SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1104

DODAVANJE PODRŠKE ZA IZVRŠNE DATOTEKE LINUXA U CUCKOO OKRUŽENJU ZA DINAMIČKU ANALIZU

Nikola Ravnjak

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 9. ožujka 2015.

Predmet: **Diplomski rad**

DIPLOMSKI ZADATAK br. 1104

Pristupnik: **Nikola Ravnjak (0036462135)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Dodavanje podrške za Linux izvršne datoteke u Cuckoo okruženju za dinamičku analizu**

Opis zadatka:

Cuckoo okruženje je alat za dinamičku analizu nepoznatih binarnih datoteka koje potencijalno sadrže zlođudni kod. Potencijalno zlođudna datoteka se izvršava unutar strogo kontroliranog okruženja pri čemu se istovremeno prati njeno ponašanje. Na temelju tog praćenja generiraju se izvještaji i određuje se je li datoteka zlođudna ili ne. Cuckoo je namijenjen za analizu isključivo Windows izvršnih datoteka i koristi se, primjerice, u sustavu VirusTotal koji održava Google. Međutim, sve je veća potreba za automatiziranom analizom zlođudnog koda s operacijskog sustava Linux.
U sklopu diplomskog rada potrebno je proučiti Cuckoo okruženje te način na koji bi se moglo proširiti za analizu zlođudnog koda s operacijskog sustava Linux. Na temelju tako obavljene analize dodati podršku za Linux binarni kod te testirati sustav. Radu priložiti izvorni kod razvijenih i korištenih programa. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:
Doc. dr. sc. Stjepan Groš

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Siniša Srbljić

Zahvaljujem se mentoru, doc. dr. sc. Stjepanu Grošu, na iskazanom razumijevanju, savjetima i pomoći prilikom izrade diplomskog rada.

Sadržaj

1. Uvod

Zloćudni programi predstavljaju ozbiljne prijetnje za sve korisnike informacijskih tehnologija. Postojeće metode zaštite često ne pružaju dovoljnu zaštitu korisnicima, a broj štetnih i ilegalnih programa je u velikom porastu.

Tijekom istraživanja zloćudnog programa, stručnjaci obično posežu za pitanjima kao što su: Koje radnje može maliciozan program izvršiti na sustavu? Kako se širi? Je li održava kontakt s napadačem? Sva ova pitanja se mogu odgovoriti pomoću dinamičke analize malicioznog programa u kontroliranom okruženju. Jedan takav automatizirani sustav je Cuckoo Sandbox [1]. Cuckoo Sandbox pomaže u shvaćanju načina rada zloćudnih programa tako što za samo nekoliko sekundi daje detaljan uvid u ponašanje zloćudnog programa. Ovaj alat trenutno podržava analizu za gotovo sve tipove datoteka koje se mogu izvršavati na Windowsima. Međutim Cuckoo ne podržava analizu Linux izvršnih datoteka. U okviru ovog rada je dodana podrška za analizu Linux izvršnih datoteka na Cuckoo Sandbox.

Ovaj rad je podijeljen u šest poglavlja. U drugom poglavlju su opisani osnovni pojmovi vezani za statičku i dinamičku analize. Potom su detaljnije opisane mogućnosti dinamičke analize na Linux operacijskim sustavima. U trećem poglavlju su prikazane osnovne informacije i način rada Cuckoo Sandboxa. Četvrto poglavlje prikazuje usporedbu i odabir tehnologija i alata korištenih pri izradi rada.. U petom poglavlju je opisan sustav za dinamičku analizu zloćudnih programa na Linuxu koji je dodan u Cuckoo Sandbox u sklopu ovoga rada. Šesto poglavlje prikazuje rezultate rada programa. Rad završava zaključkom, sažetkom i privitkom koji uključuje upute za instaliranje cjelokupnog okruženja za dinamičku analizu zloćudnih programa i upute za pokretanje Cuckoo Sandboxa.

2. Analiza zloćudnih programa

Svaki program koji uzrokuje štetu korisniku, računalu ili mreži nazivamo zloćudni program (engl. *malicious software*, skraćeno *malware*) [1]. Možemo ih podijeliti u različite kategorije kao što su virusi, trojanski konji ili crvi.

Analiza zloćudnih programa je vještina seciranja zloćudnih programa da bi se otkrilo kako taj program identificirati, kako on radi, i kako ga ukloniti. Analiza zloćudnih programa se dijeli na statičku analizu izvornog kôda i dinamičku analizu. Često se, radi što boljeg uvida u ponašanje zloćudnog programa koristi kombinacija obje vrste analize, pri čemu rezultati statičke analize izvornog kôda pomažu prilikom dinamičke analize.

2.1. Statička analiza

Pod statičkom analizom kôda se podrazumijeva analiziranje izvornog kôda aplikacije prije njenog izvršavanja. Zbog toga se statička analiza naziva i analiza kôda i odnosi se na analizu bez pokretanja programa. Jedna od dobrih strana statičke analize leži u činjenici da su unaprijed poznate instrukcije koje program može izvršiti, te ne postoji potreba za nagađanjem ili interpretacijom ponašanja pojedinih instrukcija.

Izvorni kôd zloćudnog uzorka nije čitljiv. Zbog toga se kao glavni nedostatak statičke analize može istaći potreba za pristupom izvornom kôdu same aplikacije. Početni korak statičke analize je pretvorba programskog kôda u odgovarajuću internu reprezentaciju koja je pogodna za daljnju analizu. Stoga je potrebno obaviti disasembliranje i dekompajliranje. *Disassembler* generira asemblerski kôd koji se može čitati i analizirati kako bi se utvrdilo što i kako program radi. Asemblerski kôd se može analizirati naprednim postupkom reverznog inženjerstva (engl.*reverse engineering*). Postupak pretvorbe se provodi u nekoliko koraka od kojih su najvažniji leksička i sintaksna analiza. Leksička analiza je postupak kojim se programski kôd transformira u niz leksičkih oznaka (engl. *token*), odbacujući pritom nevažne podatke poput komentara i praznina. Mnogi alati provode samo

leksičku analizu nakon čega traže određene uzorke pomoću kojih se određuje provođenje daljnjih akcija [2]. Sintaksna analiza je postupak kojim se temeljem leksičkih oznaka te skupa produkcija gramatike određenog jezika gradi stablo parsiranja, odnosno apstraktno sintaksno stablo. Takvo stablo odražava strukturu programskog kôda. Izrada apstraktnog sintaksnog stabla omogućava praćenje kontrolnog toka programa.

Napredna statička analiza sastoji se od reverznog inženjerstva učitavanjem izvršne datoteke u *disassembler* i pregledavanje naredbi programa koje izvršava procesor (engl. *Central Processing Unit*, skraćeno CPU) kako bi se utvrdilo što program radi. Izvršne datoteke započinju sa zaglavljem (engl. *header*) koje sadrži informacije o kôdu, tipu aplikacije, veličini datoteke i potrebnim bibliotekama. Kada se kompilira izvorni kôd u binarni prikaz, neke informacije zaglavlja se izgube. Te informacije zaglavlja su od velikog značaja zbog čega gubitak dovodi do otežavanja daljnje analize kôda.

Jedna od najkorisnijih informacija koje možemo prikupiti od izvršnog programa je lista funkcija koje program uvozi. Uvezene funkcije su funkcije koje koristi glavni program a koje su pohranjene u nekom drugom programu kao što su biblioteke koje sadrže osnovne funkcionalnosti za različite programe. Biblioteke su povezane s glavnom izvršnom datotekom povezivanjem (engl. *linkingom).* Biblioteke mogu biti povezane statički, dinamički ili tijekom izvođenja. Statičko povezivanje je najrjeđe korištena metoda povezivanja. Obično se koristi na UNIX operacijskim sustavima. Ako je biblioteka statički povezana s izvršnim programom, cijeli kôd biblioteke se kopira u izvršnu datoteku što dovodi do porasta veličine izvršnog programa. Ovu vrstu povezivanja je teško detektirati jer ništa u zaglavlju ne ukazuje da datoteka sadrži povezani kôd. Izvršne datoteke koje koriste povezivanje tijekom izvođenja, povezuju biblioteke samo onda kada se poziva neka funkcija iz biblioteke. Dinamičko povezivanje je najčešći oblik povezivanja. Operacijski sustav pronalazi potrebne biblioteke kada se program učita. Kada program poziva funkcije iz povezane biblioteke, funkcije se izvode unutar biblioteke. Zaglavlje sadrži informacije o svakoj biblioteci koja će se učitati i svakoj funkciji koja se koristi. Zbog toga je vrlo važno identificirati biblioteke i funkcije jer se na taj način lako može pretpostaviti što program radi.

Prednost statičke analize jest ta da može otkriti ponašanje programa pod neuobičajenim uvjetima zato što se mogu ispitati dijelovi programa koji se normalno neće izvesti. Osnovnom statičkom analizom je moguće utvrditi koje su točno funkcije uvezene ali bez napredne analize se ne može znati kako je neka funkcija upotrebljena i je li uopće pozvana. Mnoge analize zloćudnog kôda najprije provode statičku analizu zato što je sigurnija od dinamičke analize [3].

2.2. Dinamička analiza

Dinamička analiza ili ponašajna analiza je proces izvršavanja zloćudnog programa u promatranom okruženju kako bi se utvrdilo ponašanje tog programa. Za razliku od statičke analize izvornog kôda, dinamička analiza se izvršava za vrijeme izvođenja programa. Obično se izvodi nakon što završi osnovna statička analiza. Dinamička analiza se može sastojati od praćenja zloćudnog programa tijekom izvođenja ili ispitivanja sustava nakon što je zloćudni program izvršen. Za razliku od statičke analize, dinamička analiza omogućuje promatranje točne funkcionalnosti programa jer primjerice, postojanje naziva neke funkcije u binarnoj datoteci ne znači da će se funkcija zapravo izvršiti. Prednost ove analize je mogućnost vrlo brzog i preciznog prikaza podataka kao što su informacije o kreiranim i obrisanim datotekama ili ključevima registra, posjećenim mrežnim stranicama, kreiranim procesima i sl. Cilj dinamičke analize je pokretanje nepoznate i sumnjive aplikacije ili datoteke unutar izoliranog sustava i dobivanje informacija o tome što promatrani program radi [4].

Dinamička analiza ima ograničenja jer nisu možda sve putanje u kôdu dostupne za izvršavanje kada se program izvodi. Također je moguće da maliciozni program prima različite argumente za koje se drugačije ponaša. Zato bez poznavanja svih mogućnosti, dinamička analiza neće otkriti sve funkcionalnosti programa. To je jedan od razloga zašto dinamičkoj analizi prethodi osnovna statička analiza u kojoj je moguće saznati informacije o funkcijama koje program poziva tijekom izvođenja. Glavni nedostatak dinamičke analize jest da je jedini način da utvrdimo svrhu programa, pokretanje programa i promatranje daljnjih događanja. To je zato što alati za dinamičku analizu mogu otkriti izvor problema

tek nakon što je do njega došlo. Nedostatak takvog postupka leži u činjenici da je potrebno moći reproducirati problem da bi se mogao naći njegov uzrok, što nije uvijek jednostavan posao. Pokretanjem malicioznog programa može doći do različitih poteškoća kao što su brisanje podataka ili slanje elektroničke pošte. Zbog toga je nepoznate programe sigurnije izvoditi u okruženju u kojem se ne može napraviti šteta. Najpopularnije takvo okruženje za dinamičku analizu i testiranje programa je *sandbox*. *Sandbox* je sigurnosni mehanizam za pokretanje sumnjivih programa u sigurnom okruženju bez straha od štete na "stvarnom" sustavu [1]. Postoje različiti *sandbox* mehanizmi kao što su zatvor (engl. *jail*), *Applet* programi, virtualna računala (engl. *Virtual Machine*, skraćeno VM) i *sandbox* na stvarnim računalima. Najpopularniji i najviše korišteni *sandbox* mehanizam je virtualno računalo ili virtualni stroj. Virtualna stroj se koristi često kao *sandbox* mehanizam u dinamičkoj analizi zloćudnih programa. Nakon što se zloćudni program pokrene u virtualnom stroju, obično se pokreće i monitor koji nadzire cijeli sustav i rad zloćudnog programa.

