

Verifikacija softvera - Vežbe

Ivan Ristović

Ana Vulović

2022-10-10

Kolekcija materijala sa vežbi za kurs Verifikacija softvera na Matematičkom fakultetu, Univerziteta u Beogradu.

Teme i alati radeni na vežbama:

- Debugovanje koristeći alate za debugovanje i razvojna okruženja
 - `gdb` - debugovanje koda na niskom nivou
 - `QtCreator` - debugovanje C/C++ kodova
 - `Intellij IDEA` - debugovanje Java koda
 - `Visual Studio` - debugovanje C, C++ i C# kodova
- Testiranje jedinica koda
 - Pisanje testabilnog koda
 - `QtTest` , `Catch` (C, C++)
 - `JUnit` (Java)
 - `xUnit` , `NUnit` (C#)
- Praćenje pokrivenosti koda testovima
 - `lcov` (C, C++)
 - `JaCoCo` (Java)
 - `dotnet-coverage` (C#)
- "Mock" testiranje
 - Pisanje *mock* klasa od nule (C++)
 - `Moq` (C#)
 - *Mock* testiranje baza podataka
- Profajliranje
 - `Valgrind` (memcheck, cachegrind, callgrind, hellgrind, drd)
 - `perf`
- Statička analiza
 - `KLEE`
 - `CBMC`
 - `Clang` statički analizator
- Alati i jezici za formalnu verifikaciju softvera
 - `Dafny`

Sadržaj

1	Debagovanje	3
1.1	QtCreator Debugger	3
1.1.1	Buffer Overflow primer (QtCreator)	3
1.2	gdb	5
2	Instalacije	9
2.1	Alati za debugovanje i razvojna okruženja	9
2.1.1	QtCreator	9
2.1.2	gdb	9

1 Debagovanje

1.1 QtCreator Debugger

QtCreator dolazi sa debugger-om koji predstavlja interfejs između QtCreator-a i native debugger-a (gdb, CDB, LLDB, ...). Moguće je debugovati Qt aplikacije, ali i native C/C++ aplikacije, kačiti se na već pokrenute procese i formirati debug sesije, i mnogo toga.

Više informacija je moguće pronaći u Qt dokumentaciji.

1.1.1 Buffer Overflow primer (QtCreator)

Ukoliko debugujemo Qt aplikacije, dovoljno je pokrenuti aplikaciju u Debug modu. Pokrenuti QtCreator i otvoriti BufferOverflow.pro projekat. Prilikom otvaranja projekta, QtCreator će otvoriti konfiguracioni dijalog - konfigurisati projekat sa podrazumevanim podešavanjima.

Napomena: QtCreator podrazumevano uključuje `clang` statički analizator ukoliko je on dostupan. Iako nije neophodno, ukoliko želite da sami pronađete probleme u izvornom kodu a tek potom proverite svoje pronalaskе koristeći `clang`, isključite `clang` analizator u podešavanjima (`Edit -> Preferences -> Analyzer`).

Unutar QtCreator-a vidimo nekoliko pogleda (`Edit` , `Design` , `Debug` itd.). Trebalo bi da je `Edit` pogled već selektovan. Otvoriti `main.c` fajl i prodiskutovati o logici programa i njegovim slabim tačkama.

Unutar `Project` prozora možemo dodavati ili menjati postojeće konfiguracije za kompilaciju i pokretanje programa. To uključuje i dodavanje argumenata komandne linije (`Run` odeljak).

Pokrenuti debug sesiju. Isprobati debug akcije za kontrolisanje izvršavanja programa (`Debug` meni u glavnom meniju):

- `Interrupt`
- `Continue`
- `Step over`
- `Step into`
- `Step out`
- `Set/Remove breakpoint`

Možemo testirati ponašanje programa tako što prvo unesemo ispravnu lozinku `MyPassword` a zatim i proizvoljnu neispravnu lozinku (npr. `SomePassword`). Program se naizgled ispravno ponaša ali to ne znači da je u potpunosti ispravan.

Posmatrajući definicije promenljivih `password` i `ok`, možemo zaključiti da će se one na steku naći jedna do druge. Proverimo da li je zaista tako: desnim klikom na promenljive u prozoru za prikaz promenljivih na steku selektovati opciju `Open Memory Editor -> ... at object address ...`. Nastaviti izvršavanje i primetiti inicijalizaciju memorije za promenljivu `ok`. Pošto je promenljiva `password` neposredno pre promenljive `ok` u memoriji, i pošto se sadržaj promenljive `password` unosi sa standardnog ulaza, ukoliko uspemo da pređemo granice promenljive `password` onda možemo upisati proizvoljnu vrednost u promenljivu `ok` (pa i `"yes"`)!

