BAB IV IMPLEMENTASI DAN PENGUJIAN

IV.1 Implementasi Sistem

Tahap Implementasi sistem adalah tahap penciptaan modul atau *plugin DynamoRIO*, tahap lanjutan dari kegiatan perancangan sistem. Tahap ini merupakan tahap dimana sistem siap untuk dijalankan.

IV.1.1. Lingkungan Implementasi Sistem

Spesifikasi *environment* yang digunakan dalam pengimplementasian modul *DynamoRIO* ini adalah sebagai berikut:

a. BASH(Bourne Again SHell) versi 4.5
b. LLVM(Low Level Virtual Machine) versi 3.2
c. DynamoRIO versi 6.0
d. GNOME Terminal Emulator versi 3.6.2
e. Linux Kernel versi 3.13
f. GNOME Desktop versi 3.18

IV.1.2. Implementasi Pembangunan Dynamic Binary Instrumentation

Implementasi pembangunan *DynamoRIO plugin* dibagi kedalam 3 tahap, diantaranya adalah sebagai berikut :

A. Pembangunan *DynamoRIO*
B. Pembangunan dukungan LLVM
C. Pembangunan *DynamoRIO plugin*

A. Pembangunan *DynamoRIO*

Pada tahap implementasi ini *DynamoRIO* dikompilasi dengan *compiler* yaitu GCC(GNU Compiler). Alat lainnya yang harus terlebih dahulu dipasang adalah *binutils, perl* dan *cmake.*Adapun pembangunannya adalah sebagai berikut:

Tabel IV.1 Pembangunan DynamoRIO

No	Perintah
1	git clone https://github.com/DynamoRIO/dynamorio.git
2	cd dynamorio
3	mkdir build
4	cd build
5	cmake -DDR_EXT_DRMGR_STATIC=ON -DDR_EXT_DRSYMS_STATIC=ON \ -DDR_EXT_DRUTIL_STATIC=ON -DDR_EXT_DRWRAP_STATIC=ON \ -DDR_EXT_DRX_STATIC=ON ..
6	make -j2

B. Pembangunan Dukungan LLVM

Pembangunan LLVM ini bertujuan untuk membuat keluaran *intermediate representation.* Adapun paket dukungan yang harus dikompilasi adalah sebagai berikut:

Tabel IV.2 Pembangunan Dukungan LLVM

No	Perintah
1	svn co http://llvm.org/svn/llvm-project/llvm/branches/release_32/ llvm
2	cd llvm/tools
3	svn co http://llvm.org/svn/llvm-project/cfe/branches/release_32/ clang
4	cd ../projects
5	svn co http://llvm.org/svn/llvm-project/compiler-rt/branches/release_32/ compiler-rt
6	svn co http://llvm.org/svn/llvm-project/poolalloc/branches/release_32/ poolalloc
7	svn co http://llvm.org/svn/llvm-project/safecode/branches/release_32/ safecode
8	cd ..
9	mkdir build
10	cd build
11	cmake -DDynamoRIO_DIR=dynamorio/exports/cmake ..
12	make -j2

C. Pembangunan *DynamoRIO Plugin*

Modul yang dibangun dapat disalin kedalam *directory DynamoRIO* maupun tidak dengan kondisi saat pemanggilan ditetapkan lokasi *shared library* atau pustaka dimana *plugin* ini tersimpan. Adapun tahap kompilasinya sebagai berikut:

Tabel IV.3 Pembangunan DynamoRIO Plugin

No	Perintah
1	cd dynamorio
2	mkdir dynsmc
3	cd dynsmc
4	LDFLAGS="-Wl,-O1,-L/lib64,-L/usr/lib64" CFLAGS="-O2 -fPIC -m64" ./configure
5	make -j2

IV.1.3. Implementasi Pembuatan Random Test Case Generation

Pembuatan *Random Test Case Generation* yang mewakili setiap kategori *Self-Modifying Code* diman berisi biner acak valid maupun tidak valid *executable* bertujuan untuk memenuhi kualitas program setelah dilakukan pengujian *manual unit testing.* Adapun pembangunannya menggunakan *bash scripting* sebagai berikut:

```
#!/bin/bash

# Lokasi test case berada di folder test yang berisi biner program
# Self-Modifying Code maupun bukan dan beberapa random biner(untuk
# mencapai error atau false positive)
head -c 1M </dev/urandom > test/acak
list=(`ls test`)
for filename in ${list[@]}
do
   dynamorio/bin64/drrun -c dynsmc/libsmc.so -- ${filename}
done
```