Sandboxom se strogo nadgledaju i ograničavaju resursi na računalu, poput mrežnog pristupa ili prostora na tvrdom disku namijenjenom za privremenu pohranu podataka ili prostora u memoriji, za pokretanje programa ili kôdova koji su potekli iz nepoznatih izvora. Vrlo je važna i mogućnost podešavanje razina pristupa nekog programa na računalu. Određenom procesu su dodijeljeni ograničeni resursi i pravila kojima se određuju njegove mogućnosti djelovanja na računalu. *Sandbox* odvaja korisničke zahtjeve i procese od jezgre operacijskog sustava. Najveća prednost *sandboxa* je da zbog virtualizacije i odvajanja datoteka operacijskog sustava i rada korisnika, mogućnost da računalo bude zaraženo zlonamjernim programom je svedena na minimum. Jedan od najvećih nedostataka dinamičke analize pomoću *sandbox* virtualnog stroja jest prepoznavanje virtualnog stroja od strane malicioznog programa. U tom slučaju zloćudni program se prestane izvoditi ili se počne ponašati drugačije. Maliciozni programi koriste razne tehnike kako bi detektirali izvođenje na virtualnom stroju. Najčešće se koriste informacije o komunikaciji s nadzirateljem kao što je IP adresa. Virtualizacijska okruženja imaju unaprijed zadane IP adrese tako da zloćudni programi često tako detektiraju izvođenje na *sandboxu*. Još jedan nedostatak *sandboxa* jest nepodržavanje operacijskih sustava od strane zloćudnog programa tako da se program neće moći uvijek izvršiti.Sustav za praćenje

pokrenut unutar *sandboxa* nije uvijek u mogućnosti zabilježiti sva događanja nakon pokretanja programa zato što analiza traje određeno vrijeme. Moguće je da maliciozni program pomoću funkcije *sleep* izvodi zlonamjerne aktivnosti tek nakon nekoliko sati ili dana. Monitor neće prikazati što točno zloćudni program radi nego će izvijestiti o osnovnim funkcionalnostima i promjenama na temelju kojih treba zaključiti što promatrani zloćudni uzorak točno radi, kojeg je tipa i sl.

Potrebno je spomenuti i *debugger* kao alat koji se često koristi u naprednoj dinamičkoj analizi zloćudnog kôda. *Debugger* je programska podrška ili sklopovlje koji služi za testiranje ili pregledavanje izvođenja drugog programa [1]. *Debuggeri* pružaju uvid u sve što program radi dok se izvršava. Dizajnirani su tako da dopuste programerima mjerenje i kontroliranje izvođenja programa. Za razliku od *disassemblera* koji pruža sliku programa prije njegovog izvršavanja, *debugger* pruža informacije o programu dok se izvršava. Primjerice, *debuggeri* mogu pokazati promjene vrijednosti memorijskih adresa tijekom izvođenja programa, vrijednosti svih registara ili argumente svake funkcije koja se poziva.

2.3. Dinamička analiza zloćudnog programa na Linuxu

Za razliku od Windows operacijskih sustava koji koristi Windows API (engl. *Application Programming Interface*) za povezivanje korisničkih programa i Windowsa, Linux sustavi koriste sistemske pozive (engl. *system call*, skraćeno *syscall*) za komunikaciju između programa i jezgre (engl. *kernel*). Stoga je za praćenje rada zloćudnog programa prilikom izvođenja potrebno nadzirati sistemske pozive.

2.3.1. Sistemski pozivi

U Linux operacijskom sustavu, instanca programa koji se izvršava se naziva proces. Svaki put kada se pokrene program, kreira se proces za taj program i jedinstveni identifikacijski broj koji se naziva *PID*. Pokrenutom procesu se dodijeli memorija u kojoj su smješteni izvršni kôd, programski podaci, stog (engl. *stack*), gomila (engl. *heap*) i opisnici datoteka (engl. *file descriptor*, skraćeno FD).

Sistemski pozivi su primaran način komunikacije programa i operacijskog sustava. Sučelje sistemskih poziva uključuje velik broj funkcija koje operacijski sustav nudi aplikacijama koje se na njemu izvršavaju. Te funkcije omogućavaju aplikacijama otvaranje datoteka, uspostavljanje mrežnog priključka, čitanje datoteka i sl. Kao posljedica toga, promatranje sistemskih poziva može donijeti odličan uvid u ono što program radi, a može biti i od neprocjenjive vrijednosti za rješavanje problema, praćenje ili identificiranje uskog grla. Sistemski poziv je sučelje između korisničke aplikacije i servisa na strani jezgre operacijskog sustava. Nije moguće direktno pristupati jezgri operacijskog sustava. Sistemski poziv izgleda kao funkcijski poziv za programe koji ga koriste ali je u praksi kompliciraniji od funkcijskog poziva jer zahtjeva transakciju između korisničkog načina (engl. *user mode*) i načina jezgre (engl. k*ernel mode*). Postoji oko tristo različitih sistemskih poziva koji se mogu grupirati u pet kategorija [5], [6]:

1. kontrola procesa
2. upravljanje datotekama
3. upravljanje uređajima
4. održavanje informacija
5. održavanje komunikacija

Najosjetljiviji sistemski pozivi su oni zaduženi za datotečni sustav i mrežne operacije [7], [8]. Najvažniji sistemski pozivi koji rukuju s datotečnim sustavom su *access*, *open*, *read* i *write* dok su *listen*, *socket*, *bind* i *connect* zaduženi za mrežne operacije. Sistemski pozivi *clone* i *execve* su također važni jer su to jedini pozivi zaduženi za stvaranje novih procesa i dretvi i za pokretanje programa.

2.3.2. Praćenje sistemskih poziva

Monitor i cenzor su dva najčešće korištena načina nadzora sistemskih poziva [9].

Monitor je sustav koji prati sistemske pozive i prikazuje informacije kao što su naziv sistemskog poziva, argumenti i povratna vrijednost. Praćenjem sistemskih poziva promatraju se informacije koje su vrlo bliske jezgri. Promatrajući sistemske pozive monitorom potpuno se zanemaruju događanja unutar procesa. Cijeli proces se tretira kao

crna kutija. Takav pristup ima smisla u okruženjima gdje svaki pristup datoteci ili svaki pristup mreži zahtjeva sistemski poziv zbog pristupanja jezgri operacijskog sustava. U mnogim programima, sistemski pozivi se događaju relativno rijetko pa je praćenje poziva puno korisnije od gledanja pojedinačnih instrukcija računala. Nadgledanje ponašanja malicioznog uzorka praćenjem sistemskih poziva je prikladno zato što takav pristup pokriva sve interakcije između procesa i okruženja na kojem se izvodi i zato što se rukuje s informacijama koje su korisne. Najpoznatiji alati za praćenje sistemskih poziva na Linux operacijskom sustavu su *strace* [10], *dtrace* [11], *SystemTap* [12] i *sysdig* [13]. Informacije koje se mogu dobiti praćenjem sistemskih poziva su obično vrlo prikladne za filtriranje po nazivu poziva, argumentu ili povratnoj vrijednosti.

Za razliku od pasivnih monitora sistemskih poziva, cenzori sistemskih poziva ne dozvoljavaju nanošenje štete okruženju u kojem se maliciozni uzorak izvršava. Najpopularniji primjeri cenzora su *Janus* cenzor i *Systrace* [14]. Janus cenzor funkcionira na način da propušta izvođenje prihvatljivih sistemskih poziva do kraja dok neprihvatljive sistemske pozive prekida. Systrace presreće sistemske pozive i komunicira s procesom na razini korisnika (engl. *user-level*) koji odlučuje o politici (engl. *policy*). Politike su izražene kao pravila koja sadrže naziv sistemskog poziva, argumente i radnju. Obično se najprije generiraju pravila a zatim pokreće naredba koja učitava pravila i odbacuje sve sistemske pozive koji se ne slažu s pravilima. Budući da je cenzuriranje sistemskih poziva implementirano od strane procesa koji je na razini korisnika, taj proces mora čuvati informacije o procesu kojeg prati. To se odnosi na podatke kao što su putanja trenutnog direktorija, otvorene datoteke, otvoreni mrežni *socket* i sl. Takve se informacije održavaju od strane jezgre pa pokušaj praćenja promjena od strane procesa na razini korisnika može dovesti do raznih problema. Problemi se većinom odnose na pokretanje višedretvenih procesa. Višedretveni procesi su procesi u kojima nekoliko dretvi izvođenja dijele isti adresni prostor.. Kada dretva napravi sistemski poziv samo je ta dretva blokirana. Bilo kada nakon što je cenzor razmotrio argumente, druga dretva istog procesa može i dalje mijenjati vrijednosti argumenata kao što je primjerice putanja trenutnog direktorija. Ovakve i slične situacije, u kojem dva ili više procesa čitaju ili pišu po zajedničkoj memoriji ili koriste neke zajedničke podatke, a konačni rezultat ovisi o tome u kojem se

trenutku koji proces izvršava, nazivaju se uvjeti utrke (engl. *race conditions*). Sistemski cenzori su ranjivi na *race conditions* bez obzira pokrenemo li ih u procesu na razini korisnika ili u jezgri operacijskog sustava. Zbog ovih problema cenzori sistemskih poziva se u dinamičkoj analizi obično koriste samo u kombinaciji s monitorom sistemskih poziva.

Postoji i alati koji mogu pomoći u dinamičkoj analizi a zaduženi su za praćenje procesa. Praćenje procesa je tehnika dinamičke analize pomoću koje je moguće pratiti promjene na datotečnom sustavu, mreži, procesima i dretvama. Iako se bilježi mnogo podataka, takvim praćenjem nije moguće uhvatiti sve sistemske pozive i mrežne aktivnosti. Prvenstveno jer se ponekad radi o više desetaka tisuća događaja u minuti koje treba procesirati. Najpoznatiji alati za praćenje procesa na Linux operacijskom sustavu su *htop* [15] i *lsof* [16].

3. Cuckoo Sandbox

Cuckoo Sandbox je alat koji omogućuje izvođenje dinamičke analize zloćudnog programa. Koristi se za automatizirano pokretanje i analizu datoteka i prikupljanje opsežnih rezultata analize koji opisuju ono što zloćudni program radi dok se izvodi u izoliranom Windows operacijskom sustavu. U nastavku su opisane osnovne informacije, arhitektura i način rada Cuckoo Sandbox-a.

3.1. Osnovno o Cuckoo Sandboxu

Cuckoo je program otvorenog kôda (engl. *open source*) koji predstavlja automatizirani sustav za analizu zloćudnih programa [3]. Cucoo sandbox je nastao kao *Google Summer of Code* projekt 2010. godine unutar organizacije *The Honeynet Project*. Prva inačica objavljena je 05.02.2011. godine. Trenutna stabilna inačica je *Cuckoo Sandbox* 1.2. Sve komponente su implementirane u programskom jeziku Python 2.7 osim monitora koji je većinom pisan u programskom jeziku C.

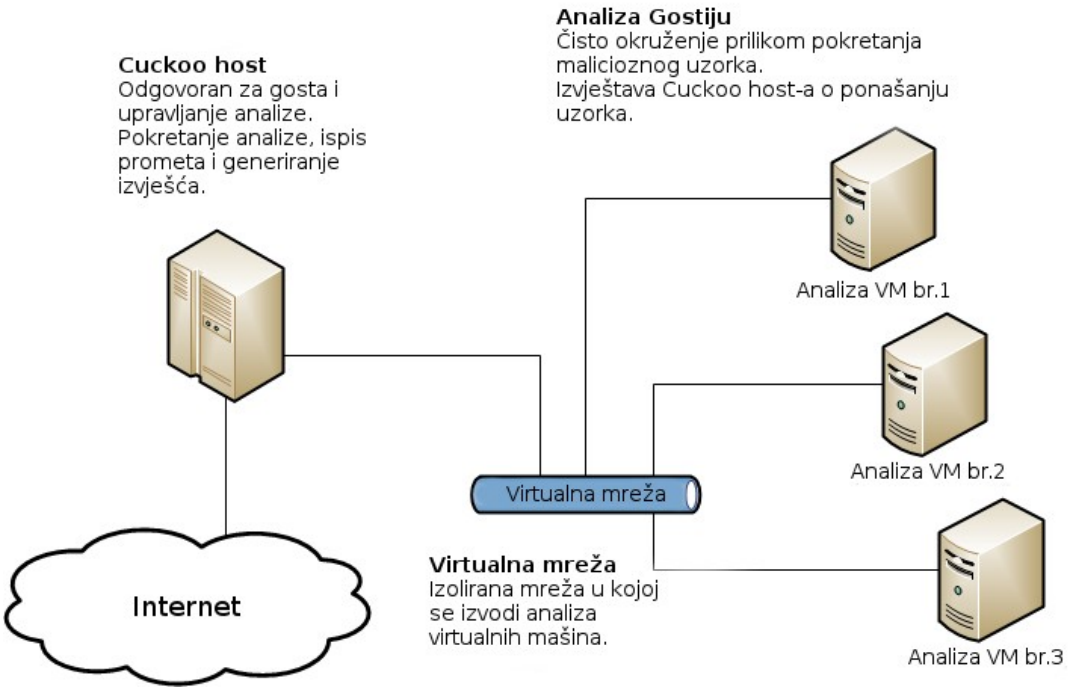
Cuckoo je dizajniran kao samostalna aplikacija ali se lako može integrirati u velike sustave, zahvaljujući izuzetno modularnom dizajnu. *Cuckoo* se koristi za analizu Windows izvršnih datoteka, DLL datoteka, PDF dokumenata, Microsoft Office dokumenata, URL i HTML datoteka, PHP skripti, CPL datoteka, Visual Basic skripti, ZIP datoteka, Java JAR datoteka, Python skripti i drugih datoteka. Zbog modularnosti i odličnih skriptnih sposobnosti gotovo da nema ograničenja na vrstu datoteka koje može analizirati. To je jedan od glavnih razloga zašto ovaj alat zajednica razvija vrlo brzo i uspješno.

