|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |

Fakultet Tehničkih nauka

Sigurnost i bezbednost

u Smart Grid

sistemima

Projektni zadatak 3 - Dokumentacija

Novi Sad, 2017

# **1. SISTEM**

U ovom odeljku se nalazi kratak opis projektnog zadatka, dizajn sistema, kao i arhitektura sistema.

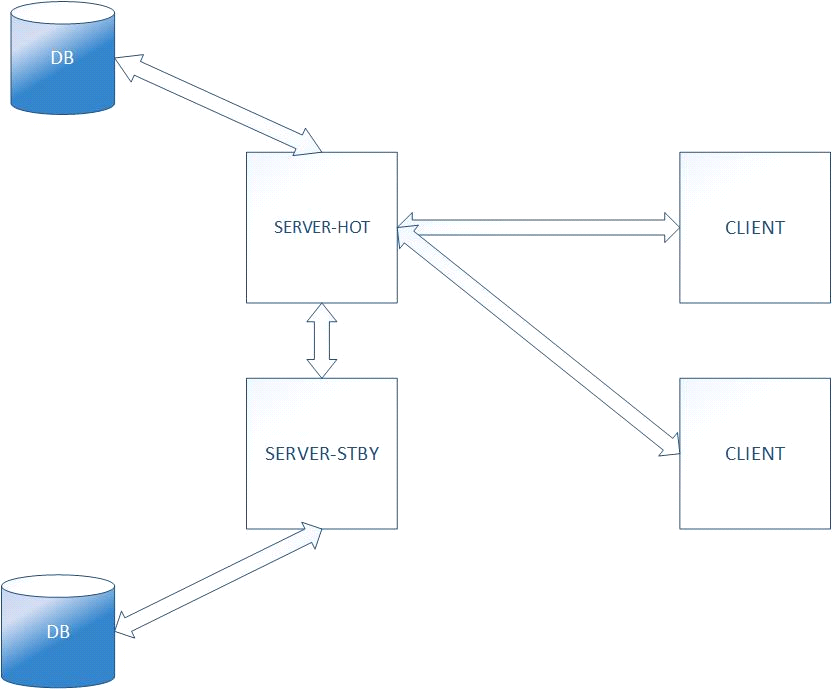
## 1.1 OPIS SISTEMA

Implementirati klijent – servis model koji simulira anti – malware.

* Anti – malware servis je komponenta kojoj klijenti pristupaju sa zahtevom za proveru potpisa određene aplikacije koja je izvršava. Autentifikacija između servera i klijenta se vrši pomoću sertifikata. Komunikacija se vrši preko TCP protokola.
* Klijentska aplikacija ima dve uloge :
* Praćenje aplikacija i procesa u izvršavanju aplikacija. Kada se aplikacija pokreće, klijentska aplikacija treba da preuzme digitalni potpis aplikacije koja se pokreće i da pošalje Anti – malware servisu koji treba da vrati odgovor da li je aplikacija koja se pokreće bezbedna ili je zaražena.
* Druga uloga je mogućnost prosleđivanja putanje do određenog fajla na disku koji klijentska aplikacija treba da proveri, da preuzme digitalni potpis i da pošalje Anti – malware servisu koji treba da vrati odgovor da li je aplikacija koja se pokreće bezbedna ili je zaražena.
* Prilikom slanja podataka od klijenta ka servisu podaci treba da budu kodirani 3DES algoritmom, tj. ne šalju se u obliku otvorenog teksta.
* Implementirati 3DES algoritam u CBC modu. Prilikom izrade zadatka voditi računa o mogućnostima paralelizacije operacija.
* Potpis aplikacije koja se proverava predstavlja heš vrednost, i kao takva heš vrednost se čuva u jedinstvenoj bazi podataka (MySQL, SQLite, Mongo...) kojoj pristupa servis.
* Sve akcije klijentske aplikacije treba da budu logovane u Windows Event Log-u, kao i odgovor anti – malware servisa o bezbednosti aplikacije koja je predmet analize.
* Dodatno, server treba da omogući Failover funkcionalnost. Potrebno je obezbediti redundantnu server komponentu koja u slučaju nedostupnosti primarnog servera može da preuzme njegovu ulogu :
* Održavanje integriteta podataka između primarne i backup komponente.
* Mogućnost klijenta da komunicira sa novom server komponentom nakon failovera.

