>
        service_description             <servis_od_kog_zavisimo>
        dependent_host_name             <zavisni_host>
        dependent_service_description <zavisni_servis>
        execution_failure_criteria    <kriter_obustav_povera>
        notification_failure_criteria <kriter_obustav_obavešt>
        }
```

Primeri:
```
define servicedependency{
        host_name                       galeb,gandalf
        service_description             PING
        dependent_host_name             el
        dependent_service_description POP3
        execution_failure_criteria    n
        notification_failure_criteria u,c
        }

 define servicedependency{
        host_name                       galeb
        service_description             PING,CPU_LOAD
        dependent_host_name             el
        dependent_service_description *
        execution_failure_criteria    u,c
        notification_failure_criteria w,u,c
        }
```

Definicijom objekta zavisnosti servisa definiše se kontrola izvršavanja provere servisa i obaveštavanja o tom servisu u zavisnosti od statusa servisa na nekom drugom hostu ili servisa na drugim hostovima. Ovde je jako važno napomenuti da se u definiciji koristi isključivo ili *hostgroup_name* ili *host_name* direktiva!

Izvršavanje provera zavisnog servisa i/ili obaveštavanje o istom servisu se obustavljaju ukoliko se zadovolje kriterijumi iz odgovarajućih direktiva. Označavanje kriterijuma je identično kao kod definicije objekta kontakta.

U datim primerima prikazana je fleksibilnost definicija za kreiranje najrazličitijih zavisnosti servisa. U prvom slučaju, budući da se provera rada POP3 servisa na hostu `el` nema smisla ako ne rade *`check_nrpe`* i *`check_pop3`* plaginovi na serveru `galeb`, u slučaju nedostupnosti ovog servera (što je ekvivalentno CRITICAL i UNKNOWN statusu PING servisa na ovom serveru), obavešetnja o hostu `el` će biti potisnuta.

Ako izvršavanje provera i obaveštavanje o nekom servisu zavise od rada više hostova, može se navesti lista zapetom razdvojenih hostova. Ovde u prvom primeru je pored servera `galeb` u pokazne svrhe naveden nadgledajući host `gandalf`.

U drugom primeru, prikazana je zavisnost svih provera servisa na hostu `el` (oznaka '`*`' podrazumeva sve servise) od statusa servisa PING i opterećenja procesora (CPU_LOAD) na serveru `galeb`. U ovom slučaju, zavisnost je konfigurisana tako da se obaveštenja o svim servisima na hostu `el` potiskuju u slučaju pojave bilo kog non-OK statusa servisa PING ili CPU_LOAD na `galebu`, a da se provere obustavljaju u slučaju CRITICAL i UNKNOWN statusa.

Definicija objekta zavisnosti hosta (*host dependency*)

Format:
```
define hostdependency{
        host_name                       <host_od_kog_zavisimo>
        hostgroup_name                  <grp_host_od_koje_zavisimo>
        dependent_host_name             <zavisni_host>
        notification_failure_criteria   <kriter_obustav_obavešt>
        }
```

Primer:
```
define servicedependency{
        host_name                       galeb
        dependent_host_name             el
        notification_failure_criteria   u,c
        }
```

Za razliku od oprethodno opisanog objekta, definicijom objekta zavisnosti hosta definiše se kontrola obaveštavanja o hostu u zavisnosti od statusa nekog drugog hosta ili grupe hostova. Ovde je jako važno napomenuti da se u definiciji koristi isključivo ili *`hostgroup_name`* ili *`host_name`* direktiva!

Sve ostale opcije su potpuno analogne prethodno opisanom objektu i prosto neće biti ponovo objašnjavane.

Definicija objekta eskalacije servisa (*service escalation*)

Format:
```
define serviceescalation{
        host_name                       <ime_hosta>
        service_description             <ime_servisa>
        first_notification              <prvo_eskalirano_obavešt>
        last_notification               <posl_eskalirano_obavešt>
        contact_groups                  <grupa/e_kontakata>
        notification_interval           <interval_eskalir_obaveš>
        }
```

Primer:
```
define serviceescalation{
        host_name                       el
```

```
service_description             POP3
first_notification              2
last_notification               6
contact_groups                  linux-admins
notification_interval           10
}
```

Definicijom objekta eskalacije servisa definiše se kontrola eskaliranog obaveštavanja grupa kontakata.

Sve opcije su potpuno jasne iz formata, a detalji su razrađeni u poglavlju o eskalacijama obaveštenja na str.26

Definicija objekta eskalacije hosta (*host escalation*)

Format: `define hostescalation{`

```
host_name                       <ime_hosta>
first_notification              <prvo_eskalirano_obavešt>
last_notification               <posl_eskalirano_obavešt>
contact_groups                  <grupa/e_kontakata>
notification_interval           <interval_eskalir_obaveš>
}
```

Definicija objekta eskalacije obaveštenja o hostu je u potpunosti ista kao i definicija objekta eskalacije obaveštenja o servisu s jedinom razlikom da ovde iz očiglednih razloga ne postoji direktiva *service_description*.

Definicija objekta eskalacije grupe hostova (*hostgroup escalation*)

Format: `define hostgroupescalation{`

```
hostgruop_name                  <ime_grupe_hostova>
first_notification              <prvo_eskalirano_obavešt>
last_notification               <posl_eskalirano_obavešt>
contact_groups                  <grupa/e_kontakata>
notification_interval           <interval_eskalir_obaveš>
}
```

Definicija objekta eskalacije obaveštenja o grupi hostova je identična prethodnoj definiciji.

Definicija objekta grupe hostova (*host group*)

Format: `define hostgroup{`

```
hostgroup_name          <ime_grupe_hostova>
alias                   <alias_grupe>
contact_groups          <nadležne_grupe_kontakata>
members                 <lista_hostova_članova_grupe>
}
```

Primeri: `define hostgroup{`

```
hostgroup_name          router-hosts
alias                   Ruteri
contact_groups          router-admins
members                 rtr1,acc1,modem-server,modemsrv
}
```

```
define hostgroup{
        hostgroup_name          farm-server-hosts
        alias                   Produkcioni serveri ETF-a
        contact_groups          linux-admins
        members                 ns,pingvin,kondor,www1,proxy,
phone,news,ubbg,med,aleluja,ahil2,nagios,kiklop,gentoo,zmaj,
galeb,mps,kerber,polaris,zmaj14,merlin,kiklop2
        }
```

Definicijom objekta grupe hostova definiše se grupa mrežnih uređaja ili hosotva koji su slični po nekim svojim odlikama. Definicija podrazumeva definisanje imena grupe, liste hostova članova grupe i grupe/grupa kontakata koje se obaveštavaju u slučaju non-OK statusa nekog od članova grupe hostova.

U prvom primeru je data definicija grupe hostova *router-hosts* čiji su članovi fakultetski ruteri. U drugom primeru, prikazana je grupa servera koju čine produkcione mašine fakultetske mreže.

Preimućstvo definisanje grupa hostova će se tek videti iz primera definicije objekta servisa.

Definicija objekta hosta (*host*)

Format:
```
define host{
        [name                   <ime_obrasca_za_dalje_korišć>]
        [use                    <ime_obrasca_koji_se_koristi>]
        host_name               <ime_hosta>
        alias                   <alias_hosta>
        [parents                <ime_roditeljskog_hosta>]
        contact_groups          <nadležne_grupe_kontakata>
        check_command                 <komanda_provere_hosta>
        max_check_attempts            <maks_br_provera>
        notifications_enabled         <0/1>
        notification_interval         <br_jedinič_intervala>
        notification_period           <validni_period_obavešt>
        notification_options          <filter_obaveštenja>
        event_handler_enabled         <0/1>
        flap_detection_enabled        <0/1>
        process_perf_data             <0/1>
        retain_status_information     <0/1>
        retain_nonstatus_information  <0/1>
        [register                     <0/1>]
        }
```

Primeri:
```
define host{
        name                          switch-generic-template
        use                           generic-host-template
        parents                       switch-2948
        check_command                 check-host-alive
        register                      0
        }

define host{
```

```
host_name                       galeb
alias                           Studentski terminal server
parents                         farma-servera
contact_groups                  linux-admins
check_command                   check-host-alive
max_check_attempts              5
notifications_enabled           1
notification_interval           20
notification_period             24x7
notification_options            d,u,r
event_handler_enabled           1
flap_detection_enabled          1
process_perf_data               0
retain_status_information       0
retain_nonstatus_information    1
register                        1
}
```

Kako je objekat hosta jedan od centralnih Nagiosovih objekata, definicija hosta sadrži brojne direktive. Prve dve direktive su opcione, ali su veoma korisne i služe za kreiranje obrazaca za hostove koji mogu da se nasleđuju. U prvom primeru data je definicija jednog obrasca hosta (*switch-generic-template*) kojim je opisan generički svič na mreži fakulteta, a koji pak nasleđuje osobine iz obrasca generičkog hosta (*generic-host-template*). U drugom primeru data je jedna kompletna definicija hosta koja ne nasleđuje direktive.

Direktiva `parents` je takođe opciona, ali jako korisna jer se pomoću nje definiše hijerarhijski odnos hostova u mreži u odnosu na nadgledajući host. U prvom primeru je očigledno da su svi nadgledani svičevi iz perspektive Nagiosa iza sviča `switch-2948` te je ova direktiva ubačena u generički obrazac za svičeve kao njihova zajednička osobina.

Iza standardnih direktiva slede direktive koje su objašnjene u poglavlju o obaveštavanju (str. 24). Potom slede direktive kojim se definišu komanda lokalnog event hendlera, kontroliše detekcija flepovanja, pamćenja podataka o performansama, pamćenja statusnih i nestatusnih informacija o hostu, i na kraju, direktiva koja govori da li se radi o objektu hosta (`register 1`) ili samo obrascu (`register 0`). Ako se ova poslednja direktiva ne navede u definiciji, podrazumeva se da je definisan "pravi" host.

Definicija objekta servisa (*service*)

Format: `define service{`