Posmatrajmo isečak koda:

```
scanf ("%s", password);
```

`scanf` nam ne garantuje da će učitati najviše 16 karaktera (koliko smo rezervisali za promenljivu `password`). Testirajmo ponašanje za niske dužine veće od 16 karaktera, npr. `WillThisPassNow?yes`. Vidimo da smo dobili privilegije iako lozinka nije ispravna.

Postavimo uslovni *breakpoint* nakon poziva `scanf`, da se program zaustavi ukoliko se promeni vrednost promenljive `ok`. Desnim klikom na `Breakpoint` i selekcijom opcije `Edit Breakpoint` možemo definisati uslov `ok != "no"` u `Condition` polju. Možemo takođe pratiti inicijalizaciju promenljive `ok` u `Memory Editor` -u. Nakon što se vrednost promeni, možemo je ručno vratiti na `"no"`.

Tok izvršavanja možemo pratiti i preko prikaza steka, odakle možemo videti redosled pozivanja funkcija. Konkretnu liniju u kodu koja je sledeća za izvršavanje, QT obeležava žutom strelicom levo od koda. Ukoliko se dogodi da smo mnogo napredovali i želimo da se pomerimo na neku ranije naredbu, dovoljno je da strelicu prevučemo na naredbu koja nam je potrebna, bez potrebe za ponovnim pokretanjem programa. Za inspekciju asemblerskog koda možemo otvoriti *Disassembler* prozor.

Ostaje pitanje - šta naš program radi ukoliko je lozinka toliko dugačka da prevazilazi veličinu steka (npr. `ThisPasswordSimplyExceedsMaximumLengthAvailableOnStack`)? Dobićemo poruku `*** stack smashing detected ***`. To je posledica opcije `gcc -a -fstack-protector` koja generiše zaštitnu promenljivu koja je se dodeljuje osetljivim funkcijama. Osetljivim se smatraju one funkcije koje koriste dinamičku alokaciju ili imaju bafere veće od 8B. Zaštitna promenljiva se inicijalizuje pri ulasku u funkciju i proverava se na izlasku. Ukoliko provera ne bude tačna, štampa se prikazana poruka i program prekida sa izvršavanjem. Ukoliko želimo da učinom kod još ranjivijim na ovakve napade, možemo pomenutu zaštitu isključiti opcijom `-fno-stack-protector`. U slučaju Qt projekta potrebno je dodati ovu opciju u `QMAKE_CFLAGS` promenljivu. U definiciji projekta (datoteci ekstenzije `.pro`) uneti

sledeći red a zatim pozvati `qmake` i izvršiti ponovno prevoženje projekta:

```
QMAKE_CFLAGS += -fno-stack-protector
```

Da bismo sprečili napade prekoračenjem bafera savet je da se primenjuju dobre programerske prakse i da se:

- preoverava upravljanje memorijom tokom programa koristeći neki od alata poput `valgrind` `memcheck`
- upotrebljava `fgets()` funkcije umesto `gets()` ili `scanf()` koje ne vrše provere granica promenljivih.
- upotrebljava `strncmp()` umesto `strcmp()` , `strncpy()` umesto `strcpy()` , itd.

1.1.1.1 Debugovanje spoljašnje aplikacije kroz QtCreator Ukoliko pokrenemo prethodni primer izvan QtCretor-a, to ne znači da ne možemo debugovati taj program. Štaviše, moguće je QtCreator debugger zakačiti za bilo koji program koji se izvršava ili će se tek izvršiti. Iz menija `Debug` biramo `Start Debugging` . Na raspolaganju su nam opcije:

- *Attach debugger to started application* - da bismo debugovali aplikaciju koja je već pokrenuta.
- *Attach debugger to unstarted application* - da bismo debugovali aplikaciju koja je će biti pokrenuta. Zadaje se putanja do izvršne verzije programa. Debugger će se aktivirati kada aplikacija bude pokrenuta.
- *Start and Debug External Application* - slično kao prethodno samo što će se program pokrenuti odmah. Možemo čekirati opciju da se doda tačka prekida na početak `main` funkcije.

Napomena: Ako QtCreator ne uspe da se poveže na proces, moguće je da treba da se dozvoli povezivanje na procese tako što se upiše `0` u fajl `/proc/sys/kernel/yama/ptrace_scope` .

Da bismo imali informacije o linijama izvornog koda koji odgovara programu koji se debuguje, potrebno je da taj program bude preveden sa uključenim debug informacijama (za `gcc` to je opcija `-g`). Inače, debugger će nam prikazivati memoriju kojoj pristupa program.