## 1.2 ARHITEKTURA SISTEMA

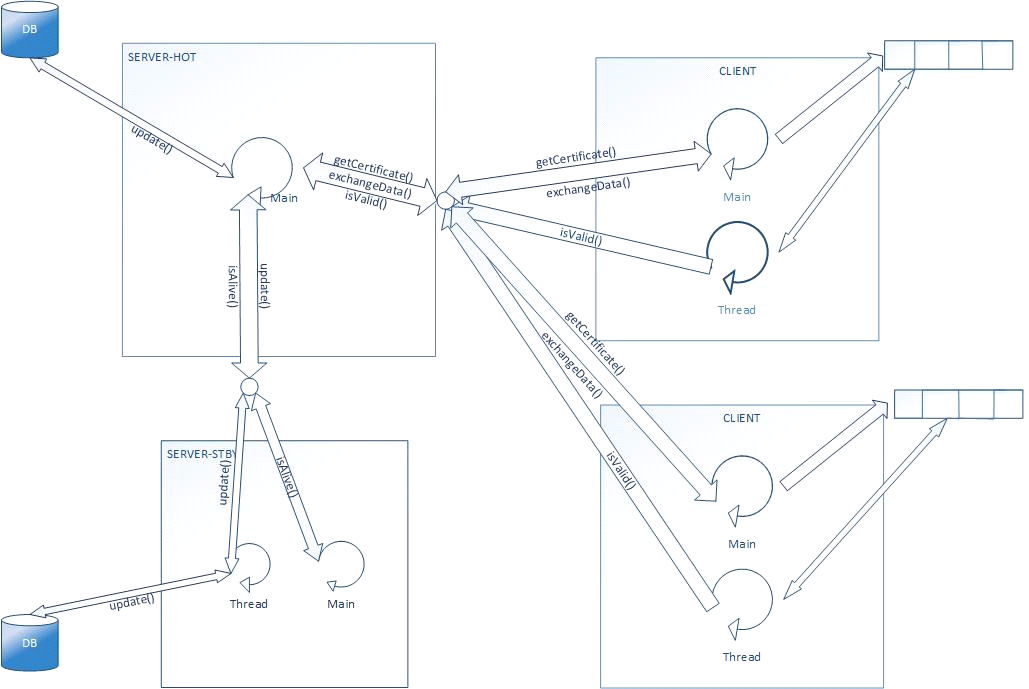
Arhitekturu sistema čine aktivni server (hot server), backup server (standby server), klijentske aplikacije, i baze podataka (po jedna baza za svaki server). U bazi podataka se nalazi tabela sa heševima aplikacija (procesa) koje nisu zaražene, tj. koje su bezbedne, kao i odgovarajuća imena aplikacija. Klijent može da unese putanju do određenog fajla na disku, ili da pokrene željeni proces na računaru, koji treba da se proveri da li je zaražen. U tom trenutku se proverava da li se heš vrednost aplikacije nalazi u white listi u bazi, tj. da li je aplikacija bezbedna. U zavisnosti od ishoda pretrage baze klijentu se javlja da li je aplikacija bezbedna ili nije. Na jedan server je moguće nakačiti proizvoljan broj klijenata. Naravno samo klijenti sa odgovarajućim sertifikatima mogu da se povežu sa servisom. Ukoliko iz nekog razloga aktivni server padne ili postane nedostupan, tada će njegovu ulogu da preuzme backup server, tj. učiniće se prevezivanje. Podržana je funkcionalnost održavanja integriteta podataka između aktivnog i backup servera. Na slici 1.1 je prikazana arhitektura sistema.



*Slika 1.1 Arhitektura sistema*

## 1.3 DIZAJN SISTEMA

Dizajn sistema sadrži detaljan model, tj. osnovne komponente sistema, tip komunikacije, kao i tok podataka između njih, što je prikazano na slici 1.2.



*Slika 1.2 Dizajn sistema*

# 2. STRUKTURE PODATAKA

U ovom poglavlju su prikazane strukture podataka koje su korišćene za implementaciju sistema.

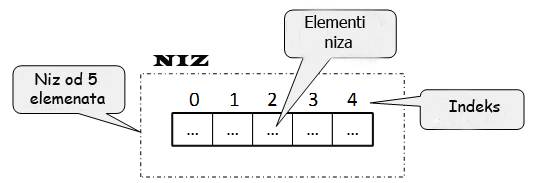
## 2.1 Dictionary

Dictionary je možda i najčešće korišćena kolekcija u C# koja predstavlja asocijativni kontejner. Asocijativna iz razloga što se prilikom skladištenja podacima dodeljuje ključ (key) koji služi za manipulisanje podacima u kolekciji. Dictionary važi za najbržu asocijativnu kolekciju jer u osnovi koristi HashTable strukturu. To znači da su vrednosti ključeva hash vrednosti, čime se znatno poboljšavaju performanse u radu sa ovom vrstom kolekcija. Razlika između HashTable i Dictionary kolekcije je u tome što je dictionary generički tip, što dicitonary čini type safe strkturom, odnosno nije moguće dodati slučajan tip elementa u kolekciju, a samim tip nije potrebno kastovanje podataka prilikom čitanja elemenata iz kolekcije. Vreme potrebno za dodavanje, uklanjanje i pretraga je relativno konstantno bez obzira na veličinu kolekcije.

Elementima dictionary-a se pristupa preko vrednosti ključa. Ukoliko nismo sigurni da li se ključ po kom pristupamo elementu dictionary-a zaista nalazi u dictionary-u, potrebno izvršiti odgovarajuću proveru pre pristupanja. Ta provera se najčešće izvršava koristeći metodu ContainsKey. Međutim, ukoliko postoji potreba za čestim pristupanjem elementima sa verovatnoćom da ključ ne postoji u dictionary-u, efikasnije je koristiti metodu *TryGetValue*. Ova metoda u pozadini proverava da li navedeni ključ postoji u kolekciji, a zatim vraća vrednosti elementa koji je dodeljen tom ključu. U slučaju da ključ ne postoji, biće vraćena default vrednost koja odgovara tipu elementa. Dakle, za pristupanje elementima preko ključa treba koristiti metodu *TryGetValue* kao mnogo pouzdaniji i efikasniji način pristupanja.