IV.2 Pengujian Sistem

Selanjutnya adalah tahap pengujian sistem secara *manual* setiap unit satu per satu. Tahap ini bertujuan untuk menemukan kesalahan-kesalahan ataupun kekurangan-kekurangan pada *plugin* atau modul *DynamoRIO* yang dibangun.

Pengujian ini bermaksud untuk mengetahui apakah program sudah sesuai dengan tujuan perancangan aplikasi atau belum.

IV.2.1.1 Rencana Pengujian

Rencana pengujian yang akan dilakukan pada modul *DynamoRIO* ini selengkapnya dapat dilihat pada tabel IV.4 dibawah ini :

Tabel IV.4 Rencana Pengujian DynamoRIO Plugin

Item Uji	Detail Pengujian	Komponen Inti
Memindai *file*	Memilih *file* dalam hal ini yakni biner *executable*	*readdir, fopen*
Manajemen memori	Membebani program dengan *buffer* yang besar (*buffer overflow)* dan kerentanan akuisisi kembali memori yang sudah dihancurkan (*use-after-free)*	*malloc, realloc*
Nilai *environment*	Memberi nilai opsi pada *environment*	*setenv, getenv, putenv*
Pesan Keluaran	Menampilkan hasil keluaran analisis	*strerror, stdout*
Sinyal Interupsi	Terminasi program saat berjalan (Ctrl-C)	*SIGINT*

IV.2.1.2 Perencanaan Unit Testing

Unit testing menggunakan framework GTest (*Google Test*) dimana dapat ditemukan pada *plugin* yang dibangun dengan nama folder *testing.* Adapun file yang berisi dalam direktori tersebut berupa sebagai berikut:

1. Main

   Deklarasi pemanggilan *Google Test* dengan pengujian file yang berada di dalam direktori *testing*

```
#include "gtest/gtest.h"
      int main(int argc, char **argv) {
        testing::InitGoogleTest(&argc, argv);
        return RUN_ALL_TESTS();
}
```

2. Loadable

   Loadable merupakan file yang memeriksa setiap fungsi API yang digunakan dengan *Template* testing dari *Google Test Framework*. Adapun diantaranya melakukan cek sebagai berikut:

   1. Membuka file biner

      Pengujian file tidak eksis dengan melakukan *assertion* atau membandingkan data normal yakni file eksis

```
TEST(DYNSMC, gagalBukaFile) {
       const char *path = "/target_program_tidak_eksis";
       FILE *f = fopen(path, "r");
       ASSERT_TRUE(f == NULL);
}
```

   2. Alokasi memori

      Pengujian akuisisi memori untuk seluruh target program

```
TEST(DYNSMC, Malloc) {
        S4 *x = (int*)Ident(malloc(sizeof(S4)));
        EKSPEKTASI(*x);
        free(x);
}
```

3. Stack Stress

   Pengujian membebani stack dengan *buffer* yang besar

```
TEST(DYNSMC, StackTest) {
         StackTestFunc();
      }
NOINLINE void StackStressFunc() {
  int foo[10000];
  break_optimization(foo);
}
```

4. Pesan *error*

   Pengujian pesan kesalahan

```
TEST(DYNSMC, drsym_error_t) {
        char *buf = drsym_error_t(EINVAL);
        EXPECT_NOT_POISONED(strlen(buf));
        buf = drsym_error_t(123456);
        EXPECT_NOT_POISONED(strlen(buf));
}
```

5. File Descriptor

   Pengujian *file descriptor* mengenai keadaan *system call*

```
TEST(DYNSMC, pipe) {
        int* pipefd = new int[2];
        int res = pipe(pipefd);
        ASSERT_EQ(0, res);
        EXPECT_NOT_POISONED(pipefd[0]);
        EXPECT_NOT_POISONED(pipefd[1]);
        close(pipefd[0]);
        close(pipefd[1]);
}
```