Cuckoo može prikazati sljedeće tipove rezultata:

- Tragove (engl. *trace*) win32 API poziva izvršenih od strane svih procesa koje je zloćudni program stvorio.

- Datoteke koje su kreirane, obrisane ili preuzete s Interneta od strane zloćudnog programa za vrijeme izvođenja.
- Ispis memorije (engl. *memory dump*) procesa
- Ispis mrežnog prometa u PCAP (engl. *packet capture***)** formatu
- Snimke ekrana tijekom izvođenja
- Cijeli ispis memorije

Cuckoo Sandbox se sastoji od centralnog programa za upravljanje koji rukuje izvođenjem i analizom malicioznog uzorka. Svaka analiza se pokreće na čistom i izoliranom virtualnom stroju. Na Slici 2.1 se može vidjeti da je infrastruktura Cuckoo-a sastavljena od stroja koji nadzire analizu ili nadziratelja (engl. *host*) i jednog ili više izoliranih strojeva (engl. *guest machine*). U nadziratelju je smješten centralni program za upravljanje dok se na virtualnim strojevima obavlja dinamička analiza uzorka. Nadziratelj pokreće glavnu kompenentu *sandboxa* koja upravlja cijeli proces analize dok su virtualni strojevi izolirana okruženja gdje se maliciozni programi sigurno izvode i analiziraju. Potrebno je još spomenuti dvije komponente pomoću kojih komuniciraju nadziratelj i virtualni stroj. Na strani nadziratelja postoji *ResultServer* koji barata s rezultatima koje šalje virtualni stroj tijekom analize. Na virtualnom stroju mora biti pokrenut *Agent* koji šalje podatke nadziratelju koristeći XML RPC (engl. *Remote Procedure Call*) Server. XML RPC je protokol koji koristi XML za šifriranje i HTTP kao mehanizam za prijenos podataka.



Slika 2.1 Osnovna arhitektura Cuckoo okruženja za dinamičku analizu [17]

3.2. Način rada Cuckoo Sandboxa

3.2.1. Komponente sustava

Osnovne komponente analizatora su *Scheduler*, *Analyzer* i *Monitor*. Glavna komponenta koja se izvršava na nadziratelju je *Scheduler* koji priprema virtualne strojeve, čeka i učitava nadolazeće zadatke odnosno analize malicioznih programa. Zatim otprema neobrađene zadatke u neku od virtualnih strojeva i pokreće *Analysis Manager* koji vodi brigu o cjelokupnom izvršavanju procesa analize i rukovanju s dodijeljenim virtualnim strojem. *Analysis Manager* također odabire slobodni virtualni stroj odnosno *sandbox* u kojem će se analiza izvršavati, priprema konfiguracijske datoteke, kreira potrebne direktorije i komunicira s virtualnim strojem. Nakon pripreme *Analysis Manager* pokreće analizu i zatim sprema, procesira i prikazuje rezultate analize.

Analyzer je komponenta koja se izvodi na virtualnom stroju. Zadužen je za izvršavanje zloćudnog programa, pokretanje monitora, rukovanje s poslužiteljem cijevi (engl. *pipe*

server) i upravljanje s pomoćnim modulima koji se nazivaju *Auxiliaries*. Poslužitelj cijevi prima obavijesti od *Monitora* ako je stvoren novi proces i kreirana ili izbrisana neka datoteka.

Monitor je komponenta zadužena za praćenje Windows API funkcija. To je ujedno srž analize uzorka. Praćenje se obavlja umetanjem (engl. *injection*) DLL-a (engl. *Dynamic-link library*) u memoriju procesa koji izvršava maliciozni program. Praćenje i presretanje Windows API funkcija se obavlja korištenjem metode *inline hooking*. Presreću se samo Windows API funkcije za koje je definiran potpis (engl. *Signature*). Potpis sadrži osnovne informacije o funkciji kao što su kategorija, povratna vrijednost i biblioteka kojoj funkcija pripada. Glavna ideja *inline hookinga* jest preusmjeravanje Windows API funkcija na neke vlastite funkcije *monitora* kako bi se mogla izvesti obrada prije ili poslije izvršavanja Windows funkcije. To može uključivati provjeru parametara, slanje obavijesti nadziratelju ili filtriranje poziva. Inline hooking presreće pozive pomoću kuka (engl. *hooks*), a svaka kuka je definirana potpisom. Kuke se postavljaju direktnim modificiranjem kôda unutar ciljne Windows funkcije. Obično se prvih nekoliko bajtova prepisuje skokom. Skok dopušta preusmjereno izvršavanje vlastitih procedura monitora prije nego se Windows funkcija počne izvršavati. Primjerice, ukoliko se dogodi poziv Windows API funkcije koja briše neku datoteku, taj se poziv presreće i poziva se procedura koja je opisana u potpisu za navedene funkcije brisanja datoteka. Procedura javlja nadziratelju preko poslužitelja cijevi poruku oblika "*FILE_DEL:putanja*". Takvu poruku parsira analizator i poduzima akcije koje uključuju spremanje datoteke prije nego ju zloćudni program obriše i dodavanje ovog incidenta u izvješća.

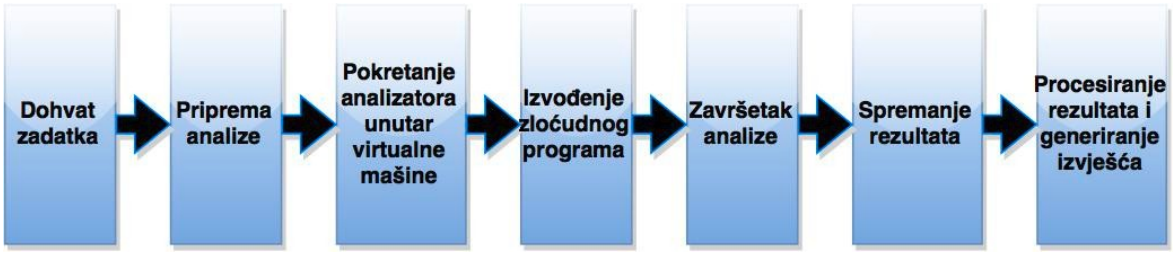
3.2.2. Proces analize

Najprije *Scheduler* dohvati zadatak iz lokalne baze podataka (*Slika 2.2*). Svaki zadatak predstavlja maliciozan uzorak koji je proslijeđen *Cuckoo Sandboxu* na obradu. Postoji nekoliko načina slanja zadatka *Cuckoo Sandboxu*:

- pomoću naredbenog retka
- preko mrežnog preglednika

- putem REST API-ja
- putem Python funkcija

Osnovni načini slanja malicioznog uzorka za sustav za analizu su detaljnije opisani u uputama za korištenje programske podrške koje se nalaze u privitku.



Slika. 2.2 Tijek izvođenja analize

U drugom koraku *Scheduler* pripremi analizu odnosno učitava konfiguracijske datoteke i podešava parametre sustava analize kao što su vrsta virtualnog stroja, IP adresa nadziratelja i virtualnog stroja, podešavanja maksimalnog vremena izvođenja i sl. Cuckoo na osnovu nekoliko konfiguracijskih datoteka kreira datoteku pod nazivom *analysis.conf* u kojoj se nalaze sve potrebne informacije o načinu izvršavanja analize. Nakon pripreme se pokreće čisti virtualni stroj. To znači da se virtualni stroj pokreće od zadnjeg snimka (engl. *snapshot*) koji sigurno radi dobro odnosno nije zaražen nekim zlođudnim programom i nema drugih poteškoća. Zatim se pokreće analizator unutar virtualnog stroja što je na Slici 2.2 označeno kao treći korak. Analizator najprije priprema okruženje za analizu. To se odnosi na stvaranje direktorija potrebnih za spremanje rezultata i na inicijaliziranje *logginga*. *Logging* je postupak bilježenja događaja koji su važni za analizu. Analizator zatim pokreće pomoćne module. Nakon toga analizator pokreće maliciozan program koji je trenutno na redu među neobrađenim zadacima tako da odabire vrstu paketa iz modula *Analysis Packages*. Svaki paket odgovara tipu datoteke zlođudnog programa. Analizator ovisno o tipu malicioznog uzorka pokreće program koji izvršava taj uzorak. Primjerice, ako je zlođudan uzorak *PowerPoint* prezentacija onda će analizator odabrati paket koji pokreće navedeni uzorak s programom *Microsoft PowerPoint*. Nakon pokretanja zlođudnog programa, analizator ubacuje monitor u memoriju procesa kako je opisano u prethodnom poglavlju. Monitor bilježi i sprema sve važne informacije za analizu

zloćudnog uzorka. Analiza završava kada završi izvođenje zloćudnog uzorka ili kada dođe to isteka vremena analize. Zadano maksimalno vrijeme analize je 60 sekundi, a po potrebi se može povećati ili smanjiti. Potom se spremaju rezultati i generiraju izvještaji.

3.2.3. Moduli

Cuckoo sadrži sedam različitih modula koje je lako prilagođavati i nadograđivati:

- *Analysis Packages*
- *Auxiliary*
- *Machinery*
- *Processing*
- *Reporting*
- *Signatures*

Analysis Packages je zapravo skup Python klasa koji određuje kako upravljati malicioznim uzorkom odnosno kako ga pokrenuti i kako komunicirati s njim. Ako korisnik nije zadao paket s kojim se pokreće zloćudni uzorak, analizator pomoću Python biblioteke *Magic* ili pomoću Windows naredbe *file* dohvaća podatkovni tip malicioznog uzorka i na temelju tog tipa poziva određeni paket iz modula *Analysis Packages*. Svaki paket sadrži skup putanji do programa s kojim se pokreće zloćudni uzorak i funkciju *start.* Funkcija *start* izvršava maliciozni uzorak pokretanjem programa uz pomoć skupa putanji i zatim pokreće monitor.

Modul *Auxiliary* su pomoćne klase koje se pozivaju prije izvođenja malicioznog uzorka. Zasada postoje četiri *Auxiliary* klase a to su: *Disguise*, *Human*, *Screenshots* i *Sniffer*. *Disguise* i *Human* služe za prikrivanje virtualizacijske okoline. Identifikacijsku oznaku *Windowsa* ili *Windows ProductId* vrlo često koriste zloćudni programi kako bi otkrili postavke Cuckoo Sandboxa. Zbog toga se prije izvršavanja zloćudnog uzorka *ProductId*, u *Windows* registru, postavi na slučajnu vrijednost. *Human* klasa služi za oponašanje korisnikovog rada na računalu. To se postiže korištenjem funkcija za pomicanje miša, za klikanje mišem i za pritiskanje tipki. *Screenshots* klasa hvata prikaz ekrana (engl.