```
[name                       <ime_obrasca_za_dalje_korišć>]
[use                        <ime_obrasca_koji_se koristi>]
host_name                   <ime_hosta>
hostgroup_name              <ime_grupe_hostova>
service_description         <ime_servisa>
contact_groups              <nadležne_grupe_kontakata>
is_volatile                       <0/1>
active_checks_enabled             <0/1>
passive_checks_enabled            <0/1>
```

```
        parallelize_check               <0/1>
        obsess_over_service             <0/1>
        check_freshness                 <0/1>
        event_handler_enabled           <0/1>
        [event_handler                  <ime_komande_e_hendlera>]
        check_command                   <komanda_provere_hosta>
        max_check_attempts              <maks_br_provera>
        notifications_enabled           <0/1>
        normal_check_interval           <normal_interval_provera>
        retry_check_interval            <interval_ponovn_provera>
        notification_period             <validni_period_obavešt>
        notification_options            <filter_obaveštenja>
        flap_detection_enabled          <0/1>
        process_perf_data               <0/1>
        retain_status_information       <0/1>
        retain_nonstatus_information    <0/1>
        [register                       <0/1>]
        }
```

Primeri:

```
define service{
        name                            critical-service-template
        active_checks_enabled           1
        passive_checks_enabled          0
        parallelize_check               1
        obsess_over_service             1
        check_freshness                 0
        event_handler_enabled           1
        flap_detection_enabled          1
        process_perf_data               0
        retain_status_information       0
        retain_nonstatus_information    1
        is_volatile                     0
        check_period                    24x7
        max_check_attempts              2
        normal_check_interval           3
        retry_check_interval            1
        notifications_enabled           1
        notification_interval           60
        notification_period             24x7
        notification_options            w,u,c,r
        register                        0
}
define service{
        use                             critical-service-template
        hostgroup_name                  dns-servers
        service_description             DNS by nslookup
        contact_groups                  linux-admins
        check_command                   check_dns!147.91.8.7
}
```

Objekat servisa je jednako važan kao objekat hosta i njegova definicija se sastoji od najviše direktiva. Veliki broj ovih direktiva se ponavlja iz definicije hosta i ima isto značenje te ću ovde objasniti samo one direktive koje se ne pojavljuju u definiciji hosta.

Direktivom `service_description` se zadaje ime objekta servisa koji je pridružen nekom hostu, odnosno na njemu se izvršava. Obzirom da postoji potreba da se isti servis pridruži većem broju hostova (npr. PING), iza `host_name` direktive se može navesti lista zapetom razdvojenih imena hostova, a ako i to ne zadovoljava, servis se može pridružiti jednoj ili više grupa hostova pomoću direktive `hostgroup_name`.

Direktiva `obsess_over_service` se koristi kod nadgledanja distribuiranim Nagios serverima i na nivou definisanog servisa kontroliše podnošenje rezultata provera centralnom Nagios serveru. Prethodno, u glavnoj konfiguracionoj datoteci mora biti omugućena ta ista direktiva kojom se ova opcija kontroliše na nivou čitavog programa.

Ukoliko se koriste pasivne provere servisa na regularnoj vremenskoj bazi, direktivom `check_freshness` se može uključiti provera zastarelosti rezultata provera servisa.

Na kraju, direktivom `is_volatile` određuje se da li je servis "nepostojan" ili ne. Ova opcija se koristi samo kod nekih specifičnih servisa (napadi na sistem, zaglavljivanje nekog važnog procesa na sistemu i sl.) kod kojih se zahteva da se po ulasku u non-OK status odmah pređe u HARD stanje greške. Smisao prelaska u HARD stanje greške je pozivanje odgovarajućeg *event* handlera koji je obično komanda koja treba odmah da reaguje i pokuša da reši problem (npr. privremeno zatvori neke portove na sistemu, restartuje zaglavljeni proces i sl.).

Ovde su data dva primera definicije servisa gde je prvi primer zapravo definicija obrasca, a drugi definicija DNS servisa koji iz datog obrasca nasleđuje većinu direktiva.

Definicija objekta vremenskog perioda (*time period*)

Format:
```
define timeperiod{
        timeperiod_name   <ime_perioda>
        alias             <alias_za_period>
        [sunday           00:00-24:00]
        [monday           00:00-24:00]
        [tuesday          00:00-24:00]
        [wednesday        00:00-24:00]
        [thursday         00:00-24:00]
        [friday           00:00-24:00]
        [saturday         00:00-24:00]
        }
```

Primeri:
```
define timeperiod{
        timeperiod_name   24x7
        alias             24 sata dnevno, 7 dana u nedelji
        sunday            00:00-24:00
        monday            00:00-24:00
        tuesday           00:00-24:00
        wednesday         00:00-24:00
        thursday          00:00-24:00
```

```
        friday              00:00-24:00
        saturday            00:00-24:00
        }
    define timeperiod{
        timeperiod_name     nikad
        alias               period bez validnih intervala
        }
```

Definicija objekta vremenskog perioda sadrži direktive kojim se zadaju ime objekta vremenskog perioda, njegov alias i opsezi validnih intervala svakog dana u nedelji. Ime svakog dana predstavlja opcionu direktivu.

U datim primerima predstavljene su dve definicije perioda. Prva definiše period `24x7` koji se javlja u gotovo svim primerima defuinicija prethodno opisanih objekata, dok druga definiše period bez validnih intervala. Ovakav interval bi mogao da se koristi u definiciji servisa koji ne treba nikad da se proverava, a ovakav servis bi pak mogao da se pridruži nekom virtuelnom hostu koji formalno zahteva da mu se pridruži bar jedan servis.

Konfiguracione datoteke dodatnih informacija

Dodatne informacije sastoje se od opcionih definicija hostova i servisa koje koriste CGI skriptovi i prevashodno obezbeđuju vizuelno udobniji web interfejs programa. Definicije dodatnih servisa ne utiču na funkcionalnost Nagiosa i uvedene su da bi:

- u URL-ovima prikazali dodatne informacije o hostovima ili servisima,
- preko web interfejsa hostove i servise prikazali prigodnim ikonicama,
- iscrtali hostove na status mapama po korisnički definisanim 2D ili 3D koordinatama.

Kao kod definisanja objekata u Nagiosu, postoji nekoliko različitih metoda za čuvanje definicija dodatnih informacija.

- *default* (stari) metod – definicije se čuvaju u konfiguracionoj datoteci CGI skriptova. Ovaj metod je zadržan samo zbog kompatibilnosti unazad. Savetuje se upotreba drugih metoda.
- metod na bazi obrazaca (*template based*) – definicije se čuvaju u odvojenim konfiguracionim datoetkama. Grupe hostova ili servisa se definišu lako i brzo.
- Metod baze podataka – definicije se čuvaju u bazi podataka. Ukoliko se želi ovaj metod, Nagios je potrebno prevesti sa odgovarajućim flegovima u konfiguracionom skriptu.

Nasleđivanje

U *template-based* metodu konfigurisanja ugrađen je koncept nasleđivanja. Pod ovim se podrazumeva nasleđivanje svih direktiva iz definicije nekog objekta pri definisanju nekog drugog, novog objekta. Definicija objekta koja ne predstavlja realan objekat već se koristi samo za prenošenje osobina na druge objekte naziva se obrazac (*template*), pri čemu obrazac može da sadži i nepotpun skup direktiva. Ova metoda je bazirana na rekurzivnom parsiranju konfiguracionih datoteka. U ovu svrhu koriste se direktive `name`, `use` i `register`.

Ovom metodom se znatno olakšava inicijalno konfigurisanje i održavanje konfiguracije Nagiosa. Moguće je formirati objekat “obrazac” (*template object*) koji poseduje zajedničke osobine određene grupe hostova i potom, umesto kompletnih definicija hostova, pomoću direktive `use` u novoj definiciji objekta

samo deklarisati obrazac koji se koristi i navesti samo one direktive koje su karakteristične za taj objekta.

U slučaju višestrukog definisanja direktiva (i u obrascu i u definiciji objekta), Nagios uvek uzima u obzir direktivu koja je navedena u definiciji objekta dok se direktiva iz obrasca zanemaruje. Nagios dozvoljava definisanje obrazaca i sa nepotpunim skupom direktiva ukoliko se u krajnjoj definiciji objekta dodefinišu sve nedostajuće obavezne direktive.

Nasleđivanje osobina objekata se može vršiti i lančano, u više nivoa. Najpre se može definisati najopštiji objekat sa osobinama koje su zajedničke za sve ili veliku većinu objekata. Potom se iz ovog najopštijeg obrasca mogu izvesti manje opšti obrasci koji nasleđuju osobine opšteg obrasca i definišu one direktive koje su karakteristične za određene kategorije objekata. Ove kategorije se dalje mogu deliti na potkategorije definisane specijalinijim obrascima dok se na kraju ne dođe do definicije samog objekta. Definicija je tada jako kratka i jednostavana jer se većina direktiva nasleđuje kroz lanac obrazaca. Ukoliko se želi promena neke od nasleđenih direktiva, dovoljno je u definiciju uneti tu direktivu sa novom vrednošću i objekat će promeniti tu osobinu.

Primeri

Ovde ću dati primere korišćenja nasleđivanja u mojoj konfiguraciji Nagiosa. Nasleđivanje sam koristio kod definisanja objekata hostova i servisa. Ove definicije čine gro konfiguracije i bez upotrebe koncepta nasleđivanja, konfiguracija bi bila jako nepregledna i teška za održavanje.

Kao prvi primer upotrebe nasleđivanja navešću definicije objekata servisa. Kako definicije objekata servisa sadrže brojne direktive koje se ponavljaju tj. imaju iste vrednosti, prirodno rešenje je da se definiše obrazac generičkog objekta servisa čije bi direktive nasledili svi ostali objekti servisa. Obzirom da status DNS servisa i status PING servisa na nekoj studentskoj radnoj stanici nije jednako važan, odlučio sam da definišem tri generička obrasca – generičke objekte kritičnih, uobičajenih i nekritičnih servisa. Ovi obrasci se razlikuju po normalnom intervalu provere, broju ponovnih pokušaja provera pre podizanja alarma, filterima obaveštenja, kao i intervalima obaveštavanja usled perzistiranja problema. U primerima su opisane samo relevantne direktive sa komentarima.

U mojoj konfiguraciji definicije kritičnih servisa poput DNS servisa, PING glavnog rutera i sl. nasleđuju direktive iz obrasca *critical-service-template*.