6. Current Working Directory

   Pengujian tentang lokasi *path* yang sedang digunakan apakah eksis atau tidak berkenaan kemungkinan direktori sudah dihapus

```
TEST(DYNSMC, getcwd) {
  char path[PATH_MAX + 1];
  char* res = getcwd(path, sizeof(path));
  ASSERT_TRUE(res != NULL);
  EXPECT_NOT_POISONED(path[0]);
}
```

7. Shared Memory

Pengujian *shared memory* dari pemanggilan proses

```
TEST(DYNSMC, shmat) {
  void *p = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                 MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
  ASSERT_NE(MAP_FAILED, p);
  ((char *)p)[10] = *GetPoisoned<U1>();
  ((char *)p)[4095] = *GetPoisoned<U1>();
  int res = munmap(p, 4096);
  ASSERT_EQ(0, res);
  int   id   =   shmget(IPC_PRIVATE,   4096,   0644   |
IPC_CREAT);
  ASSERT_GT(id, -1);
  void *q = shmat(id, p, 0);
  ASSERT_EQ(p, q);
  EXPECT_NOT_POISONED(((char *)q)[0]);
  EXPECT_NOT_POISONED(((char *)q)[10]);
  EXPECT_NOT_POISONED(((char *)q)[4095]);
  res = shmdt(q);
  ASSERT_EQ(0, res);
  res = shmctl(id, IPC_RMID, 0);
  ASSERT_GT(res, -1);
}
```

8. Pembacaan Direktori

   Pengujian baca direktori dilakukan untuk memeriksa apakah program dapat membaca direktori atau tidak

```
TEST(DYNSMC, readdir) {
  DIR *dir = opendir(".");
  struct dirent *d = readdir(dir);
  ASSERT_TRUE(d != NULL);
  EXPECT_NOT_POISONED(d->d_name[0]);
  closedir(dir);
}
```

9. Canonicalized Path

   Pengujian absolut *pathname* merupakan pengecekan apakah target program adalah *symbolic links* atau tidak

```
TEST(DYNSMC, realpath) {
  const char* relpath = ".";
  char path[PATH_MAX + 1];
  char* res = realpath(relpath, path);
  ASSERT_TRUE(res != NULL);
  EXPECT_NOT_POISONED(path[0]);
}
```

10. Environment Variable

    Pengujian untuk mengganti dan mengubah nilai *environ* pada *shell*

```
TEST(DYNSMC, putenv) {
        char s[] = "AAA=BBB";
        putenv(s);
        for (char **envp = environ; *envp; ++envp)
      {
          EXPECT_NOT_POISONED(*envp);
          EXPECT_NOT_POISONED(*envp[0]);
        }
}
```

11. Penyalinan Memori

Pengujian fungsi salin memori pada dari *source* ke *destination*

```
TEST(DYNSMC, memcpy) {
        char* x = new char[2];
        char* y = new char[2];
        x[0] = 1;
        x[1] = *GetPoisoned<char>();
        memcpy(y, x, 2);
        EXPECT_NOT_POISONED(y[0]);
        EXPECT_POISONED(y[1]);
}
```

12. Penyalinan String

Pengujian fungsi salin *string* dari *source* ke *destination*

```
TEST(DYNSMC, strcpy) {
        char* x = new char[3];
        char* y = new char[3];
        x[0] = 'a';
        x[1] = *GetPoisoned<char>(1, 1);
        x[2] = 0;
        strcpy(y, x);  // NOLINT
        EXPECT_NOT_POISONED(y[0]);
        EXPECT_POISONED(y[1]);
        EXPECT_NOT_POISONED(y[2]);
}
```

13. Cetak Keluaran

Pengujian cetak keluaran ke *stdout* maupun *stderr*

```
TEST(MemorySanitizer, dr_printf) {
        char buff[10];
        break_optimization(buff);
        EXPECT_POISONED(buff[0]);
        int res = dr_printf(buff, "%d", 1234567);
        ASSERT_EQ(res, 7);
        ASSERT_EQ(buff[0], '1');
        ASSERT_EQ(buff[1], '2');
        ASSERT_EQ(buff[2], '3');
        ASSERT_EQ(buff[6], '7');
        ASSERT_EQ(buff[7], 0);
        EXPECT_POISONED(buff[8]);
}
```