1.2 gdb

GNU Debugger (gdb) je debugger koji se može koristiti za debugovanje (najčešće) C/C++ programa. Preko gdb je moguće pokrenuti program sa proizvoljnim argumentima komandne linije, posmatrati stanje promenljivih ili registara procesora, pratiti izvršavanje kroz naredbe originalnog ili asembliranog koda, postavljanje bezuslovnih ili uslovnih tačaka prekida i sl. Više o gdb se može pročitati na ovom linku.

Koristeći gdb možemo učitati program iz prethodnog primera. U slučaju da želimo da ručno prevedemo program, neophodno je da se postaramo da je prosleđen `-g` flag `gcc` kompilatoru:

```
$ gcc main.c -g -o BufferOverflow
```

Pošto je QtCreator već preveo ovaj program i u `build-*` direktorijum ostavio izvršivi fajl, možemo ga direktno učitati u gdb:

```
$ gdb BufferOverflow
...
Reading symbols from BufferOverflow...
(gdb)
```

Alternativno, možemo program učitati u već pokrenuti gdb proces:

```
$ gdb
...
(gdb) file BufferOverflow
Reading symbols from BufferOverflow...
(gdb)
```

U slučaju da debug simboli nisu prisutni, možemo u definiciji projekta dodati odgovarajući flag (pod `QMAKE_CFLAGS`).

Spisak komandi koje gdb pruža možemo dobiti komandom `help` . Za sve ove naredbe postoje i skraćene varijante - npr. za `running` možemo kucati `run` ili samo `r` .

```
(gdb) help
List of classes of commands:

aliases -- User-defined aliases of other commands.
breakpoints -- Making program stop at certain points.
data -- Examining data.
files -- Specifying and examining files.
internals -- Maintenance commands.
obscure -- Obscure features.
running -- Running the program.
stack -- Examining the stack.
status -- Status inquiries.
support -- Support facilities.
text-user-interface -- TUI is the GDB text based interface.
tracepoints -- Tracing of program execution without stopping the program.
user-defined -- User-defined commands.
```

Tačke prekida možemo postaviti komandom `breakpoints` , aktiviramo/deaktiviramo

komandama `enable` / `disable` , a brišemo komandom `delete` . Komandom `continue` nastavljamo rad programa do naredne tačke prekida. Tačke prekida mogu biti linije, npr. `main.c:15` ali i funkcije, npr. `main` ili `main.c:grant_privilege` . Postavimo tačku prekida na funkciju `grant_privilege` , kontrolišimo izvršavanje programa naredbama `step` (nalik na *step into*) i `next` (nalik na *step over*):

```
(gdb) b main.c:grant_privilege
Breakpoint 1 at 0x1171: file ../01_buffer_overflow/main.c, line 5.
(gdb) r
Starting program: BufferOverflow
Breakpoint 1, grant_privilege () at ../01_buffer_overflow/main.c:5
5      {
(gdb) s
7          char ok[16] = "no";
(gdb) n
9          printf("\n Enter the password : \n");
(gdb) n

Enter the password :
10          scanf("%s", password);
(gdb) n
MyPassword12      if (strcmp(password, "MyPassword")) {
(gdb) info locals
password = "MyPassword\000\367\377\177\000"
ok = "no", '\000' <repeats 13 times>
(gdb) n
15          printf("\n Correct Password \n");
(gdb) n

Correct Password
16          strcpy(ok, "yes");
(gdb) n
19          if (strcmp(ok, "yes") == 0) {
(gdb) n
23          printf("\n Root privileges given to the user \n\n");
(gdb) n

Root privileges given to the user

25      }
(gdb) finish
Run till exit from
#0  grant_privilege () at ../01_buffer_overflow/main.c:25
```

```
main () at ../01_buffer_overflow/main.c:43
43          return 0;
```

Komandom `info` možemo dobiti informacije o lokalnim promenljivim, registrima itd. (npr. `info locals` tj. `info registers`).

Alternativno, moguće je iskoristiti drugačiji korisnički interfejs, komandom `tui` npr. `tui enable` . Tako dobijamo pogled na izvorni kod na jednoj polovini korisničkog interfejsa. Komandom `tui reg` možemo dobiti i prikaz registara, npr. `tui reg all` .

Koristeći gdb možemo dodati i uslovne tačke prekida, npr. `b grant_privilege:12 if ok != "no"` . Komandom `info breakpoints` možemo videti spisak tačaka prekida.

2 Instalacije

2.1 Alati za debugovanje i razvojna okruženja

2.1.1 QtCreator

Instalirati QtCreator sa zvanične stranice. Alternativno, moguće je i instalirati ceo Qt radni okvir koji uključuje i QtCreator.

Za neke Linux distribucije je dostupan paket `qt<VERZIJA>-creator` .

2.1.2 gdb

Za većinu Linux distribucija je dostupan paket `gdb` . `gdb` je za neke distribucije deo paketa za razvoj (npr. `build-essential` za Ubuntu).