## 2.2 Nizovi

U rešavanju raznih problema javlja se potreba za postojanjem većeg broja podataka istog tipa koje predstavljaju jednu celinu. Zbog toga se u programskim jezicima uvodi pojam niza ili u opštem slučaju pojam polja. Ovo možemo predstaviti slikom.



Elementi niza su numerisani sa 0,1,2..,n-1.

Ovi brojevi se nazivaju indeksima elemenata niza. Broj elemenata u nizu predstavlja njegovu dužinu. Nizovi mogu biti različitih dimenzija. Najčešće se koriste jednodimenzionlani nizovi ili vektori i dvo-dimenzionalni nizovi ili matrice.

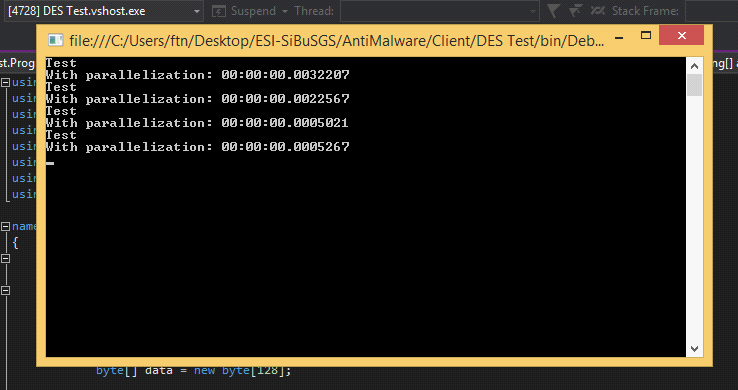
Niz predstavlja složeni tip podataka, sačinjen od nekolicine drugih podataka istog ili različitog tipa. Svaki podatak u nizu se naziva njegovim elementom, a svaki element ima svoj indeks, preko kojeg pristupamo tom elementu u nizu.

Nizovi u okviru našeg sistema se koriste u dosta slučajeva. Uglavnom se koriste za podatke, pre svega nizove bajtova podataka koji se enkriptuju i dekriptuju, kao i ključeve i inicijalni vektor koji su predstavljeni kao nizovi bajtova. Takođe nizovi se dosta koriste za implementaciju 3DES algoritma.

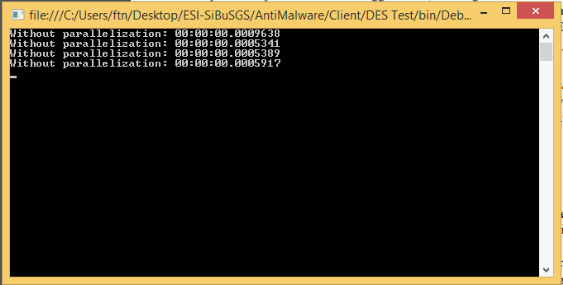
# 3. TESTIRANJE

Vršeno je više testiranja koja su opisana u daljem tekstu. Testovi pokrivaju sve pozitivne i negativne testne slučajeve (npr. uspešna i neuspešna autentifikacija).

Testirana je mogućnost ubrzanja programa pomoću paralelizacije. Paralelizacija je odrađena na dekripciji kod 3DES algoritma. Slika 3.1 prikazuje kojom brzinom se izvršava 3DES enkripcija i dekripcija sa paralelizovanom dekripcijom. Enkripcija i dekripcija su pozivani više puta. Na slici 3.2 nalazi se prikaz enkripcije i dekripcije bez paralelizacije. Na osnovu testova, zaključili smo da je algoritam bez paralelizacije znatno brži pri inicijalnom pokretanju, a posle su rezultati slični.

