Term Project

CS 510

Priyam Biswas

Introduction:

Valgrind is a Dynamic Binary Analysis (DPA) tool that uses Dynamic Binary Instrumentation (DPI) framework which can find memory leaks in programs, such as buffer overflows and bad memory management.

There are several existing tools in valgrind such as lackey, cachegrind, callgrind etc. In this project I had to implement provenance tracking tool in valgrind. For that, I have used valgrind version 3.12.0 and to check the program the following commands need to be given:

```
git clone https://github.com/beduin23/Valgrind_IFS.git
cd Valgrind_IFS/valgrind-3.12.0/
./autogen.sh
./configure –prefix=`pwd`/inst
sudo make
sudo make install
cd testcase
clang –m32 test1.c –o test1
../inst/bin/valgrind –tool=prov_trace ./test1
```

Implementation:

In this section, implementation issues are discussed:

Provenance Sources:

In this project, standard input (stdin) and file are considered as provenance sources. Both of these sources can be tracked via '**read**' system call and then the input we are getting from the read is considered as tainted.

Provenance target:

Variables and function pointers are considered as provenance targets.

Shadow Memory:

According to the slide, I have generated three types of shadow memory : memory, temporary and registers. Instead of type Bool in the slide, I have defined my own data structure to keep track of different provenance sources.

```
typedef struct shadow_mem_ty {
  taint_source source_t[50];
  Bool bit_taint;
  Int source_num;
} shadow_mem;
```

For each taint source, I am keeping source address and corresponding value.

Information Flow Tracking:

To properly track provenance sources, I had to instrument and develop functions.
- The trace starts at the “main” function and it’s done by the case ‘Ist_Mark’
- Next, to handle control statements like if-else, I have to instrument case ‘Iex_ITE’
- To handle data transfer from memory to temporary register, I have instrumented load instruction
- To handle data transfer from temporary register to memory, I have instrumented store instruction
- Case ‘Iex_RdTmp’ handles data transfer from one temporary to another temporary register
- Case ‘Ist_Put’ handles data transfer from temporary register to register
- Data flow from register to temporary is handled by case ‘Iex_Get’
- Different kind of operations are handled by instrumenting operational instructions.

For the implementation, the following files were helpful.
- Memcheck/mc_main.c
- lackey/lk_main.c
-VEX/pub/ir_defs.c
- VEX/pub/libvex_ir.h

Evaluation:
To evaluate my provenance tracking tool, I have generated several test cases. Among eight test cases testcase 1 and 4 deals with simple assignment, testcase 2-3 deal deals with various operations, testcase 5-6 deals conditional statements, testcase7 deals function pointer and finally testcase 8 deals file input. Testcases and their corresponding output are following.

Test Case 1:
Code:

```
#include <stdio.h>
void main(void) {
    int a, x;
    read(0, &a, sizeof(int));
    x = a;
}
```

Output:

```
2 warnings generated.
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test1
==1046== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1046== Developed by Priyam Biswas
==1046== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1046== Command: ./test1
==1046==
q
reading: arg1: fefa3fd4, arg2: 4
Trace -- 0xFEFA3FD0 [DD]: [mem: 0xFEFA3FD4: value: q]  [mem: 0xFEFA3FD5: value: \n]

Trace -- 0xFEFA3FD1 [DD]: [mem: 0xFEFA3FD4: value: q]  [mem: 0xFEFA3FD5: value: \n]

Trace -- 0xFEFA3FD2 [DD]: [mem: 0xFEFA3FD4: value: q]  [mem: 0xFEFA3FD5: value: \n]

Trace -- 0xFEFA3FD3 [DD]: [mem: 0xFEFA3FD4: value: q]  [mem: 0xFEFA3FD5: value: \n]

==1046==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Test Case 2:

Code:

```
#include <stdio.h>
void main(void) {
    int a, b, x;
    read(0, &a, sizeof(int));
    read(0, &b, sizeof(int));
    x = a + b;
}
```

```
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test2
==1084== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1084== Developed by Priyam Biswas
==1084== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1084== Command: ./test2
==1084==
q
reading: arg1: fec83fd0, arg2: 4
r
reading: arg1: fec83fcc, arg2: 4
Trace -- 0xFEC83FC8 [DD]: [mem: 0xFEC83FCC: value: r]  [mem: 0xFEC83FCD: value: \n]  [mem: 0xFEC83FD0: value: q]  [mem: 0xFEC83FD1: v
alue: \n]

Trace -- 0xFEC83FC9 [DD]: [mem: 0xFEC83FCC: value: r]  [mem: 0xFEC83FCD: value: \n]  [mem: 0xFEC83FD0: value: q]  [mem: 0xFEC83FD1: v
alue: \n]

Trace -- 0xFEC83FCA [DD]: [mem: 0xFEC83FCC: value: r]  [mem: 0xFEC83FCD: value: \n]  [mem: 0xFEC83FD0: value: q]  [mem: 0xFEC83FD1: v
alue: \n]