Screenshot) i šalje slike nadziratelju u JPEG (engl. *Joint Photographic Experts Group*) formatu. Frekvencija hvatanja *screenshota* se može lako podesiti, a početna zadana vrijednost je jedna sekunda. *Sniffer* je pomoćna klasa koja služi za pokretanje i zaustavljanje ispisa mrežnog prometa. Za tu svrhu se koristi *tcpdump*. Tcpdump je alat koji preko naredbenog retka analizira mrežne pakete. Pokazuje TCP/IP i ostale pakete koji se šalju ili primaju mrežom na koju je spojeno računalo. Ovaj modul također osigurava da tcpdump ne hvata promet XML RPC *Agenta* i *ResultServer* paketa jer se oni odnose na komunikaciju *sandboxa* s nadziratelјem.

Machinery modul se sastoji od skupine klasa koje predstavljaju različite vrste virtualnih strojeva. Ovaj modul definira funkcije koje dohvaćaju, pokreću i zaustavljaju virtualnog stroja. Zasada postoji podrška za *esx*, *kvm*, *VirtuakBox*, *VMWare*, *Xenserver* i fizički *sandbox*.

Processing modul služi za procesiranje i normaliziranje rezultata. Postoji klasa u ovom modulu za svaki dio izvješća. Svaka od tih klasa sadrži funkciju *run* koja vraća rječnik s informacijama nastalim procesiranjem za vrijeme i nakon analize. Primjerice, klasa *VirusTotal* dohvaća informacije o zloćudnom uzorku od mrežnog servisa za skeniranje malicioznih programa i šalje ih *Reporting* modulu. Postoje i klase za procesiranje i normaliziranje osnovnih informacija o malicioznom uzorku ili za ispis memorije i mrežnog prometa.

Reporting modul koristi normalizirani izlaz *Processing* modula i kreira izvješća. Pohranjuje izvješće u HTML (engl. *HyperText Markup Language*), JSON (engl. *JavaScript Object Notation*), MAEC (engl. *Malware Attribute Enumeration and Characterization*) XML i *MongoDB* formatima.

Signatures traži specifične događaje koji se unaprijed definiraju. Ovaj modul je vrlo koristan ako se pokreće analiza velike količine zloćudnih programa a potrebni su samo određeni uzorci ili određene informacije. U tom slučaju se kreira signature u kojem se definiraju određeni uvjeti kao što su tip uzorka ili određeni niz znakova (engl. *string)*. *Signatures* funkcioniraju na način da se modul poziva za svaki Windows API poziv i vraća istinu ako analiza sadrži predefinirane uvjete a laž ako ne sadrži. Također je moguće

postaviti opis ili jačinu opasnosti skupine malicioznog uzorka što olakšava upotrebu i razumijevanje analize korisnicima koji primjerice ne razumiju Windows API pozive ili ispis memorije.

4. Odabir alata korištenih za dinamičku analizu

Postoji velik broj kvalitetnih programa koji služe za imitaciju i izolaciju operacijskog sutava i za praćenje sistemskih poziva. U ovom poglavlju su prikazani rezultati kraćeg istraživanja najpopularnijih alata koji pomažu u analizi zloćudnih programa.

4.1. Virtualizacijska programska podrška

U prethodnim poglavljima je spomenuto da *Cuckoo* koristi virtualni stroj kao *sandbox* u kojem se izvršava dinamička analiza malicioznog uzorka. Virtuanlni stroj je efikasan i izoliran duplikat stvarnog stroja odnosno računala [18]. To je najčešće korišteni tip *sandboxa* zato što imitira punokrvno računalo, nazvano gost, na glavnom računalu kojeg nazivamo nadziratelj. Osim što je analiziranje zloćudnog programa znatno sigurnije na virtualnom stroju, također je i puno brže i lakše vratiti sustav na stanje prije izvođenja zloćudnog uzorka. Koristeći *snapshot*, čisti virtualna stroj se pokreće za svega nekoliko sekundi dok su za formatiranje diska, reinstaliranje i podešavanje sustava potrebni sati.

Najpopularniji virtualizacijski programi koji mogu imitirati Linux operacijski sustav su: *VirtualBox* [19], *VMware Workstation* [20], *VMware Player* [21], *KVM* [22], *QEMU* [23] i *XEN* [24]. Osnovne informacije za svako od ovih virtualizacijskih okruženja su prikazane u Tablici 3.1. Za sve ove programe postoji već gotova Cuckoo Sandbox-a podrška tako da je moguće koristiti bilo koje od ovih virtualizacijskih rješenje bez dodavanja posebnog *Machinery* modula.

Tablica 3.1 Usporedba virtualizacijskih programa [25], [26], [27]

Virtualizacijska programska podrška	**Tip**	**Proizvođač**	**Podržana virtualna arhitektura**	**Stvaranje slike sustava**
VirtualBox	virtzualizacija	Oracle	x86, AMD64	da

QEMU / KVM	Sklopovska virtualizacija AMD-V i Intel VT-x, paravirtualizacija	Red Hat	x86, ARM, SPARC, MIPS, MIPS64, m68k, PowerPC	da
VMware Workstation	virtualizacija i paravirtualizacija	VMware	x86, AMD64	da
VMware Player	virtualizacija	VMware	x86, AMD64	ne
XEN	paravirtualizacija	Citrix Systems	x86, AMD64, ARM	da

VMware virtualni strojevi su najstabilniji i najpouzdaniji [29]. Postoji Vmware Playre kao besplatan program koji nudi samo osnovne mogućnosti kreiranja i pokretanja virtualnih strojeva. Zbog toga nije u mogućnosti snimiti stanje virtualnog stroja. Za razliku od VMware Player-a, VMware Workstation nije besplatan ali nudi raznolike mogućnosti kao što su višestruka stvaranja slike sustava, snimanje rezultata unutar virtualnog stroja, kloniranje strojeva i sl. Sve ove mogućnosti nisu jednostavne za korištenje ali je instalacija brza i integracija između operacijskih sustava je vrlo dobra.

KVM (engl. *Kernel-based Virtual Machine*) je modul jezgre operacijskog sustava Linux koji dopušta korisničkom programu upotrebu mogućnosti sklopovske virtualizacije različitih vrsta procesora. QEMU (engl. *Quick Emulator*) izvršava kôd virtualnog stroja direktno na procesoru nadziratelja. Qemu podržava virtualizaciju kada se izvršava unutar Xen hipervizora ili koristeći KVM modul jezgre operacijskog sustava Linux. Podržava automatsku promjenu veličine virtualnih diskova i pokretanje na nadziratelju bez administracijskih ovlasti. Xen je slično kao i KVM virtualizacijski hipervizor koji nudi popuni virtualizacijski stroj s mogućnostima približno identičnim kao i pravo računalo.

VirbualBox ima najbogatiji set mogućnosti i najbrže performanse. Ima vrlo jednostavno korisničko sučelje. Iako Cuckoo Sandbox ne ovisi o virtualizacijskom rješenju,

preporučeno je korištenje VirtrualBox-a jer je VirtualBox korišten prilikom razvoja i testiranja. Zbog toga je izabran VirtualBox pri izradi sustava za analizu Linux izvršnih datoteka.

4.2. Alati za praćenje sistemskih poziva

4.2.1. Usporedba alata za praćenje sistemskih poziva

Kao što je navedeno u prethodnim poglavljima, glavni dio monitora koji prati izvođenje malicioznih programa na Linux okruženju jest nadziranje sistemskih poziva. Za razliku od Windowsa, na Linux analizatoru nije potrebno koristiti injekciju DLL-a i alociranje memorije jer postoje gotovi alati koji služe za praćenje sistemskih poziva. Strace, DTrace i Sysdig su alati koji se trenutno najviše koriste za nadzor Linux sustava.

Strace je baziran na alatu ptrace (engl. *process trace*) kojeg koriste mnogi *debuggeri* za promatranje procesa koji se izvodi. *Ptrace* funkcionira tako što pauzira promatrani proces za svaki sistemski poziv kako bi debugger pročitao stanje. *Ptrace* mehanizam omogućuje strace-u pauziranje promatranog procesa kod svakog sistemskog poziva, dekodiranje sistemskog poziva i ponovno pokretanje izvođenja promatranog procesa. Problem je što se svaki puta kod pozivanja sistemskog poziva događa prijelaz iz korisničke razine u razinu jezgre. Ta se pojava naziva razmjena konteksta (engl. *context switch*). Događa se nekoliko razmjena konteksta za svaki promatrani proces što često dovodi do potpunog zaustavljanja procesa dok *Strace* ne obavi dekodiranje. *Strace* je jednostavan, prati samo sistemske pozive, dostupan je na gotovo svim *Linux* distribucijama te generira automatiziran i smislen izlaz za svaki tip sistemskog poziva. Međutim može prouzrokovati značajna i ponekad enormna smanjenja performansi usporavajući promatranu aplikaciju čak i preko 100 puta [29]. Zbog toga nije prikladan za korištenje na većim produkcijskim sustavima.

DTrace je sofisticiraniji i puno efikasniji alat od srace-a. *Dtrace* uzima skripte koje su pisane u jeziku domene kojeg zovemo D, prevodi ih u *bytecode* i ubacuje *bytecode* na određena mjesta u jezgri. *Bytecode* se izvršava kada dođe do određenih događaja kao što su pozivi određenih funkcija u promatranog programu ili sistemski pozivi. Dtrace je vrlo

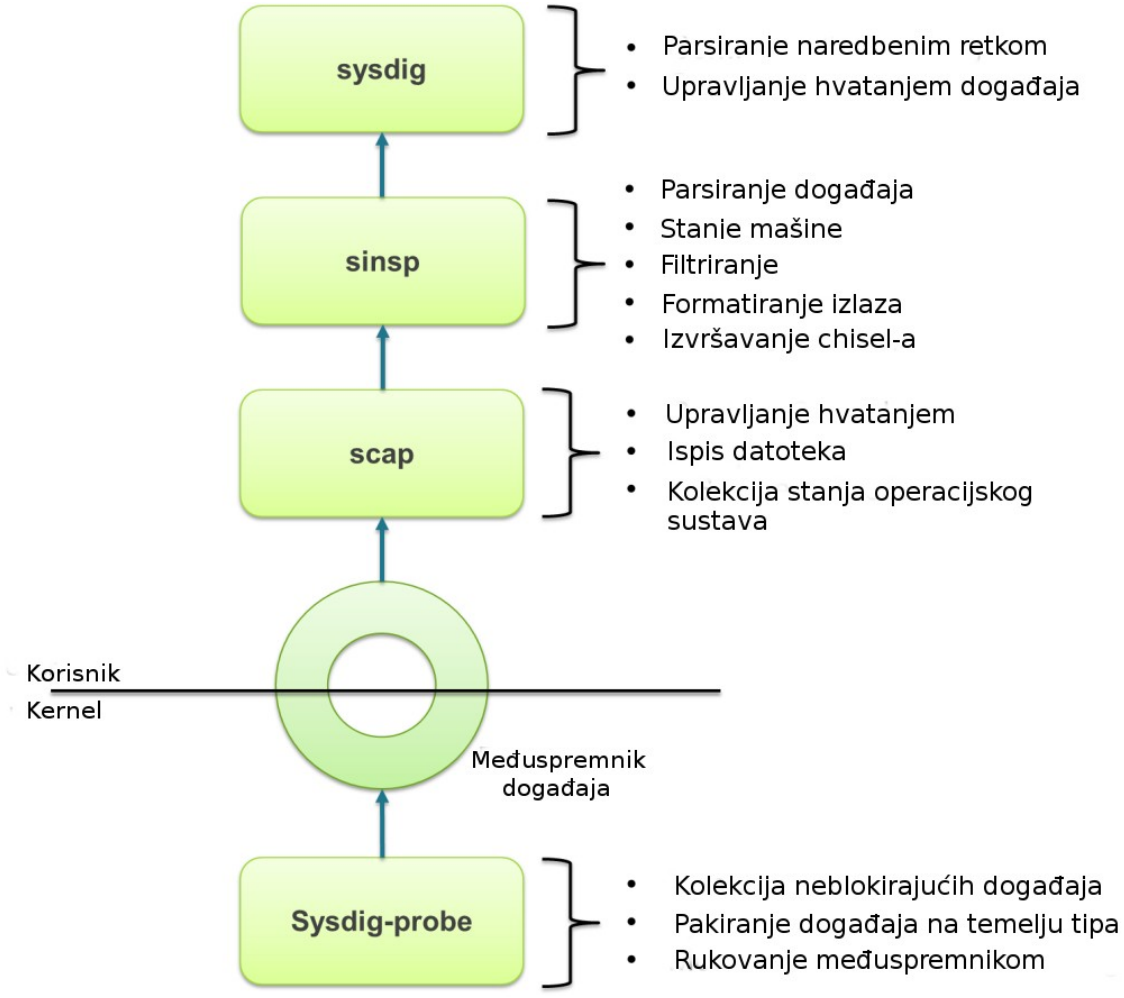
fleksibilan zahvaljujući prilagodljivosti jezika D koji se može ubaciti na različita mjesta u jezgri operacijskog sustava. Praćenjem s Dtrace-om nije potrebna razmjena konteksta kod procesiranja sistemskih poziva. Jedini problem je što su skripte u nekom D programskom jeziku često kompleksne i zbog toga mogu znatno usporiti izvođenje. Također je moguće da iako su skripte vrlo brze dođe do degradacije performansi jer je potrebno pokrenuti više skripti u isto vrijeme. Dtrace je vrlo moćan i dinamičan alat ali pisanje skripti zahtjeva značajna tehnička znanja kako bi se optimiziralo praćenje.