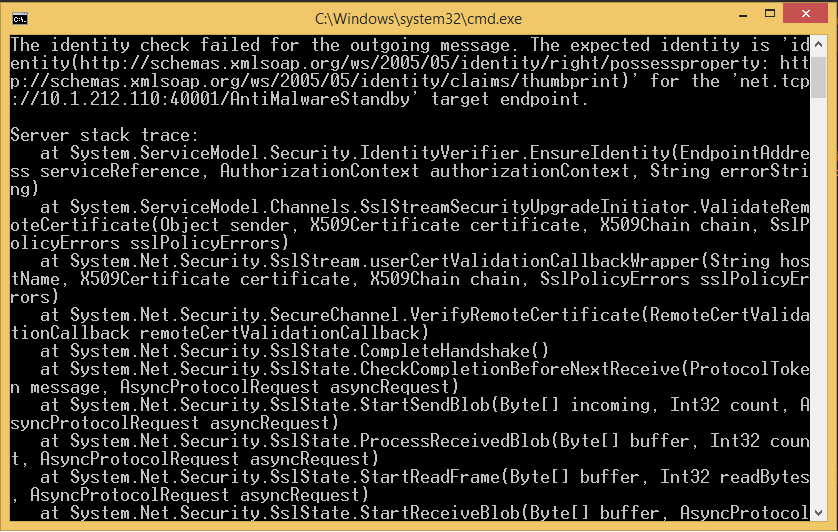


*Slika 3.1 Prikaz rada programa sa paralelizacijom*



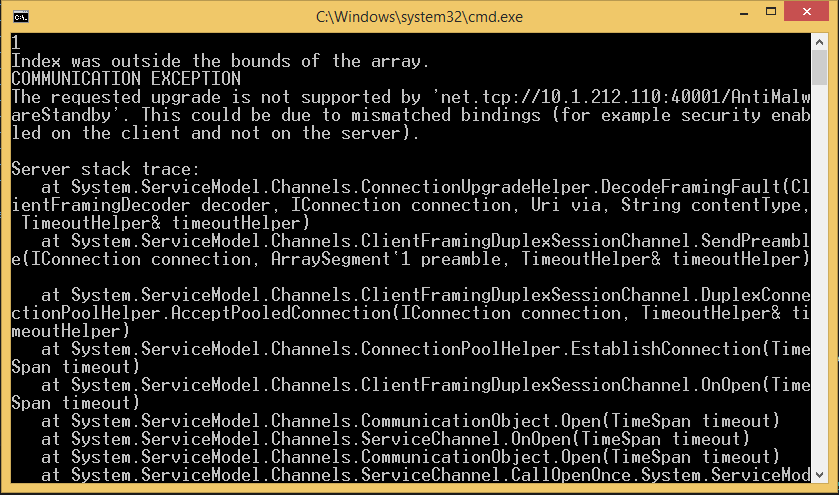
*Slika 3.2 Prikaz rada programa bez paralelizacije*

Ponašanje programa kada klijent poseduje nevalidan sertifikat i pokušava da koristi usluge sistema je prikazano na slici 3.3. Kao što se na slici može videti klijent ne uspeva da se poveže sa servisom, jer nema validan sertifikat za autentifikaciju.



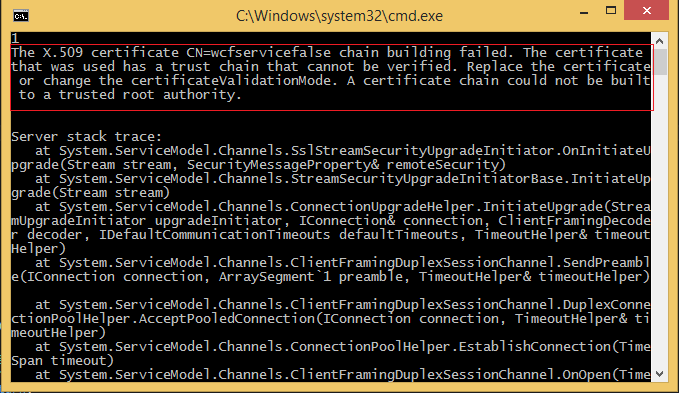
*Slika 3.3 Ponašanje programa sa nevalidnim klijentskim sertifikatom*

Na slici 3.4 nalazi se prikaz ponašanja programa ako klijent nema sertifikat, a pokušava da komunicira sa serverom. Kao što se može videti na slici u ovoj situaciji dolazi do communication exception-a, jer klijent ne poseduje nijedan sertifikat za autentifikaciju, a definisana je autentifikacija sertifikatima.



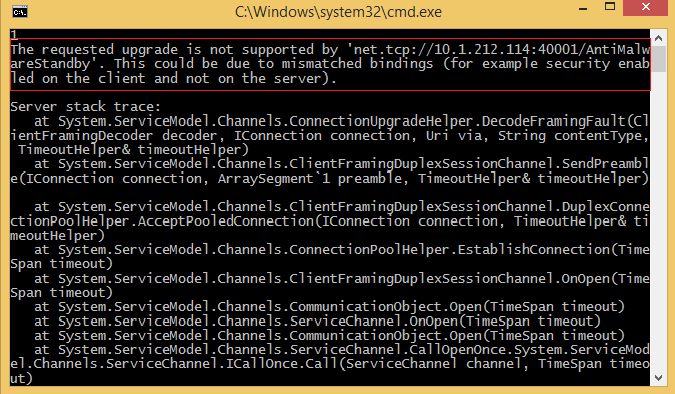
*Slika 3.4 Ponašanje programa kada klijent nema sertifikat*

Ponašanje programa kada server ima nevalidan sertifikat prikazano je na slici 3.5. U okviru poruke jasno se zaključuje da serverski sertifikat nije validan, tj. da sertifikat koristi trust chain koji se ne može proveriti, tako da dalje izvršavanje programa nije moguće bez validnog sertifikata.



*Slika 3.5 Ponašanje programa kada server ima nevalidan sertifikat*

Na slici 3.6 prikazano je kako se program ponaša kada server ne poseduje sertifikat. Dalje izvršavanje programa nije moguće usled nedostatka validnog serverskog sertifikata.