```
define service{
      # ime obrasca
      name                            critical-service-template
      # u slučaju non-OK rezultata provere servisa dozvoljen je
      # samo još jedan pokušaj provere pre podizanja alarma.
      max_check_attempts              2
      # kritični servisi se normalno proveravaju na svakih 3 minuta
      normal_check_interval           3
      # u slučaju non-OK rezultata provere, sledeća provera servisa
      # se izvršava u roku od 1 minuta.
      retry_check_interval            1
      # interval slanja obaveštenja u slučaju perzistiranja
      # problema je postavljen na 1 čas (60 min.)
      notification_interval           60
      # pošto se radi o kritičnim servisima slanje obaveštenja je
      # uvek dozvoljeno
```

```
        notification_period             24x7
        # zdravlje ovih servisa su kritično za rad mreže tako da je u
        # filteru obaveštenja dozvoljeno podizanje alarma u slučaju
        # bilo kakve promene statusa servisa
        notification_options            w,u,c,r
        ... ostale direktive ...
        }
```

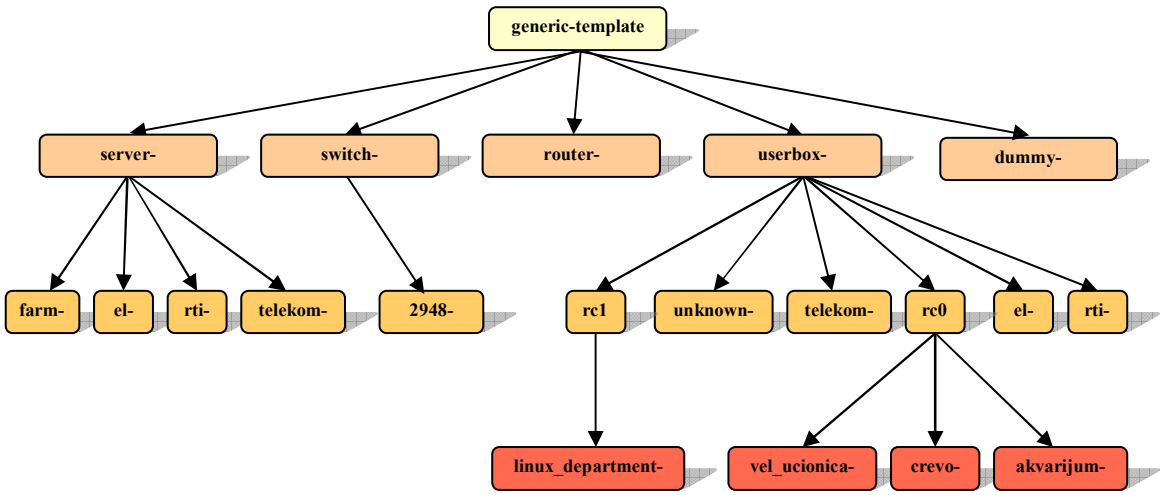
Za uobičajene servise uzete su provere POP3, FTP, SMTP, HTTP i sličnih servisa na produkcionim fakultetskim serverima. Definicije ovih servisa nasleđuju *default-service-template*.

```
define service{
        name                            default-service-template
        # ovde, za razliku od kritičnih servisa dozvoljavamo
        # maksimalno 3 provere servisa pre podizanja alarma.
        max_check_attempts              3
        # normalni interval provera ovih servisa je 5 minuta.
        normal_check_interval           5
        # u slučaju non-OK rezultata provere servisa želimo da što
        # pre ustanovimo da li je problem stvaran i ponavljamo
        # provere u intervalu od 1 minuta.
        retry_check_interval            1
        # ako problem nastavi da perzistira, obaveštavanje nadležnih
        # kontakata se nastavlja u intervalima od 6 časova.
        notification_interval           240
        # iako nisu kritični, i ovi servisi su dovoljno važni da se
        # omogući obaveštavanje tokom 24 č., 7 dana u nedelji.
        notification_period             24x7
        # filter obaveštenja dozvoljava samo obaveštenja o ulasku u
        # status CRITICAL i u slučaju oporavka servisa, RECOVERY.
        # Ostali statusi nam nisu interesantni
        notification_options            c,r
        ... ostale direktive ...
        }
```

Na kraju, provere PING servisa korisničkih radnih stanica, statusa hard diskova i sl. nasleđuju *noncritical-service-template*.

```
define service{
        name                            noncritical-service-template
        # u slučaju non-OK statusa nekritičnih servisa, dopuštamo čak
        # 5 ponovnih pokušaja provere servisa
        max_check_attempts              5
        # pošto ima jako mnogo nekritičnih servisa koje nadgledamo,
        # normalni interval provere ovakvih servisa je postavljen na
        # 45 minuta. Ovim je opterećenje nadgledajućeg servera
        # svedeno na minimum.
        normal_check_interval           45
        # u slučaju non-OK statusa, provere se ponavljaju u relativno
        # kratkom intervalu od 1 minuta kako bi što pre imali ažurnu
        # informaciju o statusu servisa.
        retry_check_interval            1
        # Pošto nisu od ključnog značaja sva obaveštenja o ovim
        # servisima su zabranjena.
        notifications_enabled           0
        ... ostale direktive ...
        }
```

U sledećem primeru dao sam nešto složeniju strukturu obrazaca koju sam uveo radi lakšeg definisanja i održavanja hostova u konfiguraciji. Svi hostovi nasleđuju obrazac generičkog hosta (`generic-template`), međutim konfiguracije određenih grupa hostova se dosta razlikuju pa sam grupisanje zajedničkih direktiva izvršio u nekoliko nivoa. Hostovi su najpre podeljeni prema funkcionalnoj (server, *switch*, *userbox* itd.), a potom i prema topološkoj pripadnosti (rti, el, rc itd.), tako da krajnji (specijalni) obrasci lančano nasleđuju direktive iz više opštijih obrazaca. Na slici 12. dato je stablo obrazaca u čijem korenu (na vrhu) se nalazi najopštiji obrazac generičkog hosta, a na čijim krajevima se nalaze krajnji obrasci koje direktno nasleđuju definicije hostova. U cilju lakšeg snalaženja, imena obrazaca sam formirana u zavisnosti od "putanje" kojom je izvođenje obrasca izvedeno tako što se pri prolasku kroz svaki čvor stabla, ispred imena obrasca doda njegovo ime. Na primer, za generički obrazac koji nasleđuju definicije produkcionih servera se dobija ime `farm-server-generic-template`.



Slika 12. Struktura definicija obrazaca hostova

Iako nije postojla realna potreba za nadgledanjem svih računara koji su uneti u konfiguraciju, jedna od ideja ovog rada bila je da se demonstrira skalabilnost Nagiosa, tj. jednostavnost konfiguracije kod nadgledanja velikog broja hostova.

Zaključak

Nagios je alat za mrežni monitoring koji mrežnom administratoru omogućava da nadzire rad računara, rutera, svičeva, štampača i servisa. Mnoge organizacije odlučuju se za skupa komercijalna rešenja kao što je *HP OpenView*. Međutim, Nagios je odlično rešenje za one kompanije koje traže proizvod koji nije skup (ili je besplatan) i koji je zbog otvorenog izvornog koda permanentno u fazi razvoja.

Trenutno, stabilna i potpuno dokumentovana verzija Nagiosa koja se može preuzeti sa interneta je verzija 1.1, mada se već koristi i alfa verzija 2.0. Razlog zbog kog verzija 2.0 još uvek nije ušla u široku upotrebu nisu toliko bagovi u programu koliko nedostatak potpune dokumentacije. U najavi je i verzija 3.0 koja će pored raznih drugih unapređenja, umesto CGI skirptova za web interfejs koristiti PHP, a čitava konfiguracija će se čuvati u bazi podataka.

Osnovna prednost Nagiosa je ista kao i njegov osnovni nedostatak: krajnja fleksibilnost. Ovo je očigledno odlično rešenje za one administratore koji žele da imaju potpunu kontrolu aplikacije i njenu fleksibilnost. To takođe podrazumeva višednevna testiranja, puno čitanja i podešavanja. Nagios se definitivno ne može instalirati po principu *clik-and-go*. Postoje brojni paketi koji su potrebni da bi Nagios radio ispravno.

Dodatak A – ručna instalacija Nagiosa

U ovom dodatku dat je postupak ručne instalacije jezgra Nagiosa i njegovih plaginova. Kao što je već prethodno rečeno, ovaj dodatak ima za cilj da čitaoca upozna sa detaljima instalacije paketa na bilo kojoj Linux platformi. Motivacija za pisanje ovog dodatka je pronađena i u činjenici da bi izneti postupak instalacije mogao da pomogne bolje razumevanje rada programa.

Čitav postupak instalacije sam sa uspehom izveo na svom kućnom računaru pod *Slackware* 9.1 distribucijom Linuxa.

Raspakivanje distribucije

Nakon dobavljanja *tarball* ili zip arhive sa zvaničnog web sajta, Nagios distribucija se raspakuje na sledeći način:

```
tar –xzvf nagios-1.1.tar.gz
```

ili u slučaju zip arhive,

```
gunzip nagios-1.1.zip
```

Kreiranje instalacionog direktorijuma

Osnovni direktorijum u kom će se Nagios instalirati se kreira na sledeći način:

```
mkdir /usr/local/nagios
```

Kreiranje Korisnika/Grupe

Nagios treba pokretati pod normalnim korisničkim nalogom, za šta je potrebno kreirati novog korisnika (i grupu) na sistemu pomoću sledeće komande (komande za dodavanje novog korisnika mogu da variraju u zavisnosti od operativnog sistema koji se koristi):

```
adduser nagios
```

Pokretanje konfiguracionog skripta

Da bi se inicijalizovale promenljive, kreirali Makefiles, itd., potrebno je pokrenuti konfiguracioni skript na sledeći način:

```
./configure prefix=<prefix>
            with-cgiurl=<cgiurl>
            with-htmurl=<htmurl>
            with-nagios-user=<someuser>
            with-nagios-grp=<somegroup>
```

pri čemu:

- `prefix` treba zameniti prethodno kreiranim instalacionim direktorijumom (po *default*-u to je direktorijum `/usr/local/nagios`)
- `cgiurl` treba zameniti konkretnom url adresom koja će se koristiti za pristupanje CGI skriptovima (po *default*-u to je `/nagios/cgi-bin`)

- `htmurl` treba zameniti konkretnom URL adresom kojom će se pristupati HTML dokumentima glavnog interfejsa kao i dokumentaciji (po *default*-u to je `/nagios`)
- `someuser` treba zameniti konkretnim imenom korisničkog naloga na sistemu koji će imati dozvole vlasnika nad instaliranim fajlovima Nagiosa (po *default*-u to je `nagios`)
- `somegroup` treba zameniti konkretnom imenom grupe kojoj pripada korisnički nalog na sistemu koji će imati dozvole vlasnika nad instaliranim fajlovima Nagiosa (po *default-u* to je `nagios`)

Prevođenje programa

Prevođenje (kompajliranje) Nagiosa i CGI skriptova vrši se pomoću standardnog programa `make`:

```
make all
```

Instalacija prevedenog programa i HTML stranica

Instalacija prevedenog programskog koda se obavlja sledećom komandom:

```
make install
```

Instalacija init skripta

Kada se program iskonfiguriše i istestira, veoma je zgodno instalirati inicijalizacioni skript koji će automatski pokretati Nagios pri inicijalizaciji sistema. Primer init skripta koji dolazi uz Nagios distribuciju bi trebalo da radi bez ikakvih izmena na svim uobičajenim Linux distribucijama. Ovaj init skript pod imenom `nagios` se instalira u odgovarajući direktorijum (na *Slackware*-u, `/etc/rc.d/`) sledećom komandom:

```
make install-init
```

Ukoliko se koristi specifičan operativni sistem, u skriptu će verovatno biti potrebno izvršiti određene promene.

Struktura direktorijuma i lokacija

Ako je instalacija prošla uspešno, posle komandi:

```
cd /usr/local/nagios
ls -l
```

trebalo bi da se vidi pet različitih poddirektorijuma. Kratak opis sadržaja svakog od direktorijuma dat je u tabeli ispod.

poddirektorijum	sadržaj
`bin/`	jezgro Nagios programa
`etc/`	konfiguracione datoteke (*main*, *resource*, *object* i *CGI*)
`sbin/`	CGI skriptovi
`share/`	HTML fajlovi (web interfejs i dokumentacija)
`var/`	prazan dir namenjen za smeštanje log

Instalacija plaginova

Da bi Nagios bio od bilo kakve koristi, neophodno je sa interneta preuzeti i instalirati Nagiosove plaginove. Plaginovi se standardno instaliraju u `/libexec` poddirektorijumu Nagios instalacije (npr. `/usr/local/nagios/libexec`). Plaginovi su skriptovi ili kompajlirani programi koji izvršavaju sve provere hostova i servisa. Razvoj Nagiosovih plaginova je neprekidan proces. Poslednje verzije plaginova mogu se preuzeti sa sledećih linkova:

1. http://www.nagios.org/download/
2. http://sourceforge.net/projects/nagiosplug/

Izvorni kod svih standardnih plaginova dolazi u okviru jedne *tarball* datoteke, kao što je ovde na primer, `nagios-plugins-1.3.1.tar.gz`. Prevođenje i instalacija plaginova se vrši na potpuno analogan način kao i instalacija Nagios jezgra.

Primer instalacije plaginova sa *default* parametrima:

```
tar –xzvf nagios-plugins-1.3.1.tar.gz
cd nagios-plugins-1.3.1
./configure --prefix=/usr/local/nagios
            --with-nagios-user=nagios
            --with-nagios-group=nagios
            --with-cgiurl=/nagios/cgi-bin
make all
make install
```

Dodatni plaginovi se mogu naći u okviru `/contrib` poddirektorijuma distribucije plaginova, gde je data zbirka plaginova razvijena od strane zajednice programera i mrežnih administratora okupljenih oko Nagios projekta. Svi plaginovi se distribuiraju pod GPL licencom sa otvorenim kodom i kao takvi se lako mogu prilagođavati sopstvenim potrebama.

Podešavanje dozvola nad datotekom eksternih komandi

Da bi se omogućilo izdavanje komadi Nagiosu kroz web interfejs, kao i dozvoljavanje eksternim aplikacijama da Nagiosu izdaju komande, potrebno je ispravno podesiti dozvole nad ovom datotekom eksternih komandi. Dozvole treba da budu takve da se eksternim aplikacijama omogući upis u datoteku `nagios.cmd`. Pošto se ovadatoteka briše prilikom svakog zaustavljanja ili reinicijalizacije programa, potrebno je zapravo podesiti dozvole nad direktorijumom u kome se datoteka svaki put kreira, `/usr/local/nagios/var/rw`.

Postupak je sledeći. Prvo treba saznati korisnička imena pod kojim se izvršavaju web server i Nagios. Na većini sistema web server se izvršava pod korisničkim imenom `nobody`, što je i ovde pretpostavljeno. Pretpostavlja se takođe da se Nagios izvršava pod predefinisanim korisničkim imenom `nagios`. Zatim, treba kreirati grupu korisnika `nagioscmd` čiji će članovi biti samo korisnici `nagios` i `nobody`. Za izvršvanje ove i sledećih komandi potrebne su administratorske privilegije.