Trace -- 0xFEC83FCB [DD]: [mem: 0xFEC83FCC: value: r]  [mem: 0xFEC83FCD: value: \n]  [mem: 0xFEC83FD0: value: q]  [mem: 0xFEC83FD1: v
alue: \n]

==1084==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Figure : Test case 2

Test Case 3:
Code:

```
#include<stdio.h>
void input(int* var) {
  read(0, var, sizeof(int));
}
int main () {

  int a, b, x, y, z;
  input(&a);
  input(&b);

  x = b * 3;
  y = x - a;
  z = x + y;

  return 0;
}
```

Output:

```
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test3
==1114== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1114== Developed by Priyam Biswas
==1114== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1114== Command: ./test3
==1114==
s
reading: arg1: fe8eafd0, arg2: 4
h
reading: arg1: fe8eafcc, arg2: 4
Trace -- 0xFE8EAFC8 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]

Trace -- 0xFE8EAFC9 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]

Trace -- 0xFE8EAFCA [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]

Trace -- 0xFE8EAFCB [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]

Trace -- 0xFE8EAFC4 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC5 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC6 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC7 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC0 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC1 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC2 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

Trace -- 0xFE8EAFC3 [DD]: [mem: 0xFE8EAFCC: value: h]  [mem: 0xFE8EAFCD: value: \n]  [mem: 0xFE8EAFD0: value: s]  [mem: 0xFE8EAFD1: v
alue: \n]

==1114==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Figure : Test case 3

Test Case 4:

Code:

```
#include <stdio.h>
void main(void) {
    int a, b, x;
    read(0, &a, sizeof(int));
    read(0, &b, sizeof(int));
    x = a;
    x = b;
}
```

Output:

```
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test4
==1151== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1151== Developed by Priyam Biswas
==1151== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1151== Command: ./test4
==1151==
s
reading: arg1: fef30fd0, arg2: 4
g
reading: arg1: fef30fcc, arg2: 4
Trace -- 0xFEF30FC8 [DD]: [mem: 0xFEF30FD0: value: s]  [mem: 0xFEF30FD1: value: \n]

Trace -- 0xFEF30FC9 [DD]: [mem: 0xFEF30FD0: value: s]  [mem: 0xFEF30FD1: value: \n]

Trace -- 0xFEF30FCA [DD]: [mem: 0xFEF30FD0: value: s]  [mem: 0xFEF30FD1: value: \n]

Trace -- 0xFEF30FCB [DD]: [mem: 0xFEF30FD0: value: s]  [mem: 0xFEF30FD1: value: \n]

Trace -- 0xFEF30FC8 [DD]: [mem: 0xFEF30FCC: value: g]  [mem: 0xFEF30FCD: value: \n]

Trace -- 0xFEF30FC9 [DD]: [mem: 0xFEF30FCC: value: g]  [mem: 0xFEF30FCD: value: \n]

Trace -- 0xFEF30FCA [DD]: [mem: 0xFEF30FCC: value: g]  [mem: 0xFEF30FCD: value: \n]

Trace -- 0xFEF30FCB [DD]: [mem: 0xFEF30FCC: value: g]  [mem: 0xFEF30FCD: value: \n]

==1151==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Figure : Test case 4

Test Case 5:

Code:

```
#include <stdio.h>
void main(void) {
    int a, b, x;
    read(0, &a, sizeof(int));
    read(0, &b, sizeof(int));
    if( a < 1)
      x  = a;
    else
        x =  b;
}
```

Output:

```
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test5
==1173== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1173== Developed by Priyam Biswas
==1173== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1173== Command: ./test5
==1173==
a
reading: arg1: fefa8fd0, arg2: 4
b
reading: arg1: fefa8fcc, arg2: 4
Trace -- 0xFEFA8FC8 [DD]: [mem: 0xFEFA8FCC: value: b]  [mem: 0xFEFA8FCD: value: \n]

Trace -- 0xFEFA8FC9 [DD]: [mem: 0xFEFA8FCC: value: b]  [mem: 0xFEFA8FCD: value: \n]

Trace -- 0xFEFA8FCA [DD]: [mem: 0xFEFA8FCC: value: b]  [mem: 0xFEFA8FCD: value: \n]

Trace -- 0xFEFA8FCB [DD]: [mem: 0xFEFA8FCC: value: b]  [mem: 0xFEFA8FCD: value: \n]

==1173==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Figure : Test case 5

Test Case 6:

Code:

```
#include <stdio.h>
void main(void)
{
    int a, b, x;
    read(0, &a, sizeof(int));
    read(0, &b, sizeof(int));
    if( a > 1)
      x  = a;
    else
      x =  b;
}
```

Output:

```
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$ ../inst/bin/valgrind --tool=prov_trace ./test6
==1194== Provenance_Tracking, Dynamic Information Flow System tool in Valgrind
==1194== Developed by Priyam Biswas
==1194== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==1194== Command: ./test6
==1194==
a
reading: arg1: fecebfd0, arg2: 4
b
reading: arg1: fecebfcc, arg2: 4
Trace -- 0xFECEBFC8 [DD]: [mem: 0xFECEBFD0: value: a]  [mem: 0xFECEBFD1: value: \n]