*Slika 3.6 Ponašanje programa kada server ne poseduje sertifikat*

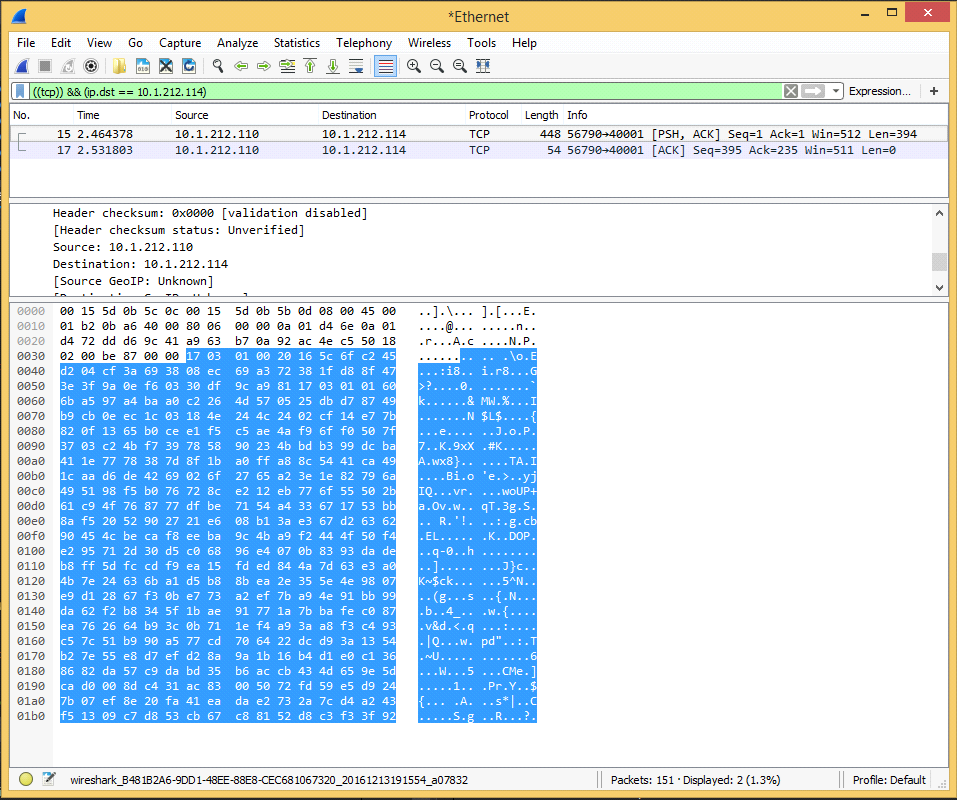
## 3.1 Testiranje ispravnosti sigurnosnih protokola kroz alat Wireshark

U okviru ovog poglavlja biće izvršeno testiranje ispravnosti sigurnosnih protokola putem alata *Wireshark*.

Wireshark je besplatan paket analizator, otvorenog koda. Koristi se za mrežno rešavanje problema, analizu, razvoj softvera, i komunikacionog protokola. Ovo je softver koji razume različite mrežne protokole. Može analizirati i prikazivati polja, zajedno sa njihovim značenjima kao što je navedeno od strane različitih mrežnih protokola.