```
groupadd nagioscmd
usermod –G nagioscmd nagios
usermod –G nagioscmd nobody
```

Ako ne postoji direktorijum u kom će se kreirati datoteka eksternih komandi, listi komandi treba dodati i sledeću

```
mkdir /usr/local/nagios/var/rw
```

Promena vlasnika direktorijuma vrši se pomoću

```
chown nagios.nagioscmd /usr/local/nagios/var/rw
```

Davanje svih dozvola korisniku `nagios` nad direktorijumom ide sa

```
chmod u+rwx /usr/local/nagios/var/rw
```

Davanje svih dozvola grupi nad direktorijumom

```
chmod g+rwx /usr/local/nagios/var/rw
```

Konačno, da bi se sve novokreirane datoteke u direktorijumu primorale da naslede njegove dozvole grupe, treba setovati *sticky* bit grupe direktorijuma komandom

```
chmod g+s /usr/local/nagios/var/rw
```

Posle izvođenja svih izlistanih komandi, direktorijum `rw` će imati ispravno konfigurisane privilegije i trebalo bi da izgleda ovako:

```
drwxrws---  2 nagios nagioscmd      1024  May 25 16:30 rw
```

Dodatak B – podešavanje web interfejsa

Naredne instrukcije predstavljaju postupak konfiguracije web servera koji treba da obezbedi pristup Nagiosovom web interfejsu. U ovom opisu podrazumevano je da je na ciljnoj mašini instaliran Apache web server, verzija 1.3. Ako se koristi neki drugi HTTP server, verovatno će biti potrebno izvršiti odgovarajuće promene u konfiguracionim fajlovima. Takođe, podrazumeva se da je kao instalacioni prefiks korišćena putanja `/usr/local/nagios`.

Konfiguracija script alias-a za CGI skriptove

Nagiosov web interfejs se u potpunosti oslanja na svoje CGI skriptove. Da bi CGI skriptovi ispravno radili, potrebno je definisati putanju putanju do skriptova pomoću ScriptAlias direktive. Podrazumevana instalacija očekuje da su oni dostupni na adresi http://imeračunara/nagios/cgi-bin/, iako je to podložno promenama korišćenjem `--with-cgiurl=<cgiurl>` opcije u konfiguracionom skriptu. U konfiguracionoj datoteci Apache servera trebalo bi dodati nešto slično ovome:

```
ScriptAlias       /nagios/cgi-bin/ /usr/local/nagios/sbin/
<Directory        "/usr/local/nagios/sbin/">
  AllowOverride   AuthConfig
  Options         ExecCGI
  Order           allow,deny
  Allow           from all
</Directory>
```

Veoma je važno da ScriptAlias direktiva mora das stoji iznad Alias linije koja je objašnjena u daljem tekstu. U suprotnom Apache može drugačije (tj. pogrešno) da parsira ove linije. Zatim, ako se Nagios instalira na mašinu koju koristi mnogo korisnika, preporučljivo je korišćenje CGIwrap[5] alata kako bi se obezbedila dodatna sigurnost između CGI skriptova i datoteke u koju se upisuju eksterne komande.

Konfiguracija aliasa za HTML fajlove

Da bi se Nagiosove HTML strane učinile dopstupnim preko web servera, u Apache-ovu konfiguracionu datoteku `httpd.config` treba dodati sledeće linije:

```
Alias             /nagios/ /usr/local/nagios/share/
<Directory        "/usr/local/nagios/share">
  Options         none
  AllowOverride   AuthConfig
  Order           allow,deny
  Allow           from all
</Directory>
```

Ovo omogućava da se korišćenjem URL adrese http://imeračunara/nagios/ pristupi web interfejsu kao i dokumentaciji. Naravno, kao alias bi trebalo navesti isto ime koje je navedeno u opciji `--with-htmurl=<htmurl>` konfiguacionog skripta.

Konfiguracija autentifikacije na web serveru

Prvi korak u konfiguraciji autentifikacije na web serveru je uveravanje da konfiguraciona datoteka web servera (npr. httpd.conf) sadrži AuthOverride AuthConfig direktivu za CGI–BIN direktorijum Nagiosa. U suprotnom, neophodno je dodati sledeće linije konfiguracionom datoteci web servera. Podrazumeva se da je posle unošenja promena u konfiguraciju web servera potrebno restartovati ga da bi promene imale efekta.

```
<Directory        /usr/local/nagios/sbin>
  AllowOverride   AuthConfig
  Order           allow,deny
  Allow           from all
  Options         ExecCGI
</Directory>
```

Ako se zahteva autorizovani pristup HTML stranicama Nagiosa, potrebno je dodati sledeće linije.

```
<Directory        /usr/local/nagios/share>
  AllowOverride   AuthConfig
  Order           allow,deny
  Allow           from all
</Directory>
```

Sledeći korak je kreiranje datoteke po imenu .htaccess u korenu vašeg CGI direktorijuma (i opciono u korenu HTML direktorijuma) za Nagios. (ovi) bi trebalo da sadrže nešto slično sledećim linijama.

```
AuthName          "Nagios Access"
AuthType          Basic
AuthUserFile      /usr/local/nagios/etc/htpasswd.users
require           valid-user
```

Dodavanje autentifikovanih korisnika

Kada je web server tako iskonfigurisan da zahteva autentifikaciju za pristup CGI skriptovima, neophodno je dodati korisnike koji mogu da im pristupe. Ovo se vrši pomoću komande htpasswd koja dolazi zajedno sa Apache web serverom.

Ovom komandom kreira se nova datoteka po imenu `htpasswd.users` u `/usr/local/nagios/etc` direktorijumu. Takođe će se kreirati username/password unos za nagiosadmin korisnika. Pri tom će biti potrebno da se unese lozinka kojim će se nagiosadmin autentifikovati na web serveru.

```
htpasswd –c /etc/nagios/htpasswd.users nagiosadmin
```

Za dalje dodavanje korisničkih imena i lozinki za sve one kojima treba omogućiti pristup CGI skriptovima, koristi se ista komanda osim flega -c koji se koristi za kreiranje inicijalnog datoteke.

```
htpasswd /etc/nagios/htpasswd.users <korisničko_ime>
```

Time je završen prvi deo konfiguracije. Ako se sada u web pretraživaču zada adresa Nagios CGI skriptova, od korisnika će se zahtevati da unese validno korisničko ime i lozinku. Ako se pojave problemi sa autentifikacijom, potrebno je proučiti dokumentaciju o korišćenom web serveru.

Omogućavanje autentifikacije/autorizacije CGI skriptova

Sledeća stvar koju treba uraditi je konfiguracija CGI skriptova tako da zahtevaju autentifikaciju korisnika koji mogu da pristupaju informacijama ili zadaju određene komande. U konfiguracionom datoteci CGI skriptova potrebno je postaviti `use_authentication` promenljivu na nenultu vrednost. Primer:

```
use_authentication=1
```

Ovim je završena konfiguracija bazične autentifikacije CGI skriptova.

Podrazumevane dozvole za CGI informacije

U ovom delu se govori o *default* dozvolama koje poseduju autorizovani korisnici.

CGI Data	Authenticated Contacts	Other Authenticated Users
Host Status Information	Yes	No
Host Configuration Information	Yes	No
Host History	Yes	No
Host Notifications	Yes	No
Host Commands	Yes	No
Service Status Information	Yes	No
Service Configuration Information	Yes	No
Service History	Yes	No
Service Notifications	Yes	No
Service Commands	Yes	No
All Configuration Information	No	No
System/Process Information	No	No
System/Process Commands	No	No

Autentifikacija na sigurnim web serverima

Ako je web server lociran u sigurnom domenu (iza *firewall*-a) ili se koristi SSL, može se definisati *default* korisnik koji može da pristupa CGI skriptovima. To se vrši definisanjem `default_user_name` opcije u konfiguracionoj datoteci CGI skriptova. Definisanjem *default* korisnika korisnicima se dozvoljava da pristupaju CGI skriptovima bez neophodnog predhodnog autentifikovanja na web serveru. Takođe, može se koristiti ako se izbegava bazična web autentifikacija pošto ona prenosi lozinke u formatu čistog teksta.

Važno: Definisanje *default* korisnika treba izbegavati osim ako se ne radi na sigurnom web serveru i ako je svako ko ima pristupa CGI skriptovima prethodno autentifikovan na neki način! Ako se ova promenljiva definiše, bilo ko ko nije autentifikovan na web serveru naslediće dozvole pridružene ovom korisniku!

Restart web servera

Da bi unešene promene u konfiguracionoj datoteci Apache servera imale efekta, potrebna je njegova reinicijalizacija

```
/etc/rc.d/httpd restart
```

ili što je ekvivalentno

```
apachectl restart
```

Verifikacija promena konfiguracionog datoteke

Nakon ovih komandi još je potrebno uveriti se da li su promene učinjene u konfiguraciji zaista uzele efekta. Ako je sve dobro urađeno, trebalo bi da se pomoću web pretraživača otvori web interfejs Nagiosa zadavanjem sledeće adrese http://imeračunara/nagios/. CGI skriptovi verovatno neće moći da prikažu nikakve informacije, ali čim se pokrene iskonfigurisani Nagios, i ovo će biti rešeno.

Dodatak C – instalacija i konfiguracija *nrpe* dodatka za Nagios

U ovom dodatku opisao sam postupak instalacije i konfiguracije `nrpe` dodatka za Nagios na serveru `galeb` kojim je omogućeno indirektno izvršavanje provera servisa na hostovima čiji se servisi nisu videli sa nadgledajućeg hosta `gandalf`, kao i izvršavanje provera lokalnih privatnih servisa na samom serveru `galeb`. Ovaj server je bio jedini server na kom sam mogao da izvedem isntalaciju pošto sam još samo na njemu imao otvoren korisnički (studentski) nalog.

U trenutku instalacije na raspolaganju sam imao verziju programa 2.0, međutim zbog potencijalnih sigurnosnih problema na koje se upozorava u dokumentaciji najnovije verzije programa, korišćena je starija verzija 1.8. Naime, u kasnijoj verziji je omogućeno da *check_nrpe* plagin zajedno sa zahtevom izvršavanja udaljenog plagina prenosi i argumente komande pomoću novoimplementiranog flega `-a`. Ovim bi se svakako dosta pojednostavilo konfigurisanje indirektnih provera, ali bi onda sigurnost sistema bila značajno narušena.

Prevođenje

Izvorni kod programa je dostupan na zvaničnom sajtu Nagios projekta. Postupak prevođenja programa je jako jednostavan. Sastoji se od otpakivanja *tarball* datoteke, pokretanja konfiguracionog skripta i prevođenja programa. Komande su sledeće:

```
tar –xzvf nagios-nrpe_1.8.tar.gz
cd nrpe-1.8
./configure
make all
```

Ovde treba primetiti da se `check_nrpe` plagin i `nrpe` demon izvršavaju na različitim mašinama, tako da je u opštem slučaju potrebno izvršiti prevođenje i na nadgledajućem i na udaljenom serveru. Pošto su na serveru `gandalf` prevođenje i instalacija obavljeni automatski, uključujući i `nrpe` dodatak, ovim je postupak prevođenja na obe mašine završen.

Instalacija

Nakon završenog prevođenja na serveru `galeb`, izvršne datoteke `nrpe` i *check_nrpe* su bile smeštene u `src/` direktorijumu. Odavde ih je trebalo ručno instalirati u pogodne direktorijume. Pošto nije bilo mogućnosti za implementaciju višestruko ulančanih indirektnih provera, plagin *check_nrpe* nije bio korišćen. Demon `nrpe` sam instalirao u direktorijum `/users/etf/mizoran/nagios/bin`, dok sam njegovu jedinu konfiguracionu datoteku, `nrpe.cfg`, smestio u direktorijum `/users/etf/mizoran/nagios/etc`.

Naravno, prethodno sam morao da pod svojim korisničkim nalogom instaliram Nagiosove plaginove. Postupak je bio sledeći:

```
tar –xzvf nagios-plugins-1.3.1.tar.gz
cd nagios-plugins-1.3.1
```

```
./configure --prefix=/users/etf/mizoran/nagios
            --with-nagios-user=mizoran
            --with-nagios-group=etf
make all
make install
```

Modovi rada *nrpe* demona

Program nrpe se može pokrenuti na dva načina: kao samostalni (pozadinski) demonski program ili kao servis pod `inetd` ili `xinetd` superserverom.

Ako se izvršava u modu demona, `nrpe` agent vrši autentifikaciju zahteva za izvršavanjem plaginova prostim upoređivanjem IP adrese pozivajućeg hosta sa listom dozvoljenih IP adresa u konfiguracionoj datoteci.

Ukoliko se izvršava pod `inetd` pod `xinetd` superserverom, pristup `nrpe` agentu može da se vrši pomoću `TCP wrapper`-a. U tom slučaju, potrebno je uraditi sledeće:

- U konfiguracionoj datoteci `/etc/services` potrebno je dodati sledeću liniju sa odgovarajućim brojem porta koji se koristi za komunikaciju između *check_nrpe* plagina i `nrpe` demona

  ```
  Nrpe            5666/tcp        # NRPE
  ```

- U konfiguracionim datotekama `inetd` ili `xinetd` treba dodati `nrpe` demon. Ispod je dat postupak konfigurisanja za oba superservera. U ovom slučaju, direktive *server_port* i *allowed_hosts* u konfiguracionoj datoteci `nrpe.cfg` se ignorišu.

***** *INETD* *****

Ako se na sistemu koristi superserevr **sa** *tcpwrappers* alatom, u `/etc/inetd.conf` treba upisati sledeću liniju:

```
nrpe stream tcp nowait <korisnik> /usr/sbin/tcpd <nrpe_bin>
-c <nrpe_cfg> --inetd
```

Ako se na sistemu koristi superserevr **bez** *tcpwrappers* alata, u `/etc/inetd.conf` treba upisati sledeću liniju:

```
nrpe stream tcp nowait <korisnik> <nrpe_bin> -c <nrpe_cfg>
--inetd
```

Pri tom se namesto <korisnik>, <nrp_bin> i <nrpe_cfg> treba navesti kosriničko ime pod kojim će se nrpe izvršavati, putanja do izvršne datoteke programa i putanja do konfiguracione dastoteke, respektivno. Na primer:

```
nrpe stream tcp nowait mizoran
/usr/sbin/tcpd /users/etf/mizoran/nagios/bin/nrpe
-c /users/etf/mizoran/nagios/etc/nrpe.cfg --inetd
```

***** *XINETD* *****

Ako sistem koristi `xinetd` superserver, potrebno je u direktorijumu `/etc/xinetd.d` kreirati datoteku pod imenom `nrpe` i sa sledećim sadržajem:

```
# default: on
# description: NRPE
service nrpe
```

```
{
            flags = REUSE
            socket_type = stream
            wait = no
            user = <korisnik>
            server = <nrpe_bin>
            server_args = -c <nrpe_cfg> --inetd
            log_on_failure += USERID
            disable = no
            only_from = <ip_adresa_1> <ip_adresa_2> ...
      }
```

Pri tom se ponovo umesto <korisnik>, <nrp_bin> i <nrpe_cfg> treba navesti kosriničko ime pod kojim će se nrpe izvršavati, putanja do izvršne datoteke programa i putanja do konfiguracione dastoteke, respektivno. Ovde još treba dodati listu IP adresa sa kojih je dozvoljen pristup `nrpe` programu.

Nakon ovih podešavanja, treba restartovati superserver odgovarajućom komandom

```
/etc/rc.d/init.d/inet restart
```

ili

```
/etc/rc.d/init.d/xinetd restart
```

Takođe, toplo je preporučljivo u konfiguracionim datotekama `/etc/hosts.allow` i `/etc/hosts.deny` terba omogućiti TCP *wrapper* zaštitu nrpe servisa.

Način pokretanja programa

U skladu sa prethodno rečenim, program se može pokretati na jedan od sledećih načina. Dati primer prikazuje jedini način na koji sam mogao da pokrenem nrpe (sa dozovlama običnog korisnika na serveru `galeb`).

Format: `./nrpe –c <nrp_cfg> --daemon | --inetd | --xinetd`

Primer:
```
/users/etf/mizoran/nagios/bin/nrpe
–c /users/etf/mizoran/nagios/etc/nrpe.cfg
--daemon
```

Konfiguracija

Ovde je dat sadržaj korišćene konfiguracione datoteke `nrpe.cfg` sa objašnjenjima direktiva u obliku komentara.

```
#############################################################################
# NRPE Config File – nrpe.cfg
# Written by: Ethan Galstad (nagios@nagios.org)
#############################################################################
# PORT NUMBER
# Port number we should wait for connections on.
# NOTE: This must be a non-priviledged port (i.e. > 1024). This
# option is ignored if NRPE is running under either inetd or xinetd
server_port=5666
# SERVER ADDRESS
# Address that nrpe should bind to in case there are more than one
# interface and you do not want nrpe to bind on all interfaces.
```

```
# NOTE: This option is ignored if NRPE is running under either
# inetd or xinetd
server_address=147.91.8.64
# ALLOWED HOST ADDRESSES
# This is a comma-delimited list of IP address of hosts that are
# allowed to talk to the NRPE daemon.
# NOTE: The daemon only does rudimentary checking of the client's
# IP address. I would highly recommend adding entries in your
# /etc/hosts.allow file to allow only the specified host to connect
# to the port you are running this daemon on.
# NOTE: This option is ignored if NRPE is running under either
# inetd or xinetd
allowed_hosts=147.91.13.30,147.91.8.64
# NRPE USER
# This determines the effective user that the NRPE daemon should
# run as. You can either supply a username or a UID.
# NOTE: This option is ignored if NRPE is running under either
# inetd or xinetd
nrpe_user=mizoran
# NRPE GROUP
# This determines the effective group that the NRPE daemon should
# run as. You can either supply a group name or a GID.
# NOTE: This option is ignored if NRPE is running under either
# inetd or xinetd
nrpe_group=etf
# DEBUGGING OPTION
# This option determines whether or not debugging messages are
# logged to the syslog facility.
# Values: 0=debugging off, 1=debugging on
debug=0
# COMMAND TIMEOUT
# This specifies the maximum number of seconds that the NRPE daemon
# will allow plugins to finish executing before killing them off.
command_timeout=60
# COMMAND DEFINITIONS
# Command definitions that this daemon will run.  Definitions are
# in the following format:
# command[<command_name>]=<command_line>
# When the daemon receives a request to return the results of
# <command_name> it will execute the command specified by the
# <command_line> argument.
# Unlike Nagios, the command line cannot contain macros - it must
# be typed exactly as it should be executed.