Sysdig je jedan od najnovijih i najnaprednijih alata za praćenje sistemskih poziva. Osim praćenja sistemskih poziva nudi uvid u mrežne aktivnosti i pohranu podataka. Sysdig je alat koji objedinjuje alate za praćenje uključujući *strace*, *htop*, *lsof* i *tcpdump* [13]. Osim toga dodaje i različite druge mogućnosti uz pomoć tzv. *chiselsa*. *Chisels* su bogata biblioteka skripti u progoramskom jeziku *Lua* koje omogućuju dodatna oblikovanja i filtriranja događaja i informacija koje generira sysdig. Primjer *chiselsa* su ispis liste najsporijih procesa ili ispis liste datoteka s najvećim brojem operacija čitanja i pisanja. Osim pregršt različitih mogućnosti koje nudi, sysdig neće usporiti procese koje prati ako dođe do većeg broja događaja.To također znači da chisels skripte ne moraju biti pažljivo optimizirane kao što je to slučaj kod D skripti u Dtrace-u.

Starce je vrlo star alat koji je predinstaliran na gotovo sve Unix operacijske sustave tako da je često jedini izbor alata za praćenje. Generira detaljne informacije za svaki sistemski poziv ali ima loš utjecaj na performanse sustava zbog čega nije najbolji izbor za produkcijska okruženja. Dtrace je efikasan i fleksibilan alat koji nudi ogroman mogućnosti praćenja stvari koje se dešavaju unutar i izvan jezgre. Problem je što uvelike ovisi o skriptama koje se izvršavaju unutar jezgre pa iziskuju visok stupanj tehničkog znanja. Sysdig je prilagodljiv alat koji pruža najviše mogućnosti s bogatim i jednostavnim načinima obrade i filtriranja informacija kao što su nazivi datoteka ili IP adresa umjesto brojeva opisnika datoteka. Uz to se ponaša jako dobro na produkcijskim okruženjima što je vrlo važno za dinamičku analizu kojoj je cilj u što manje vremena skupiti što više informacija o promatranom zloćudnom uzorku. Zbog toga je izabran sysdig kao monitor pri dinamičkoj analizi izvršnih datoteka na Linuxu.

4.2.2. Sysdig

Sysdig ima vrlo sličnu arhitekturu alatima poput *tcpdump*-a ili *wireshark*-a. Najprije se uhvate događaji u jezgri pomoću *drivera* koji se zove *Sysdig-probe* (Slika 3.1). Sysdig-probe *handler* rukuje s događajima na jednostavan način tako što kopira informacije o događajima u dijeljeni međuspremnik. Na taj način je zamrznuto izvođenje jezgre dok se *handler* ne dovrši kopiranje. Ostatak praćenja se događa na korisničkoj razini.



Slika.3.1 Arhitektura alata sysdig [30]

Međuspremnik događaja je mapiran u korisnički prostor tako da pristup bude što brži. Modul jezgre operacijskog sustava sysdiga napisan je da što manje opterećuje operacijski

sustav i zato sadrži samo najnužniju funkcionalnost, dok su više funkcije sortiranja i filtriranja smještene u korisnički prostor. Koriste se biblioteke libscap i libsinsp za čitanje, dekodiranje i parsiranje događaja. Libscap služi za praćenje operacija s podacima dok libsinsp uključuje sofisticirano praćenje stanja funkcionalnosti, filtriranje, dekodiranje događaja i Lua kompilator za izvršavanje *chiselsa*. Na vrhu je sysdig kao omotač (engl. *wrapper*) oko tih biblioteka. Velika prednost sysdiga jest ta da ako, libsinsp ili libscap nisu dovoljno brzi da zadrže sve događaje koji dolaze onda se sustav neće usporiti kao kod npr. stracea. U tom slučaju će se napuniti međuspremnik događaja i sysdig-probe će jednostavno ispuštati nadolazeće događaje. Zbog toga će se izgubiti mali dio informacija ali se neće usporiti računalo ili procesi. To je glavna prednost arhitekture sysdig-a zbog čega je idealan za produkcijska okruženja. Jedini nedostatak ovakve arhitekture jest nemogućnost pokretanja više od jedne instance sysdig-a.

Najjednostavnija i najmanje korisna naredba je jednostavno pozivanje programa sysdig u konzoli bez ikakvih parametara. U tom slučaju će sysdig ispisati sve sistemske događaje u sljedećem formatu:

%evt.num %evt.time %evt.cpu %proc.name (%thread.tid) %evt.dir %evt.type %evt.args

Navedene informacije o događaju označavaju:

- evt.num je inkrementalni broj koji označuje pojedini događaj.
- evt.cpu je broj procesora na kojem se izvršava događaj.
- proc.name označuje naziv procesa koji je generirao događaj.
- thread.id je identifikacijski broj dretve (engl. *thread id*, skraćeno TID) koja je generirala događaj. Za jednodretvene procese thread.id označava i PID.
- evt. dir je smjer događaja. > označava ulazne događaje dok < označava izlazne događaje
- evt.type je naziv događaja. Ako se radi o sistemski pozivima evt.type označava naziv sistemskog poziva.

- evt.args je lista argumenata događaja. Ukoliko je događaj sistemski poziv, lista odgovara argumentima sistemskog poziva iako su neki argumenti isključeni zbog poboljšanja performansi i jednostavnosti.

Za razliku od strace programa, sysdig razlikuje ulazne i izlazne događaje, te raspisuje mrežne i datotečne opisnike u čovjeku čitljivijem obliku. Takav ispis omogućuje direktno čitanje naziva datoteka i IP adresa te olakšava praćenje višeprocesorskih sustava. Sysdig ima odlične mogućnosti filtriranja pa je tako primjerice moguće uz poziv sysdig-a dodati argument *proc.pid* koji označava PID procesa čije događaje sysdig promatra.

Sustav za analizu Linux izvršnih datoteka je vrlo sličan sustavu za analizu datoteka na Windowsima. Tijek izvođenja analize i komponente sustava su potpuno isto. Najveća razlika je implementacija monitora i analizatora. Monitor uključuje pokretanje sysdig-a za proces koji predstavlja zloćudni proces, filtriranje i parsiranje izlaza. Analizator je vrlo sličan. Najveća razlika je izostanak modula *Analysis Packages* i detektiranja tipa datoteke zloćudnog uzorka jer je dodana podrška samo za izvršne datoteke.

U prethodnim poglavljima je naznačeno da za dinamičku analizu malicioznog uzorka najprije treba pokrenuti zloćudni program pa tek onda promatrati ponašanje i promjene na sustavu. Pokretanje Linux izvršnih datoteka se može izvršiti na nekoliko načina. Odsječak kôda prikazuje funkciju koja pokreće izvršavanje malicioznog programa.

5. Sustav za analizu Linux izvršnih datoteka

5.1. Pokretanje analize

```
def start(self, path):
      os.chmod(path, 0755)
      process = subprocess.Popen(['/bin/bash', '-c', path])
      return process.pid
```

U drugoj liniji se poziva naredba odnosno sistemski poziv koji mijenja dozvolu pristupa malicioznom uzorku. Nakon što se pokrene analiza i preda putanja do malicioznog uzorka, Cuckoo nadziratelj pošalje uzorak *sandboxu*. Nakon slanja se ponište dozvole pristupa tako da je potrebno postaviti dozvolu za izvršavanje programa. Oktalna vrijednost 755 označava da vlasnik datoteke ima dozvole čitanja, pisanja i izvršavanja dok svi ostali imaju dozvolu za čitanje i izvršavanje datoteke. Nula ispred vrijednosti prava pristupa eksplicitno označava stanje specijalnih bitova. Nula znači da nisu postavljeni specijalni bitovi.

U ovom radu za pokretanje programa, odnosno za stvaranje procesa, korišten je Python modul *subprocess*. U trećoj liniji funkcije *start*, kreira se proces za pokretanje malicioznog programa pozivom *subprocess* konstruktora *Popen* s argumentima *'/bin/bash'* i *'-c'*. *'/bin/bash'* poziva Unix ljusku koja izvršava maliciozni program dok '-c' argument čita i izvršava naredbu koja se nalazi na prvom sljedećem argumentu i zatim završava s izvođenjem [30]. Funkcija na kraju vraća PID vrijednost stvorenog procesa. Izvršavanje malicioznog uzorka se moglo ostvariti pozivom funkcije *os.system* ili koristeći *subprocess.Popen* i *shell=True* argument ali je zbog sigurnosnih rupa izabran prikazan način.

Isto kao i kod analize malicioznih programa na Windows operacijskom sustava, nakon pokretanja malicioznog uzorka potrebno je ubaciti monitor. Monitor se pokreće asinkrono u drugoj dretvi kako Python ne bi čekao kraj izvršavanja dretve monitora. Za tu svrhu je korištena klasa Thread iz standardne Python biblioteke *threading*:

```
        monitor_thread = threading.Thread(target=execute,
args=(pid,))
        monitor_thread.start()
```

Prva linija inicijalizira dretvu s argumentima *target* i *args*. Target označava pozivni objekt odnosno funkciju koja će se pozvati prilikom pokretanja dretve. *Args* su argumenti funkcije koja se poziva. Druga linija pokreće funkciju *execute* koja je zadana argumentom *target*. (*Kôd 4.1*) prikazuje funkciju koja pomoću subprocess modula pokreće monitor. Subprocess.Popen pokreće sysdig naredbu koja prati sve događaje nastale od zadanog procesa i njegove djece i u realnom vremenu šalje događaje na obradu. *sudo* argument označava izvođenje programa s administratorskim (engl. *superuser)* ovlastima . Izlaz naredbe se prosljeđuje na cijev koju čita iterator i šalje liniju po liniju na procesiranje sysdig parseru.

```
def execute(pid):
    parser = SysdigParser()
    sysdig_monitor = subprocess.Popen(['sudo', 'sysdig',
'proc.pid = ', '%s' % pid, 'or proc.apid = ', '%s' % pid],
                                      stdout=subprocess.PIPE)
    lines_iterator = iter(sysdig_monitor.stdout.readline,
b"")

    for event in lines_iterator:
        parser.process(event)
```

Kôd 4.1 Funkcija koja poziva sysdig monitor

5.2. Parsiranje sistemskih poziva i poduzimanje akcije

U prethodnim poglavljima je opisano praćenje sistemskih poziva i sysdig kao alat korišten za tu svrhu. Nakon pokretanja malicioznog programa i monitora, pokreće se parser kojemu se šalje svaki događaj kojeg sysdig zabilježi. Parser najprije pohrani u varijable svih osam informacija koje prikazuje sysdig:

```
        event_info = event.split()
```

```
            (evt_num, evt_time, evt_cpu, proc_name, thread_tid,
        evt_dir, evt_type), evt_args = \
                event_info[:7], event_info[8:]
```

Zadnja varijabla su argumenti događaja kojih može biti nekoliko ili ne mora postojati niti jedan. Najvažnija informacija o događaju je tip događaja odnosno *evt_type* koji u slučaju sistemskih poziva označava naziv sistemskog poziva. Promatraju se dvije skupine sistemskih poziva. To su sistemski pozivi zaduženi za upravljanje datotečnim sustavom i sistemski pozivi za mrežne operacije. Tipovi događaja koje analizator procesira su: *open*, *creat*, *unlink*, *unlinkat*, *read*, *write* i *connect*. Vrlo je lako dodati praćenje bilo kojeg drugog sistemskog poziva. U nastavku je prikazan dio kôda koji se izvršava ako promatrani zloćudni program napravi novu datoteku.

```
            if (evt_type == 'open' or evt_type == 'creat') and
        evt_dir == '<':
                if any("O_RDONLY" in evt_arg for evt_arg in
        evt_args):
                    return
                for evt_arg in evt_args:
                    if evt_arg.startswith('name='):
                        file_path = evt_arg[evt_arg.find("(")
        +1:evt_arg.find(")")]
                        add_file(file_path)
```

Kôd 4.2. Primjer procesiranja sysdig događaja

Ako je datoteka otvorena samo za čitanje onda se preskače takav događaj. U ostalim slučajevima se parsira naziv i putanja stvorene datoteke. Zatim se putanja funkcijom *add_file* dodaje u listu novonastalih datoteka (*Kôd 4.2*). Potom se šalje obavijest o novonastalim datotekama koje se na kraju *uploadaju* na nadziratelj. Slična procedura se izvršava ukoliko je došlo do brisanja, premještanja ili preimenovanja datoteka. U slučaju sistemskog poziva *connect*, iz argumenata se izvlače izvorišna i odredišna adresa te se rezultati zapisuju u datoteku *linux.log* koju na kraju analize parsira odgovarajući *Processing* modul.