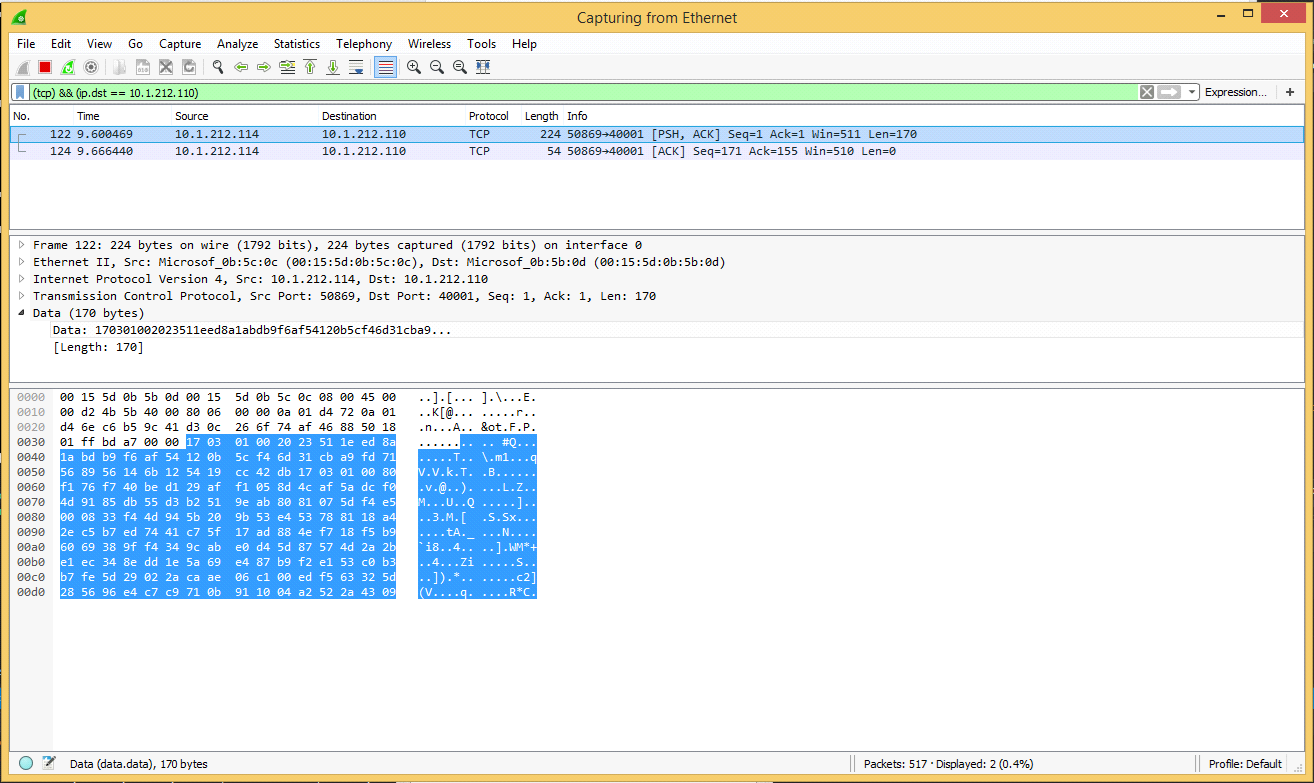
Da bi ovaj test bio uspešan potrebno je videti da li su podaci prilikom razmena poruka enkriptovani. Kao što se na narednim slikama može videti podaci prilikom razmena poruka su enkriptovani, tj. ne šalju se u čistom obliku (*plain text-u*). Dalje je potrebno pokazati da enkriptovane poruke klijentovog odgovora ka servisu, kao i poruke klijenta ka servisu koje nisu enkriptovane, nisu iste. To se može videti na slikama 3.7 i 3.8 ukoliko se uporede odgovori.

Slika 3.7 prikazuje klijentov zahtev ka servisu koji nije kriptovan.

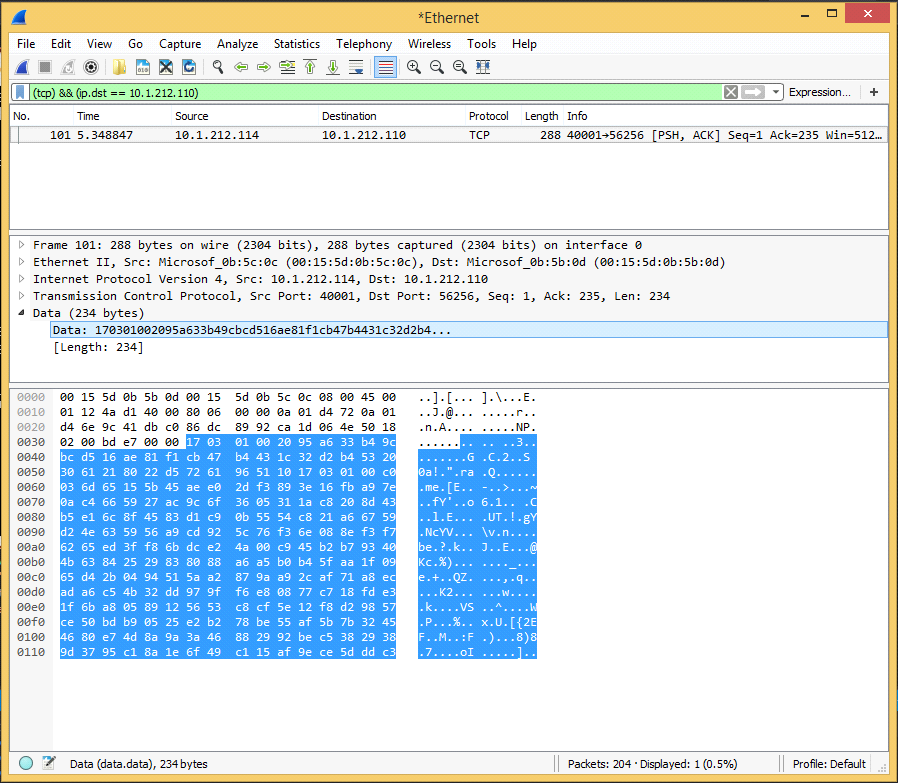


*Slika 3.7 Prikaz klijentovog odgovora ka servisu koji nije kriptovan*

Na slici 3.8 nalazi se prikaz kriptovanog klijentskog odgovora ka servisu.