#********** LOKALNI SERVISI NA SERVERU galeb ************
# trenutni broj logovanih korisnika
command[check_users]=/users/etf/mizoran/nagios/libexec/check_users
-w 15 -c 30
# provera CPU opterecenja
command[check_load]=/users/etf/mizoran/nagios/libexec/check_load -w
15,10,5 -c 30,25,20
### provera slobodnog prostora na particijama na galebu
command[check_disk_sda2]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sda2
```

```
command[check_disk_sda5]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sda5
command[check_disk_sda6]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sda6
command[check_disk_sda7]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sda7
command[check_disk_sdb2]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sdb2
command[check_disk_sdd2]=/users/etf/mizoran/nagios/libexec/check_di
sk -w 30 -c 10 -p /dev/sdd2
# provera trenutnog zombie broja procesa na galebu
command[check_zombie_procs]=/users/etf/mizoran/nagios/libexec/check
_procs -w 5 -c 10 -s Z
# procera ukupnog trenutnog broja procesa na galebu
command[check_total_procs]=/users/etf/mizoran/nagios/libexec/check_
procs -w 50 -c 100

# ****** KOMANDE PROVERA SERVISA KOJI SE NE VIDE SA gandalfa ******
# posredni check_PING el.etf.bg.ac.yu
command[check_ping_el]=/users/etf/mizoran/nagios/libexec/check_ping
-H 147.91.10.51 -w 100.0,25% -c 300.0,75% -p 1
# KRAJ DATOTEKE nrpe.cfg
```

Na strani nadgledajućeg hosta, primer definicije servisa koji se nadgleda posredstvom `nrpe` dodatka za Nagios izgleda na sledeći način:

```
define service{
      host_name               el
      service_description     PING
      check_command           check_nrpe!check_ping_el
      ... etc ...
      }
```

gde je *`check_ping_el`* krako ime komande koja je u potpunosti definisana u `nrpe.cfg` datoteci na udaljenom posredničkom serveru `galeb`.

Dodatak D – instalacija i konfiguracija *nsca* dodatka za Nagios

U ovom dodatku opisao sam postupak instalacije i konfiguracije `nsca` dodatka za Nagios. Klijentski deo programa *`send_nsca`* instalirao sam na serveru `galeb` dok je serverski deo aplikacije instaliran na nadgledajućem hostu `gandalf`. Time je eksternim entitetima sa servera `galeb` omogućeno podnošenje rezultata pasivnih provera servisa na nadgledajući host.

Kao i kod `nrpe` agentskog programa, mehanizamu `nsca` dodatka kojim se aplikaciji sa udanjene mašine dozvoljava da na lak i siguran način upisuje u datoteku eksternih komandi na nadgledajućem hostu, bio je moguć samo na serveru `galeb`.

Prevođenje

Izvorni kod programa je dostupan na zvaničnom sajtu Nagios projekta. Postupak prevođenja programa je jako jednostavan. Sastoji se od otpakivanja *tarball* datoteke, pokretanja konfiguracionog skripta i prevođenja programa. Komande su sledeće:

```
tar –xzvf nagios-nsca_2.4.tar.gz
cd nrpe-2.4
./configure
make all
```

Nakon izvršenog prevođenja, izvršne datoteke i klijentskog i serverskog dela aplikacije su smeštene u `/src` poddirektorijumu odakle ih treba ručno iskopirati u odgovarajuće direktorijume. Ciljne platforme na kojima se instalira `nsca` dodatak mogu biti različite te je generalno postupak prevođenja neophodno izvršiti i na udaljenoj mašini i na nadgledajućem hostu.

U mom slučaju, serverski deo aplikacije, `nsca`, je bio već preveden i instaliran u okviru *Gentoo*-ove automatske instalacije Nagiosa. Ručno sam instalirao samo `send_nsca` na serveru `galeb`.

Instalacija

Posle završenog prevođenja na serveru `galeb`, izvršnu datoteku programa *`send_nsca`* sam iskopirao u direktorijum `/users/etf/mizoran/nagios/bin`, dok sam njegovu jedinu konfiguracionu datoteku, `send_nsca.cfg`, smestio u direktorijum `/users/etf/mizoran/nagios/etc`.

Modovi rada *nsca* demona

Program `nsca` se može pokrenuti na tri načina: kao samostalni *single*-procesni (pozadinski) demonski program, kao samostalni multiprocesni demonski program ili kao servis pod inetd ili xinetd superserverom. Konfigurisanje je u potpunosti analogno prikazanom postupku konfigurisanja `nrpe` dodatka u Dodatku C.

Način pokretanja programa

U ovom primeru prikazan je način ručnog pokretanja `nsca` dodatka na nadgledajućem hostu. Program nije pokretan automatski jer pasivne provere praktično nisu ni vršene.

Format: `./nsca –c <nsca_cfg> --daemon | --inetd | --xinetd`

Primer: `/usr/nagios/bin/nsca –c /etc/nagios/nsca.cfg --daemon`

Konfiguracija

Ovde je dat sadržaj korišćene konfiguracione datoteke `nsca.cfg` na nadledajućem hostu sa objašnjenjima direktiva u obliku komentara.

```
##############################################################################
# NSCA Config File – nsca.cfg
# Written by: Ethan Galstad (nagios@nagios.org)
##############################################################################
# PORT NUMBER
# Port number we should wait for connections on.
# NOTE: This must be a non-priviledged port (i.e. > 1024). This
# option is ignored if NRPE is running under either inetd or xinetd
server_port=5667
# SERVER ADDRESS
# Address that nsca should bind to in case there are more than one
# interface and you do not want nrpe to bind on all interfaces.
server_address=147.91.13.30
# ALLOWED HOST ADDRESSES
# This is a comma-delimited list of IP address of hosts that are
# allowed to talk to the NSCA daemon.
# NOTE: The daemon only does rudimentary checking of the client's
# IP address. I would highly recommend adding entries in your
# /etc/hosts.allow file to allow only the specified host to connect
# to the port you are running this daemon on.
allowed_hosts=147.91.13.30,147.91.8.64
# NSCA USER
# This determines the effective user that the NSCA daemon should
# run as. You can either supply a username or a UID.
# NOTE: This option is ignored if NSCA is running under either
# inetd or xinetd
nrpe_user=nagios
# NSCA GROUP
# This determines the effective group that the NSCA daemon should
# run as. You can either supply a group name or a GID.
# NOTE: This option is ignored if NRPE is running under either
# inetd or xinetd
nrpe_group=nagios
# DEBUGGING OPTION
# This option determines whether or not debugging messages are
# logged to the syslog facility.
# Values: 0=debugging off, 1=debugging on
debug=0
# COMMAND FILE
```

```
# This is the location of the Nagios command file that the daemon
# should write all service check results that it receives.
command_file=/var/nagios/rw/nagios.cmd
# ALTERNATE DUMP FILE
# This is used to specify an alternate file the daemon should write
# service check results to in the event the command file does not
# exist. It is important to note that the command file is
# implemented as a named pipe and only exists when Nagios is
# running. You may want to modify the startup script for Nagios to
# dump the contents of this file into the command file after it
# starts Nagios. Or you may simply choose to ignore any check
# results received while Nagios was not running...
alternate_dump_file=/var/nagios/rw/nsca.dump
# AGGREGATED WRITES OPTION
# This option determines whether or not the nsca daemon will
# aggregate writes to the external command file for client
# connections that contain multiple check results.  If you
# are queueing service check results on remote hosts and
# sending them to the nsca daemon in bulk, you will probably
# want to enable bulk writes, as this will be a bit more
# efficient.
# Values: 0 = do not aggregate writes, 1 = aggregate writes
aggregate_writes=0
# APPEND TO FILE OPTION
# This option determines whether or not the nsca daemon will
# will open the external command file for writing or appending.
# This option should almost *always* be set to 0!
# Values: 0 = open file for writing, 1 = open file for appending
append_to_file=0
# MAX PACKET AGE OPTION
# This option is used by the nsca daemon to determine when client
# data is too old to be valid.  Keeping this value as small as
# possible is recommended, as it helps prevent the possibility of
# "replay" attacks.  This value needs to be at least as long as
# the time it takes your clients to send their data to the server.
# Values are in seconds.  The max packet age cannot exceed 15
# minutes (900 seconds).
max_packet_age=30
# DECRYPTION PASSWORD
# This is the password/passphrase that should be used to descrypt
# the incoming packets. Note that all clients must encrypt the
# packets they send using the same password!
# IMPORTANT: You don't want all the users on this system to be able
# to read the password you specify here, so make sure to set
# restrictive permissions on this config file!
password=n4gi0s
# DECRYPTION METHOD
# This option determines the method by which the nsca daemon will
# decrypt the packets it receives from the clients.  The decryption
# method you choose will be a balance between security and
# performance, as strong encryption methods consume more processor
# resources. You should evaluate your security needs when choosing
# a decryption method.
# Note: The decryption method you specify here must match the
#       encryption method the nsca clients use (as specified in
#       the send_nsca.cfg file)!!
# Values:
# 0 = None  (Do NOT use this option)
# 1 = Simple XOR  (No security, just obfuscation, but very fast)
# 2 = DES
# 3 = 3DES (Triple DES)
```

```
# 4 = CAST-128
# 5 = CAST-256
# 6 = xTEA
# 7 = 3WAY
# 8 = BLOWFISH
# 9 = TWOFISH
# 10 = LOKI97
# 11 = RC2
# 12 = ARCFOUR
# 14 = RIJNDAEL-128
# 15 = RIJNDAEL-192
# 16 = RIJNDAEL-256
# 19 = WAKE
# 20 = SERPENT
# 22 = ENIGMA (Unix crypt)
# 23 = GOST
# 24 = SAFER64
# 25 = SAFER128
# 26 = SAFER+
#
decryption_method=1
# KRAJ DATOTEKE nsca.cfg
```

U nastavku sledi jako jednostavna konfiguraciona datoteka programa `send_nsca`, `send_nsca.cfg`.