Trace -- 0xFECEBFC9 [DD]: [mem: 0xFECEBFD0: value: a]  [mem: 0xFECEBFD1: value: \n]

Trace -- 0xFECEBFCA [DD]: [mem: 0xFECEBFD0: value: a]  [mem: 0xFECEBFD1: value: \n]

Trace -- 0xFECEBFCB [DD]: [mem: 0xFECEBFD0: value: a]  [mem: 0xFECEBFD1: value: \n]

==1194==
priyam@priyam-pc:~/valgrind/valgrind-3.12.0/testcase$
```

Figure : Test case 6

Test Case 7:

Code:

```
#include<stdio.h>
void func(int a) {

}
int main () {

    int a;
    read(0, &a, sizeof(int));
    int (*fn) (int);
    fn = func;
    fn(a);
    return 0;
}
```

Output:

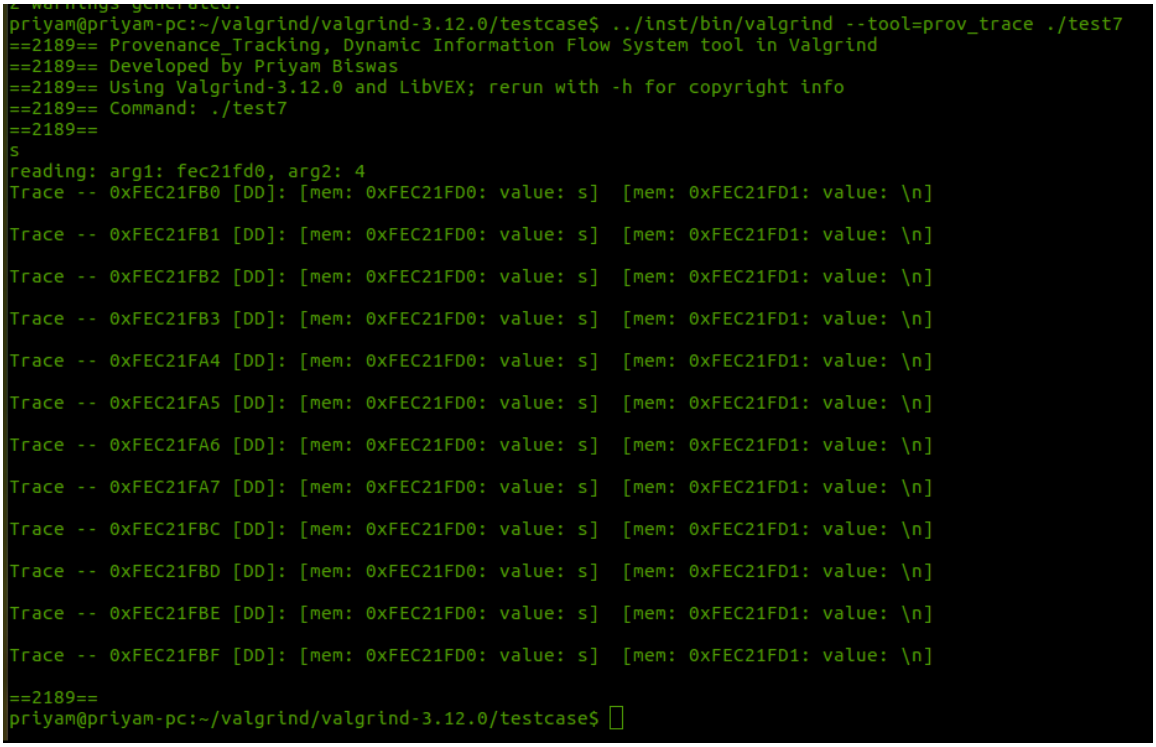

Figure : Test case 7

Test Case 8:

Code:

```
#include<stdio.h>
int main() {
    FILE *file;
    char buf[100];
    file = fopen("a.txt", "rb");
    fseek(file, 0, SEEK_SET);
    fread(buf, 10, 1, file);
    fclose(file);
    return 0;
}
```

Output:

Conclusion:

This project helped me to better understand how valgrind works as well as got a practical idea of provenance tacking what we have learned theoretically in the class. The slides discussed in the class was very helpful to start the implementation. However the limitation of my tool is that it works in 32 bit.

References:

[1] Nethercote, Nicholas, and Julian Seward. "Valgrind: a framework for heavyweight dynamic binary instrumentation." In ACM Sigplan notices, vol. 42, no. 6, pp. 89-100. ACM, 2007. http://valgrind.org/docs/valgrind2007.pdf

[2] https://www.cs.purdue.edu/homes/xyzhang/spring17/5-slicing-IFS.updated.pdf

[3] The Valgrind Quick Start Guide: http://valgrind.org/docs/manual/QuickStart.html