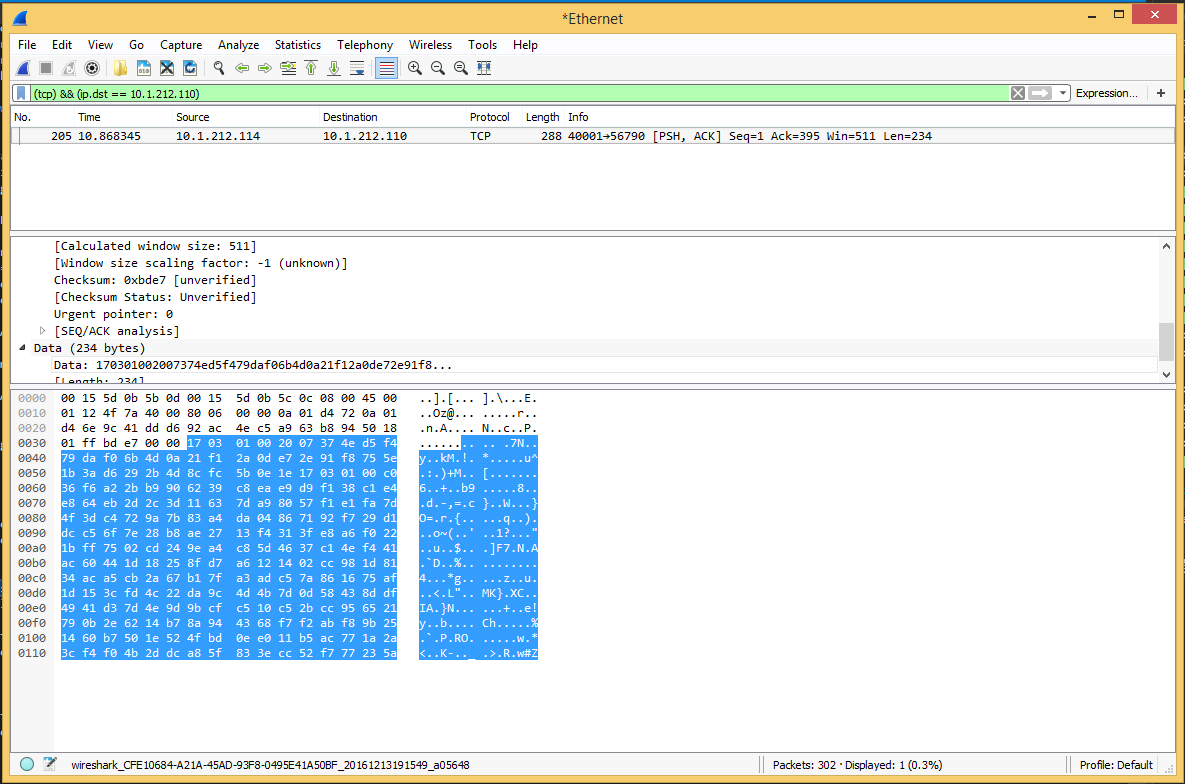


*Slika 3.8 Prikaz kriptovanog klijentskog odgovora ka servisu*

 Na slici 3.9 nalazi se prikaz kriptovanog odgovora servisa ka klijentu.

*Slika 3.9 Prikaz kriptovanog odgovora servisa ka klijentu*

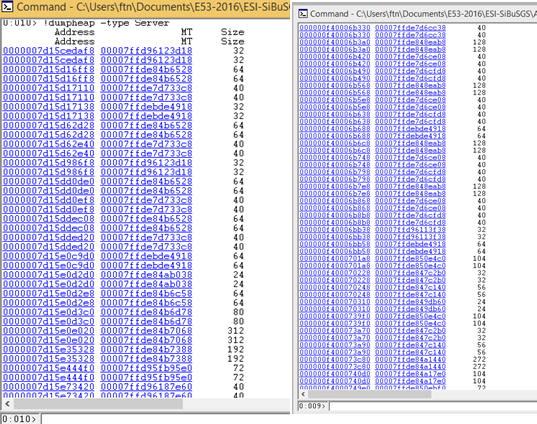
Na slici 3.10 nalazi se odgovor servisa klijentu bez TripleDES šifrovanja.

*Slika 3.10 Odgovor servisa klijentu bez TripleDES šifrovanja*

## 3.2 Testiranje curenja memorije putem alata WinDbg

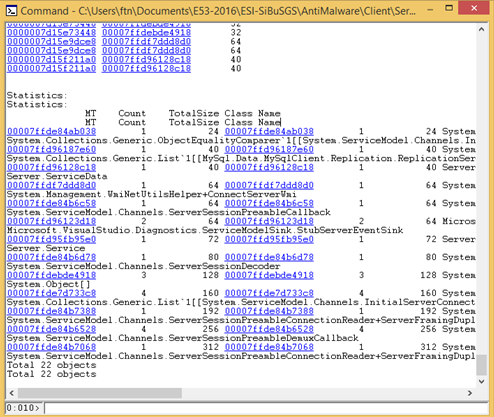
Debugging je tehnika za otkrivanje problema i grešaka (bugova) u programskom kodu koje se mogu ispoljavati stalno ili povremeno. Stavljanje breakpoint-a u izvornom kodu koristeći Visual Studio je jedan od načina za debugging, ali to nije uvek moguće, jer izvorni kod ne mora uvek biti dostupan. U tom slučaju, potrebno je koristiti debugging alate kako bi se analizirali različiti problemi kao što je pronalaženje uzroka rušenja aplikacije, problemi sa performansama (performance issues), curenje memorije (memory leak), i slično. Glavna prednost ovog alata je GUI komponenta koja znatno olakšava proces debugovanja.