```
#####################################################
# NSCA Client Config File
# Written by: Ethan Galstad (nagios@nagios.org)
# Last Modified: 02-21-2002
#####################################################
# ENCRYPTION PASSWORD
# This is the password/passphrase that should be used to encrypt
# the outgoing packets. Note that the nsca daemon must use the same
# password when decrypting the packet!
# IMPORTANT: You don't want all the users on this system to be able
# to read the password you specify here, so make sure to set
# restrictive permissions on this config file!
password=n4gi0s
# ENCRYPTION METHOD
# This option determines the method by which the send_nsca client
# will encrypt the packets it sends to the nsca daemon. The
# encryption method you choose will be a balance between security
# and performance, as strong encryption methods consume more
# processor resources.
# Note: The encryption method you specify here must match the
#       decryption method the nsca daemon uses (as specified in
#       the nsca.cfg file)!!
# Values:
# 0 = None  (Do NOT use this option)
# 1 = Simple XOR  (No security, just obfuscation, but very fast)
# 2 = DES
# 3 = 3DES (Triple DES)
# 4 = CAST-128
# ...
encryption_method=1
# KRAJ DATOTEKE send_nsca.cfg
```

Dodatak E –implementirane komande Nagiosa

Ovo je pregled svih eksternih komandi koje su implementitrane u Nagios. Svi vremenski argumenti moraju se zadavati u `time_t` formatu (broj sekundi od početka UNIX epohe).

ime komande	argumenti komande	opis
ADD_HOST_COMMENT	<host_name>; <persistent>; <author>; <comment>	Ova komanda se koristi za dodavanje komentara određenom hostu. Argument author treba da sadrži ime korisnika koji je uneo komentar. Aktuelni komentar ne bi trebalo da sadrži ni jednu semi kolonu. Persistent fleg određuje da li komentar preživljava restart programa (1=da, 0=ne)
ADD_SVC_COMMENT	<host_name>; <service_description>; <persistent>;<author>; <comment>	Koristi se za pridruživanje komentara određenom servisu. Primetiti da su potrebni i ime hosta i opis servisa.
DEL_HOST_COMMENT	<comment_id>	Briše komentar sa imenom komentara comment_id za određeni host.
DEL_ALL_HOST_COMMENTS	<host_name>	Briše sve komentare za određeni host.
DEL_SVC_COMMENT	<comment_id>	Briše komentar sa imenom komentara comment_id za određeni servis.
DEL_ALL_SVC_COMMENTS	<host_name>; <service_description>	Briše sve komentare za određeni servis.
DELAY_HOST_NOTIFICATION	<host_name>; <next_notification_time>	Odlaže sledeće obaveštenje o ovom hostu do zadatog trenutka. Nema efekta ako se stanje hosta promeni pre trenutka sledećeg obaveštavanja.
DELAY_SVC_NOTIFICATION	<host_name>; <service_description>; <next_notification_time>	Odlaže sledeće obaveštenje o ovom hostu do zadatog trenutka. Nema efekta ako se stanje hosta promeni pre trenutka sledećeg obaveštavanja. Ovo ne odlaže obaveštenje o hostu.
SCHEDULE_SVC_CHECK	<host_name>; <service_description>; <next_check_time>	Preraspoređuje sledeću proveru određenog servisa do zadatog trenutka.
SCHEDULE_HOST_SVC_CHECKS	<host_name>; <next_check_time>	Preraspoređuje sledeću proveru svih servisa određenog hosta do zadatog trenutka
DISABLE_SVC_CHECK	<host_name>; <service_description>	Privremeno onemogućava provere zadatog servisa. Provevre servisa se automatski uključuju pri reinicijalizaciji Nagiosa. Bočni efekat ove komande je privremeno

		zaustavljanje obaveštavanja o ovom servisu. Ovo ne onemogućava obaveštavanja o hostu.
ENABLE_SVC_NOTIFICATIONS	<host_name>; <service_description>	Ponovno se omogućuje obaveštavanje o zadatom servisu.
DISABLE_SVC_NOTIFICATIONS	<host_name>; <service_description>	Privremeno onemogućava obaveštavanje o zadatom servisu. Efekat se gubi pri restartu Nagiosa. Ovo ne isključuje obaveštenja o hostu.
ENABLE_HOST_SVC_NOTIFICATIONS	<host_name>	Ponovo omogućava obaveštavanje za sve servise zadatog hosta. Ne omogućava obaveštenja o samom hostu.
DISABLE_HOST_SVC_NOTIFICATIONS	<host_name>	Privremeno onemogućava obaveštenja o svim servisima zadatog hosta. Ne onemogućava obaveštenja o hostu.
ENABLE_HOST_SVC_CHECKS	<host_name>	Ponovno omogućava proveru svih servisa na zadatom hostu. Ako je jedan ili više servisa u non-OK stanju prilikom ovog onemogućavanja, kontakti bi mogli da prime obaveštenje o oporavku servisa posle ponovnog omogućavanja provera.
DISABLE_HOST_SVC_CHECKS	<host_name>	Privremeno onemogućava provere svih servisa zadatog hosta. Efekat se gubi pri restartu Nagiosa. Bočni efekat je privremeno onemogućavanje obaveštenja o zahvaćenim servisima. Ne utiče na obaveštenje o hostu.
ENABLE_HOST_NOTIFICATIONS	<host_name>	Privremeno omogućuje obaveštenja o ovom hostu. Ne utiče na obaveštenja o servisima pridruženim ovom hostu.
DISABLE_HOST_NOTIFICATIONS	<host_name>	Privremeno onemogućava obaveštenja o ovom hostu. Gubi efekat pri restartu. Ne utiče na obaveštenja o servisima pridruženim ovom hostu.
ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST	<host_name>	Omogućava obaveštenja o svim hostovima i servisima "iza" zadatog hosta (iz perspektive Nagiosa). Često se koristi kod redundantnog nadgledanja hostova.
DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST	<host_name>	Suprotno od prethodnog.
ENABLE_NOTIFICATIONS	<execution_time>	Omogućava obaveštenja o hostovima i servisima na nivou čitavog programa u zadatom trenutku.
DISABLE_NOTIFICATIONS	<execution_time>	Suprotno od prethodnog.

SHUTDOWN_PROGRAM	<execution_time>	Zaustavlja Nagios u zadatom trenutku. Napomena: Nagios se ne može restarovati kroz web interfejs ako se na ovaj način izda komanda zaustavljanja.
RESTART_PROGRAM	<execution_time>	Primorava Nagios da flushuje sve konfiguracione podatke, ponovno pročita sve konfig. fajlove i restartuje nadgledanje u zadatom trenutku.
PROCESS_SERVICE_CHECK_RESULT	<host_name>; <service_description>; <return_code>; <plugin_output>	Predaje rezultate provere određenog servisa Nagiosu. Ove "pasivne" provere se ponašaju na isti način kao i normalne "aktivne" provere.
SAVE_STATE_INFORMATION	<execution_time>	Primorava Nagios da dampuje informacije o tekućem statusu svih servisa i hostova u datoteci definisanom state_retention_file promenljivom.
READ_STATE_INFORMATION	<execution_time>	Primorava Nagios da pročita prethodno snimljene informacije o stanju svih servisa i hostova iz state_retention datoteke.
START_EXECUTING_SVC_CHECKS		Nastavlja izvršavanje provera servisa. Izvršavanje provera može biti zaustavljeno u nekom ranijem trenutku zadavanjem STOP_EXECUTING_SVC_CHECKS komandom ili postavljanjem execute_service_checks opcije na 0 u glavnom konfig datoteci. Uglavnom se koristi kod redundantnog nadgledanja hostova.
STOP_EXECUTING_SVC_CHECKS		Stopira izvrsavnje provere servisa. Provere servisa koje se ne izvršavaju u tom trenutku neće se ukljuciti u red za čekanje. Nagios se praktično stavlja u "spavajući" mod dok god se vrši monitoring. Uglavnom se koristi kod redundantnog nadgledanja hostova.
START_ACCEPTING_PASSIVE_SVC_CHECKS		Nastavlja prijem pasivnih provera servisa za sve servise.
STOP_ACCEPTING_PASSIVE_SVC_CHECKS		Zaustavlja prijem pasivnih provera servisa za sve servise.
ENABLE_PASSIVE_SVC_CHECKS	<host_name>; <service_description>	Nastavlja prijem pasivnih provera servisa za zadati servis.
DISABLE_PASSIVE_SVC_CHECKS	<host_name>; <service_description>	Zaustavlja prijem pasivnih provera servisa za zadati servis.

Reference

[1] http://www.nagios.org
[2] http://nagiosplug.sourceforge.net/developer-guidelines.html
[3] http://mcrypt.hellug.gr
[4] http://net-snmp.sourceforge.net
[5] http://cgiwrap.unixtools.org/