Osim praćenja sistemskih poziva, u analizu je dodan ispis svih dijeljenih biblioteka koje koristi zlćudni program. Za tu svrhu korišten je alat ldd (engl. *List Dynamic Dependencies)* [33] koji u datoteku *linux.log* zapisuje dijeljene biblioteke.

```
            with open('linux.log', "a") as outfile:
                subprocess.call(['ldd', self.target],
stdout=outfile)
```

Također je dodana klasa *Linux* za module *Processing* i *Reporting*. Linux klasa za procesiranje parsira zapisane IP adrese i listu dijeljenih biblioteka te rezultate šalje *Reporting* modulu. Linux klasa za *Reporting* modul zapisuje navedene rezultate u izvješća.

5.3. Pomoćni moduli

Prije izvršavanja zlćudnog programa i pokretanja monitora, pokrenu se svi pomoćni moduli. Implementirana su dva pomoćna modula: *Screenshot* i *Sniffer*. Imaju istu uloge kao i kod Windows analizatora. *Screenshot* pomoćni modul hvata prikaz ekrana *sandboxa* i šalje ga na nadziratelj.

```
def take():
    window = gtk.gdk.get_default_root_window()
    window_size = window.get_size()
    pix_buffer = gtk.gdk.Pixbuf(gtk.gdk.COLORSPACE_RGB,
False, 8, window_size[0], window_size[1])
    pix_buffer = pix_buffer.get_from_drawable(window,
window.get_colormap(), 0, 0, 0, 0, window_size[0],
window_size[1])
    if pix_buffer is not None:
        return pix_buffer
    else:
        print "Unable to get the screenshot."
```

Kôd 4.3 Funkcija koja dohvaća prikaz ekrana

Za tu svrhu se koristi standardna procedura za hvatanje prikaza ekrana uz pomoć Python biblioteke gtk (*Kôd 4.3*). Najprije se dohvati prozor i definira veličina prozora. Zatim se inicijalizira međuspremnik (engl. *buffer*) koji predstavlja objekt klase *Pixbuf*. Taj objekt

služi za operacije i rukovanje slikama. Funkcija *get_from_drawable* pretvara sliku u RGB (engl. *Red Gree Blue*) prikaz i sprema ga u međuspremnik koji je tipa *Pixbuf*. Zadana vrijednost pozivanja funkcije *take* je svakih jednu sekundu a nalazi se u varijabli *SHOT_DELAY*. Na testiranom virtualnoj stroju s 1 GB memorije s nasumičnim pristupom (engl. *Random Access Memory*), vrijeme potrebno za dohvat prikaza ekrana i slanje na nadziratelj iznosi oko 0.3 sekunde. To znači da vrijednost varijable *SHOT_DELAY* ne bi smjela biti manja od 0.3 sekunde jer će se zahtjevi za dohvaćanjem prikaza ekrana brže stvarati nego što se stignu obraditi.

Drugi pomoćni modul služi za ispis mrežnog prometa u PCAP formatu pomoću alata tcpdump. Ispis se sprema u datoteku *dump.pcap* i potom se šalje na nadziratelj. Tcpdump je pozvan s opcijama -U, -q, -s 0, -n, i -w *dump.pcap*. -U zastavica označuje da se paket zapisuje u datoteku čim je paket spremljen. Inače se paketi zapisuju u datoteku tek kada se napuni međuspremnik.-q označava brži način rada a to se postiže tako da se odbacuju detaljne informacije u vezi protokola. -s označava duljinu *snapshota* (engl. *snaplen*) odnosno količinu podataka za svaki okvir koja se sprema u datoteku. Zastavicom -n se postiže zapisivanje pravih ip adresa i *portova* bez konverzije u imena. -w označava ispis u datoteku. Važno je da tcpdump ispisuje samo mrežni promet zloćudnog programa. Stoga je potrebno isključiti praćenje koje se odnosi na komunikaciju *sandboxa* i nadziratelja kao što je prikazano u Kodu 4.4.

```
pargs.extend(["and", "not", "(", "dst", "host", host, "and",
"dst", "port", str(CUCKOO_GUEST_PORT), ")", "and", "not",
"(", "src", "host", host, "and", "src", "port",
str(CUCKOO_GUEST_PORT), ")"])

pargs.extend(["and", "not", "(", "dst", "host",
resultserver_ip, "and", "dst", "port", resultserver_port,
")", "and","not", "(", "src", "host", resultserver_ip, "and",
    "src", "port", resultserver_port, ")"])
```

Kôd 4.4 Dio kôda kojim se isključuju nepotrebna praćenja mrežnog prometa

Prva naredba dodaje argumente koji isključuju praćenje prometa XMLRPC agenta, dok druga dodaje argumente koji isključuju praćenje prometa *sandboxa* s *ResultServerom* (Kôd 4.4). Nakon ovog dijela kôda poziva se tcpdump s navedenim argumentima preko *subprocess* modula.

6. Rezultati analize

U trećem poglavlju su nabrojani tipovi izvješća koje generira Cuckoo Sandbox na kraju analize. Svi rezultati se spremaju u direktorij *storage* koji se nalazi u Cuckoo repozitoriju. Unutar tog direktorija su direktoriji *analyses* i *binaries*. *Analyses* sadrži *logove*, izvješća, datoteke, prikaze ekrana tijekom izvođenja i ispis mrežnog prometa svake analize dok *binaries* sadrži samo zloćudne programe koji su se analizirali.

Prva datoteka koja se kreira prilikom analize u *analysis* direktoriju je a*nalysis.log.* Datoteka *analysis.log* prikazuje osnovne informacije prilikom izvođenja. U nastavku je prikazan primjer *analysis*.*log* datoteke nakon uspješne analize:

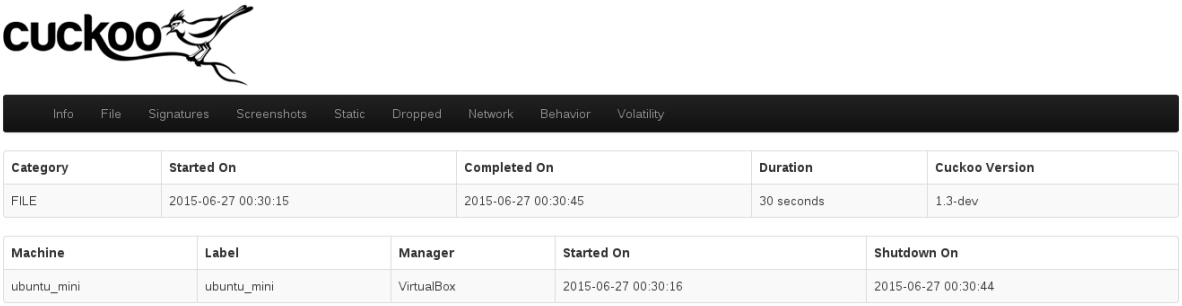
2015-06-27 00:30:15,650 [root] DEBUG: Starting analyzer from: /home/guest/asrrxu
2015-06-27 00:30:15,674 [root] DEBUG: Storing results at: /home/guest/tmp
2015-06-27 00:30:17,275 [root] DEBUG: Started auxiliary module Screenshots
2015-06-27 00:30:17,944 [root] INFO: Added new process to list with pid: 1671
2015-06-27 00:30:44,870 [root] INFO: Analysis timeout hit, terminating analysis.
2015-06-27 00:30:45,448 [root] INFO: Analysis completed.

Zatim se puni direktoriji *shots koji* sadrži prikaze ekrana tijekom analize. U direktorij *files* se uploadaju datoteke kreirane od strane zloćudnog programa tek kada je analiza završena. Datoteke koje zloćudni program pokušava obrisati, odmah se uploadaju na nadziratelj kako ne bi došlo do gubitka podataka. Izvješća se nalaze unutar *analysis* direktorija u direktoriju *reports*. Izvješća se spremaju u četiri različita formata:

- HTML
- JSON
- MAEC XML
- metadata XML

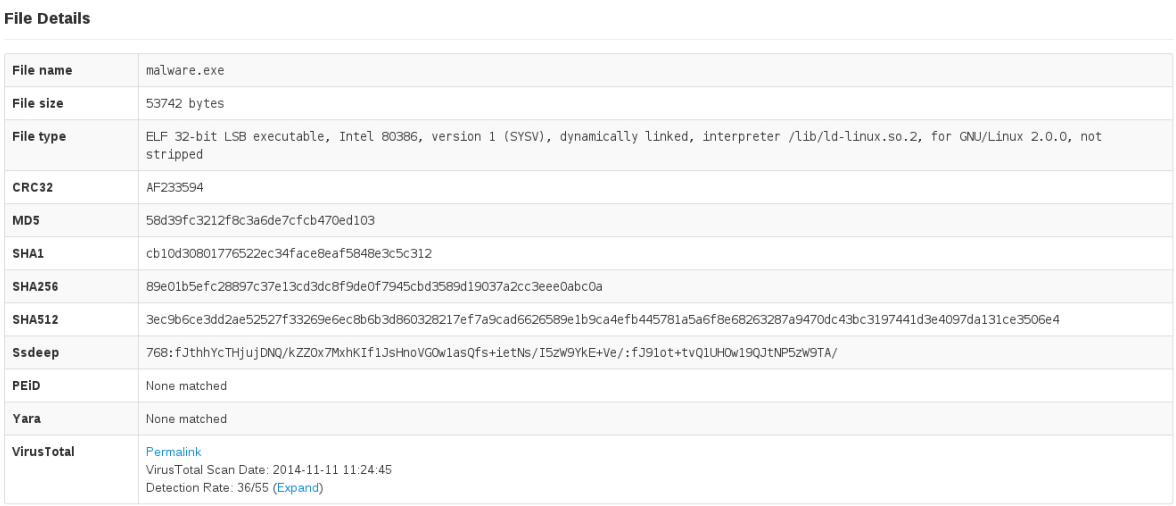
Iako izvještaj u JSON formatu daje najviše informacija, izvještaj u HTML formatu je najkorisniji jer je izgled vrlo pristupačan i može se pokrenuti u preglednik. Zbog toga ovaj

način prikaza koriste različiti servisi za prikupljanje i analizu zlćudnih programa kao što je *Malwr* [32]. Za svaki od dijelova HTML i JSON izvještaja mora postojati *reporting* modul. Sljedeći primjeri prikaza HTML izvještaja su izvještaji analize trojanskog konja. Svaki HTML izvještaj započinje s osnovnim informacijama o analizi kao što su vrijeme pokretanja analize ili naziv virtualnog stroja na kojem je pokrenuta analiza (*Slika 6.1*)



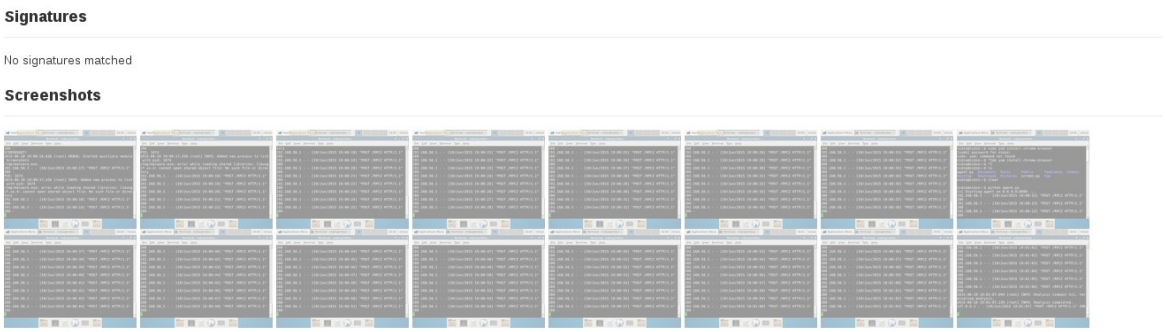
Slika 6.1 HTML prikaz osnovnih informacija o analizi

Zatim slijede detalji o analiziranom zlćudnom programu prikazani na Slici 6.2. Detalji se ponajprije odnose na različite načine za prikaz sažetka (engl. *hash*) od kojih su najvažniji MD5 i SHA1. Posebno su korisne informacije dobivene od servisa *VirusTotal* [34]. Riječ je o vrstama detekcije analiziranog zlćudnog programa od strane najboljih antivirusnih kompanija.