14. Timestamp

    Pengujian *timestamp* dilakukan untuk memperoleh *interval* waktu eksekusi

```
TEST(DYNSMC, time) {
        time_t t;
        EXPECT_POISONED(t);
        time_t t2 = time(&t);
        ASSERT_NE(t2, (time_t)-1);
        EXPECT_NOT_POISONED(t);
}
```

15. Penggunan Resource

    Pengujian penggunaan kesuluruhan *resource* program

```
TEST(DYNSMC, getrusage) {
        struct rusage usage;
        __dyn_poison(&usage, sizeof(usage));
        int    result    =    getrusage(RUSAGE_SELF,
      &usage);
        ASSERT_EQ(result, 0);
        EXPECT_NOT_POISONED(usage.ru_utime.tv_sec);
```

```
      EXPECT_NOT_POISONED(usage.ru_utime.tv_usec);
        EXPECT_NOT_POISONED(usage.ru_stime.tv_sec);

      EXPECT_NOT_POISONED(usage.ru_stime.tv_usec);
        EXPECT_NOT_POISONED(usage.ru_maxrss);
        EXPECT_NOT_POISONED(usage.ru_minflt);
        EXPECT_NOT_POISONED(usage.ru_majflt);
        EXPECT_NOT_POISONED(usage.ru_inblock);
        EXPECT_NOT_POISONED(usage.ru_oublock);
        EXPECT_NOT_POISONED(usage.ru_nvcsw);
        EXPECT_NOT_POISONED(usage.ru_nivcsw);
}
```

16. Operasi Bit

    Pengecekan *circular shift*

```
TEST(DYNSMC, LLVMOp) {
        if (!TrackingOrigins()) return;
        BinaryOpOriginTest<S8>(XOR<S8>);
        BinaryOpOriginTest<U8>(ADD<U8>);
        BinaryOpOriginTest<S4>(SUB<S4>);
        BinaryOpOriginTest<S4>(MUL<S4>);
        BinaryOpOriginTest<U4>(OR<U4>);
        BinaryOpOriginTest<U4>(AND<U4>);
        BinaryOpOriginTest<double>(ADD<U4>);
        BinaryOpOriginTest<float>(ADD<S4>);
        BinaryOpOriginTest<double>(ADD<double>);
        BinaryOpOriginTest<float>(ADD<double>);
}
```

IV.2.1.3 Hasil Unit Testing

Berdasarkan pembangunan *unit testing* selanjutnya diserahkan pada *runner* atau kompilasi terhadap program *testing.* Dari pengamatan diatas diperoleh keluaran sebagai berikut:

```
Running main() from gtest_main.cc
[==========] Running 16 tests from 1 test cases.
[----------] Global test environment set-up.
[----------] 16 tests from DYNSMC
[ RUN      ] DYNSMC.gagalBukaFile
[       OK ] DYNSMC.gagalBukaFile (0 ms)
[ RUN      ] DYNSMC.Malloc
[       OK ] DYNSMC.Malloc (0 ms)
[ RUN      ] DYNSMC.StackTest
[       OK ] DYNSMC.StackTest (0 ms)
[ RUN      ] DYNSMC.StackTest
[       OK ] DYNSMC.StackTest (0 ms)
-------------------------dipotong-------------------------------
-
[----------] 16 tests from DYNSMC (0 ms total)
[----------] Global test environment tear-down
[==========] 16 tests from 1 test cases ran. (0 ms total)
[  PASSED  ] 16 tests.
```

IV.2.2 Wawancara

Wawancara menggunakan media *e-mail* dengan para pengembang *DynamoRIO framework* dimana program diajukan untuk daftar *plugin* eksternal. Dari hasil diskusi dapat disimpulkan bahwa modul yang dibangun sangat mudah digunakan karena dilengkapi *build* script, kebutuhan analisis *Self-Modifying Code* tersedia serta tidak ada kendala dari segi *deprecated* API ataupun mendapatkan pustaka *dependencies.*