Na slici 3.11 prikazan je dump heap kada se klijent konektuje na server, prvi deo



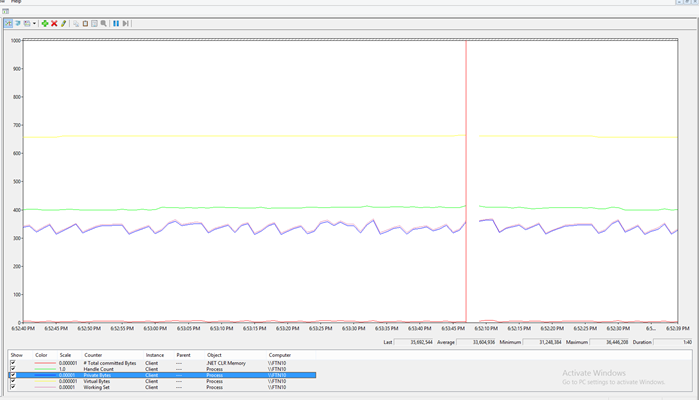
*Slika 3.11 Prikaz* dump heap-a kada se klijent konektuje na server

Slika 3.12 prikazuje dump heap kada se klijent konektuje drugi deo



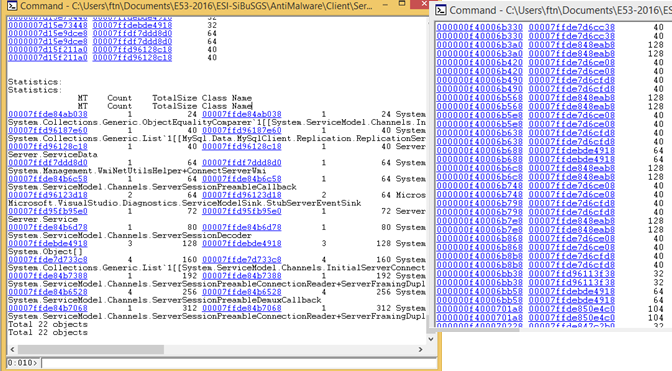
*Slika 3.12 Prikaz* dump heap-a kada se klijent konektuje

Slika 3.13 prikazuje kako se program ponaša kada je izložen stres testu. Na monitoru su prikazani sledeći podaci: *total commited bytes, handle count, private bytes, virtual bytes i working set*.



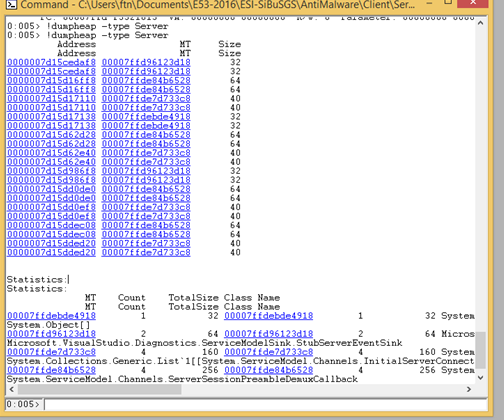
*Slika 3.13 Prikaz ponašanja programa kada je izložen stres testu*

Slika 3.14 prikazuje dump heap servera kada obradi nekoliko hiljada klijentskih zahteva



*Slika 3.14 Prikaz dump heap-a* servera kada obradi nekoliko hiljada klijentskih zahteva

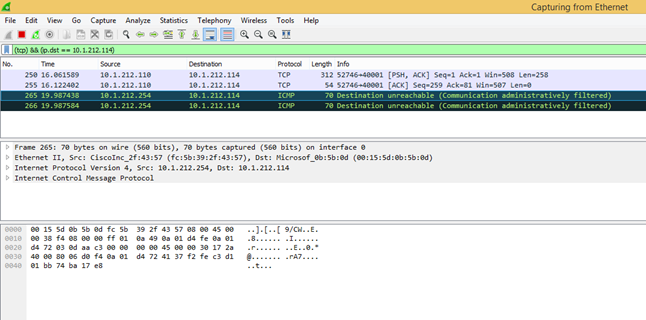
Slika 3.15 prikazuje dump heapa kada se server startuje



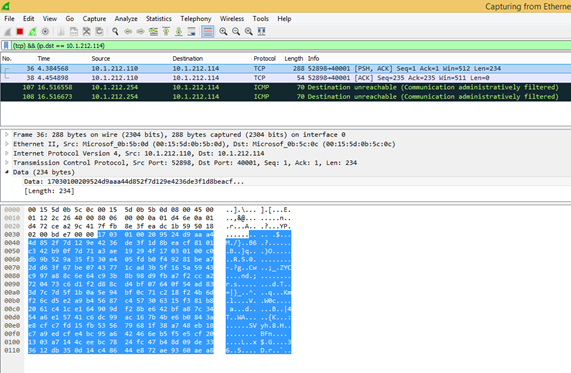
*Slika 3.15 Prikaz dump heapa kada se server startuje*

## 3.3 Testiranja vršena pomoću alata Cain and Abel

*Slika 3.16*



*Slika 3.17*



*Slika 3.18*