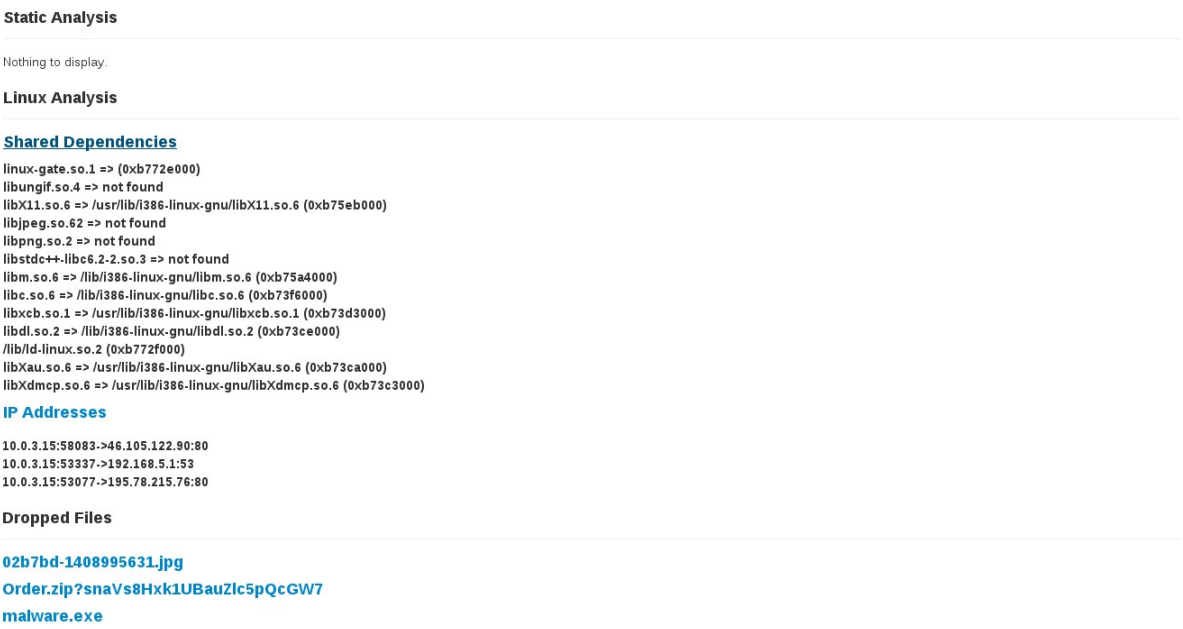
Slika 6.2. HTML prikaz detalja datoteke zlćudnog programa

Nakon detalja o datoteci u izvještaju su prikazani potpisi (*Slika 6.3*). Potpisi su detaljnije objašnjeni u prijašnjim temama a s obzirom na to da se nije radilo o automatiziranom slanju veće količine zlućudnih programa, nije bilo potrebe za definiranjem potpisa. Prikazi ekrana su slike u JPEG formatu poredane kronološki a moguće ih je povećati.



Slika 6.3 HTML prikaz potpisa analize i prikaza ekrana tijekom analize

Zatim slijede informacije o statičkoj analizi koja je definirana samo za Windows PE (engl. *Portable Executable*) datoteke. Dodan je *Linux Analysis* (*Slika 6.4*) dio koji se odnosi na Linux klase opisane u prošlom poglavlju. Ispisani su dijeljene biblioteke malicioznog programa te izvorišne i odredišne IP adrese. Na kraju se nalaze informacije o svim datotekama koje je zlućudni program kreirao, obrisao ili mijenjao. Klikom na svaku od datoteka se mogu dobiti detaljnije informacije kao na Slici 6.2.



Slika 6.4 Zadnji dio HTML prikaza izvješća

JSON ispis sadrži iste informacije kao i HTML izvješće uz dodatne podatke iz loga i *strings*. U *strings* rječniku su zapisani svi nizovi znakova dohvaćeni iz sadržaja datoteke zloćudnog programa.

MAEC izvješće sadrži u XML obliku iste informacije kao i HTML izvješće dok je *metadata* izvješće skraćeno izvješće koje sadrži samo podatke o MD5, SHA1, nazivu i tipu zloćudnog programa.

Zaključak

U trenutku kada usporedno raste količina podataka i opasnost od malicioznih programa, jasna je potreba za novim metodama otkrivanja zloćudnih programa. Međutim, otkrivanje i uklanjanje zloćudnih programa više nije dovoljno. U današnjem svijetu je postalo vrlo važno razumjeti kako zloćudni programi rade, što rade i razumjeti kontekst, motivaciju i namjeru napada. Pomoću Cuckoo Sandboxa je znatno olakšano razumijevanje incidenata kako bi se na njih moglo odgovoriti i u budućnosti zaštiti.

Ovaj rad se dotakao tema vezanih za analizu zloćudnih programa uključujući statičku i dinamičku analizu te korištenje *sandboxa* u analizi. Dinamička analiza pokazuje u kojem smjeru ide izvođenje programa dok statička analiza pokazuje zašto izvođenje ide u tom smjeru. Nije preporučljivo provoditi dinamičku analizu bez određene zaštite. Zbog toga se analiza izvodi u izoliranom okruženju a omogućuje ga virtualni stroj.

Cuckoo Sandbox besplatno pruža dinamičku analizu zloćudnih programa. Nakon pokretanja zloćudnog programa, uključuje monitor koji prati ponašanje programa i na kraju šalje rezultate analize. Rezultati su izvješća koja opisuju promatrane akcije koje je zloćudni uzorak izvodio tijekom analize na Windows operacijskim sustavima. U sklopu rada unaprijeđen je Cuckoo Sandbox dodavanjem analize za Linux izvršne datoteke praćenjem sistemskih poziva. Postoji prostor za nadogradnju sustava zato što je cilj dinamičke analize izvući što više korisnih informacija o ponašanju zloćudnog programa u što kraćem vremenu. Zasada je postignuto bilježenje informacija kao što su kreirane i obrisane datoteke od strane zloćudnog programa, osnovni podaci o zloćudnom programu, prikazi ekrana ili ispis mrežnog prometa.

Literatura

[1] Cuckoo Foundation, http://www.cuckoosandbox.org, 27.06.2015.

[2] M. Sikorski, A. Honig: *Practical Malware Analysis*, San Francisco: No Starch Press, 2012.

[3] B. Chess, J. West: *Secure Programming with Static Analysis*, Addison-Wesley, 2007.

[4] D. Oktavianto, I. Muhardianto: *Cuckoo Malware Analysis*, Birmingham: Packt Publishing, 2013.

[5] System call, https://en.wikipedia.org/wiki/System_call, 28.06.2015.

[6] Unix System Calls, http://www.softpanorama.org/Internals/unix_system_calls.shtml, 28.06.2015.

[7] *A Policy-driven, Host-Based Intrusion Detection System*, ISOC Symposium on Network and Distributed System Security, San Diego, 2002.

[8] Garfinkel, Pfaff, Rosenblum. Ostia: *A delegating architecture for secure system call interposition*. Proceedings of Network and Distributed Systems Security Symposium, San Diego, 2004.

[9] D. Farmer, W. Venema: *Forensic Discover*, Boston: Addison-Wesley, 2005.

[10] strace(1) - Linux man page, http://linux.die.net/man/1/strace, 28.06.2015.

[11] dtrace(1) - Linux man page,http://linux.die.net/man/1/dtrace, 28.06.2015.

[12] Frank Ch. Eigler, https://sourceware.org/systemtap, 28.06.2015.

[13] Draios, Inc., http://www.sysdig.org, 28.06.2015.

[14] Niels Provos, https://www.provos.org/index.php?/categories/2-Systrace, 27.06.2015.

[15] Hisham Muhammad, http://hisham.hm/htop, 28.06.2015.

[16] lsof(8) - Linux man page, http://linux.die.net/man/8/lsof, 28.06.2015.

[17] Cuckoo Foundation, http://docs.cuckoosandbox.org/en/latest/_images/architecture-main.png, 17.06.2015.

[18] R. P. Goldberg*: Survey of virtual machine research*. IEEE Computer Magazine, 1974(June), 34–45

[19] Oracle Corporation, https://www.virtualbox.org, 28.06.2015.

[20] VMware, Inc., http://www.vmware.com/products/workstation, 28.06.2015.

[21] VMware, Inc., http://www.vmware.com/products/player, 28.06.2015.

[22] Red Hat, Inc., http://www.linux-kvm.org, 28.06.2015.

[23] Red Hat, Inc., http://www.qemu.org, 28.06.2015.

[24] Citrix Systems, Inc., http://www.xenproject.org, 28.06.2015.

[25] Citrix Systems, Inc., http://wiki.xenproject.org/wiki/Xen_Project_Release_Features, 27.04.2015.g

[26] Comparison of platform virtualization software, https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software, 27.04.2015.

[27] Software Insider, http://virtualization.softwareinsider.com, 27.04.2015.

[28] Jacob O'Gara, http://www.digitaltrends.com/computing/best-virtual-machine-apps-for-mac-linux-and-windows-pcs/, 27.04.2015.

[29] Brendan Gregg, http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html, 06.05.2015.

[30] Draios, Inc., https://sysdig.com/wp-content/uploads/2014/04/Blog2-pic3-1024x886.png, 18.06.2015.

[31] Bash Reference Manual, http://www.gnu.org/software/bash/manual/bashref.html, 19.06.2015.

[32] The Shadowserver Foundation, https://malwr.com, 27.06.2015.

[33] ldd(1) - Linux man page, http://linux.die.net/man/1/ldd, 27.06.2015.

[34] Google Inc., https://www.virustotal.com, 27.06.2015.

[35] GitHub, Inc., https://github.com, 28.06.2015.

[36] Cuckoo Foundation, http://docs.cuckoosandbox.org/en/latest/installation/host/requirements, 27.06.2015.

[37] Cuckoo Foundation, http://docs.cuckoosandbox.org/en/latest/installation/guest/network,27.06.2015.g

Sažetak

Naslov: Dodavanje podrške za izvršne datoteke Linuxa u Cuckoo okruženju za dinamičku analizu

Sažetak: Cuckoo sandbox je besplatan automatizirani sustav otvorenog kôda za dinamičku analizu koji zadnjih godina privlači sve više i više pažnje. Zasada podržava samo analizu datoteka koje se mogu izvršavati na Windowsima. Osnovna ideja ovog projekta je implementiranje funkcionalnosti koja će korisnicima omogućiti analizu Linux izvršnih datoteka. Glavni alati koji to omogućuju su virtualni stroj korišten kao sandbox i sysdig koji predstavlja alat za praćenje sistemskih poziva na Linuxu. Taj dodatak će rezultirati boljim i konkurentnijim sustavom za dinamičku analizu koji je svima dostupan.

Ključne riječi: Cuckoo Sandbox, dinamička analiza zloćudnih programa, Linux, sandbox, sistemski pozivi, sysdig

Abstract

Title: Adding support for Linux executables in Cuckoo sandbox

Summary: Cuckoo Sandbox is a free and open source automated malware analysis system that has been gaining more and more attention in recent years. For now, it supports only Windows binaries analysis. The major idea of this project is to implement a functionality which will make it possible for the users to analyze Linux binaries. Main tools that allow such analysis are virtual machine used for sandbox and sysdig representing tool which trace Linux system calls. This will result in better and more competitive dynamic analysis system available to everyone.

Keywords: Cuckoo Sandbox, dynamic malware analysis, Linux, sandbox, system calls, sysdig

Skraćenice

API	Application Programming Interface
CPU	Central Processing Unit
CPL	Combined Programming Language
DLL	Dynamic-Link Library
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protokol
JAR	Java ARchive
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
LDD	List Dynamic Dependencies
MAEC	Malware Attribute Enumeration and Characterization
PCAP	Packet Capture
PDF	Portable Document Format
PID	Process IDentification
QEMU	Quick EMUlator
RAM	Random Access Memory
REST	REpresentational State Transfer
RGB	Red Green Blue
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
XML	EXtensible Markup Language
TID	Thread Identification

Privitak

Upute za instaliranje programske podrške

Instalacija i podešavanje okruženja za dinamičku analizu zloćudnog programa može biti dugotrajan i problematičan proces. Važno je pravilno podesiti *sandbox* kako zloćudni program ne bi prouzrokovao štetu. Korišteni operacijski sustav za nadziranje sustava dinamičke analize je Linux Fedora 20. Najprije je potrebno instalirati program za verzioniranje *Git* [35].

```
sudo yum install git
```

Zatim je potrebno preuzeti Cuckoo Sandbox repozitorij naredbom:

```
git clone https://github.com/nidzo17/cuckoo.git
```

Zatim je potrebno instalirati sve potrebne programe i biblioteke. Najprije se instaliraju:

- python
- python-pip
- mongodb

Ostale neophodne biblioteke se mogu instalirati pokretanjem sljedeće naredbe iz Cuckoo direktorija [36]:

```
sudo pip install -r requirements.txt
```

Ovom naredbom su se instalirale sljedeće biblioteke:

- sqlalchemy
- dpkt
- python-magic
- jinja2
- pymongo
- bottle

- pefile
- django
- chardet
- nose

Za podršku MAEC izvješća potrebno je instalirati *cybox* inačicu 2.1.0.9 i *maec* inačicu 4.1.0.11 korištenjem *pip* programa za instaliranje

Kako bi se pratile mrežne aktivnosti koje obavlja zloćudni program tijekom izvršavanja, potrebno je pravilno podesiti i instalirati tcpdump.

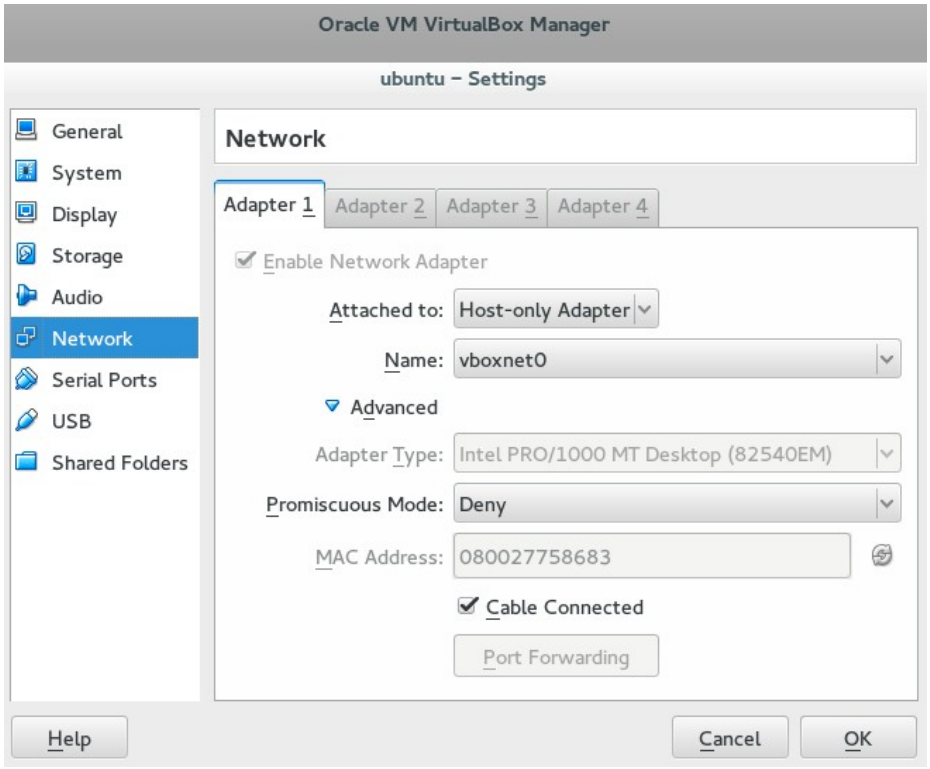
```
sudo yum install tcpdump
sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
```

Posljednja naredba omogućava Cuckoo Sandbox-u pokretanje tcpdumpa bez administratorskih ovlasti.

Kao što je opisano u radu, prilikom izrade i testiranja sustava za dinamičku analizu zloćudnih programa je korišten virtualni stroj kao sandbox. Potrebno je instalirati neki od virtualizacijskih programa a preporučuje se VritualBox. Sljedeće naredbe preuzimaju najnoviji VirtualBox i instaliraju biblioteke o kojima ovisi VirtualBox. Zatim se instalira VirtualBox i pokrene modul.

```
yum update
yum install binutils gcc make patch libgomp glibc-headers
glibc-devel kernel-headers kernel-devel dkms
yum install VirtualBox-4.3
/etc/init.d/vboxdrv setup
```

Sljedeći korak je preuzimanje instalacije i instaliranje nekog od Linux operacijskih sustava na VirtualBox. Nakon instalacije potrebno je podesiti mrežu u kojoj virtualni strojevi mogu komunicirati samo međusobno i s nadzirateljem. To se postiže dodavanjem sučelja *vboxnet0* u izborniku *File->Preferences->Newtork->Host-only Network*. Zatim u padajućem izborniku *Settings* treba izabrati *Network*. *Attached to* podesiti na Host-*only Adapter* i *Name* na *vboxnet0* kao što je prikazano na Slici 1.



Slika 1 Podešavanje mreže virtualnog stroja preko VirtualBox-a

Na nadziratelju, u Cuckoo repozitoriju, u datotekama *cuckoo.conf* i *virtualbox.conf* je potrebno postaviti IP adrese virtualnog stroja i nadziratelja. Zadana IP adresa virtualnog stroja je 192.168.56.101 dok je IP adresa nadziratelja 192.168.56.1.

Zatim je potrebno omogućiti spajanje virtualnog stroja na internet naredbama [37]:

```
iptables -A FORWARD -o eth0 -i vboxnet0 -s 192.168.56.0/24 -m
conntrack --ctstate NEW -j ACCEPT
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,Rmports
su funkELATED -j ACCEPT
iptables -A POSTROUTING -t nat -j MASQUERADE
sysctl -w net.ipv4.ip_forward=1
```

Sve što treba imati instalirano na virtualnom stroju su Python 2.7 i sysdig.

```
sudo yum install python
sudo yum install sysdig
```

Sysdig alat se mora pokretati s administratorskim ovlastima. Zbog toga je na kraj datoteke */etc/sudoers* potrebno dodati:

```
Cmnd_Alias SYSDIG = /usr/bin/sysdig
%sysdigers ALL=(ALL) NOPASSWD: SYSDIG
```

Posljednji korak podešavanja virtualnog stroja je preuzimanje datoteke *agent.py* iz Cuckoo repozitorija i pokretanje naredbom

```
python agent.py
```

Na kraju je potrebno snimiti *snapshot* kako bi se analiza uvijek pokretala na ovako podešenom virtualnom okruženju s pokrenutim *agentom*.

Upute za korištenje programske podrške

Kako bi se pokrenula analiza potrebno je pokrenuti Cuckoo Sandbox i poslati mu zadatak.

Cuckoo Sandbox pokrećemo naredbom:

```
python cuckoo.py
```

Ako je Cuckoo uspješno instaliran i podešen, na ekranu bi se trebala ispisati poruka:

```
  eeee e   e eeee e   e  eeeee eeeee
  8  8 8   8 8  8 8   8  8  88 8  88
  8e   8e  8 8e   8eee8e 8   8 8   8
  88   88  8 88   88   8 8   8 8   8
  88e8 88ee8 88e8 88   8 8eee8 8eee8

 Cuckoo Sandbox 1.3-dev
 www.cuckoosandbox.org
 Copyright (c) 2010-2015

 Checking for updates...
 Good! You have the latest version available.

2013-04-07 15:57:17,459 [lib.cuckoo.core.scheduler] INFO:
Using "virtualbox" machine manager
2013-04-07 15:57:17,861 [lib.cuckoo.core.scheduler] INFO:
Loaded 1 machine/s
2013-04-07 15:57:17,862 [lib.cuckoo.core.scheduler] INFO:
Waiting for analysis tasks...
```

U trećem poglavlju su nabrojani svi načini slanja zadatka Cuckoo Sandbox. Najlakši način za slanje zadatka je preko naredbenog retka uz pomoć *submit* alata. Iz Cuckoo repozitorija se pokrene naredba:

```
./utils/submit.py /putanja/do/datoteke
```

Cuckoo će zapisivati tijek izvođenja analize u naredbeni redak. Osim putanje do zlućudnog programa kao glavnog argumenta, *submit.py* može primiti desetke drugih argumenata od kojih je najkorišteniji *timeout* koji služi za ograničavanje trajanja analize. Osim *timeout* argumenta postoje i argumenti koji određuju prioritet, paket s kojim će se pokretati zlućudni program, platformu i virtualni stroj na kojem će se izvršiti analiza i sl.

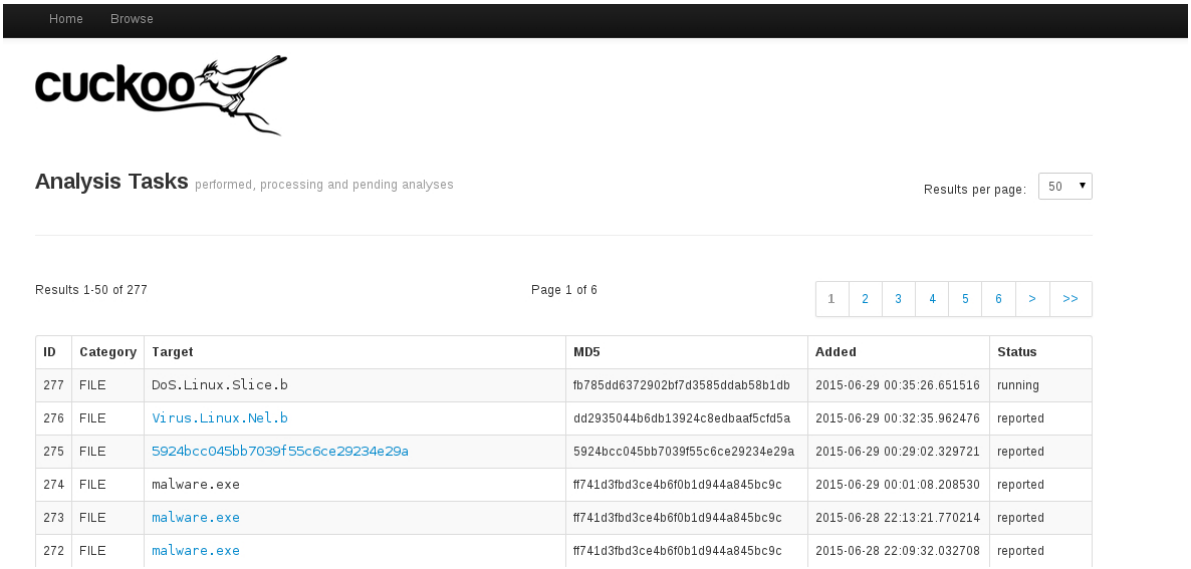
Primjerice, sljedećom naredbom se pokreće analiza s trajanjem od 60 sekundi na virtualnom stroju s nazivom *ubuntu*:

```
./utils/submit.py --timeout 60 /putanja/do/datoteke --machine
ubuntu
```

Drugi najkorišteniji način slanja zadatka Cuckoo Sandbox-u je preko internet preglednika. Najprije je potrebno pokrenuti naredbu:

```
./utils/web.py
```

Nakon toga se pokreće poslužitelj na lokalnom računalu pomoću kojeg je moguće slati zadatke ali i pregledavati rezultate postojećih analiza (*Slika 2*). Sučelju se pristupa preglednikom na adresi http://0.0.0.0:8080/



Slika 2 Pregledavanje postojećih analiza preko mrežnog sučelja