CSE509-Lab3 Binary Instrumentation

Sanket Goutam, Sai Bhavana Ambati

1 Warmup Questions

What are the key steps in writing a Pintool? What are the similarities across the three pintools you studied? What are the differences?

There are two major parts of any Pintool. The *Instrumentation routine* and the *Analysis Routine*. Every Pintool we write will register instrumentation callback routines with Pin that are called from Pin whenever new code needs to be generated. This instrumentation callback routine represents the instrumentation component. It inspects the code to be generated, investigates its static properties, and decides if and where to inject calls to analysis functions.

The analysis function gathers data about the application. This is the part of the Pintool that we implement based on the kind of analysis we want to perform on different applications.

For example, among the sample programs we looked at the following are the analysis routines for each:

- Instruction Count : docount() gets called everytime an instruction is encountered
- Memory Reference Trace : In this program, every time an instruction is called the instruction routine simply checks if the instruction is a memory read or a memory write instruction. Based on this, it will invoke the analysis routine RecordMemWrite() or RecordMemRead() handlers which simply print the trace logs.
- Finding Values of function arguments: In this tool, we monitor function calls malloc() and free() and using the analysis routines we print their arguments and return addresses into a trace file.

Apart from the instruction routine and analysis routine, the basic structure of any Pintool also remains similar. The structure of the main() of every Pintool will remain the same, i.e., PIN_Init followed by instruction trace functions, and then the exit handler PIN_AddFiniFuction.

Summarize your observations when you run these tools on simple and complex applications.

```
./pintool3 inscount0 hello-world/helloWorld
    Hello World!!
Count 2171244

./pintool3 inscount0 /bin/ls
hello-world    inscount0.out  makefile.rules   malloctrace.out  obj-
   intel64           pinatrace.cpp  pin.log   pintool.log
inscount0.cpp  makefile       malloctrace.cpp  MyPinTool.cpp
   output_inscount0.out  pinatrace.out  pintool3  temp.tmp
Count 478635

./pintool3 inscount0 /bin/gedit
Unable to init server: Could not connect: Connection refused
```

```
(gedit:13973): Gtk-WARNING **: 20:08:35.192: cannot open display:
Count 37994877
```

Note: I was using SSH session without X-forwarding so firefox didn't even run without X-Display being set.

Usage of pintools to perform different kinds of analysis is very similar to the ptrace based system call interception we did in the previous lab. However, this seems to be much more powerful than the ptrace based interception. For instance, in the Memory reference tool we were able to use APIs exposed by Pin to determine what kind of an instruction it is and perform the analysis routine accordingly. This granular level of program abstraction allows a lot of flexibility in designing tools for analysis, debugging, or even attack mitigation.

We can design a Pintool to monitor system calls, function arguments, return addresses, maintain a shadow stack to ensure that an attacker is not able to modify any arguments or return addresses on the stack. This level of granularity and expressiveness is difficult to achieve with ptrace based system call interception.

2 Backward-Edge CFI

Implementation of the Backward-Edge CFI tracking pintool is provided in the tar.gz file. Instructions on how to compile the binaries, and how to run them is provided in the README.md file.

To start testing the implementation, you will need to extract the contents of sanket_pintool.tar.gz into *PIN_BASE_FOLDER/source/tools*. Please refer to the directory structure below to ensure that the tool is present in the correct directory. The scripts to run the program depends on exact location of binaries.

```
sekar@sekar-VirtualBox:~/Desktop/Lab3/pin-3.20-98437-gf02b61307-gcc-
    linux/source/tools$ cd Lab3_sanket_pintool/

sekar@sekar-VirtualBox:~/Desktop/Lab3/pin-3.20-98437-gf02b61307-gcc-
    linux/source/tools/Lab3_sanket_pintool$ ls

exploit  hello-world  makefile  makefile.rules  old_pintool.cpp
    pintool  README.md  sanket_pintool_tls.cpp


sekar@sekar-VirtualBox:~/Desktop/Lab3/pin-3.20-98437-gf02b61307-gcc-
    linux/source/tools/Lab3_sanket_pintool$ pwd

/home/sekar/Desktop/Lab3/pin-3.20-98437-gf02b61307-gcc-linux/source/
    tools/Lab3_sanket_pintool
```

Before you can start testing the implementation, you will need to build all the binaries. Steps are provided in the README.md file. Basically, there are three binaries you need to build, sanket_pintool_tls, exploit code binaries inside *exploit* folder, and helloWorld binary (if needed).

2.1 Analysis on sample programs – Exceptions

The README file provides all the test cases that I ran the program against, including complex applications that require multithreading. Some exceptions that I noticed, which I was not able to handle is mismatch violations occuring for GTK based applications.

When running the pintool against gedit or firefox (with XDISPLAY set), I notice a lot of mismatches. With gedit, it works out well in the sense that the application runs properly and pintool is able to generate the analysis report once the application terminates.

However, with firefox the pin program continuously detaches from the program and keeps on trying to reattach. Firefox never runs and the pintool just keeps detaching and reattaching itself. When I exit the program manually by sending 'Ctrl+c' signal, it terminates and displays the analysis done till that point.

The analysis report from gedit is shown below (limiting to functions with error count ¿ 2). As we can see, most of the mismatches are happening due to GTK functions. I noticed a similar behavior with Firefox as well.

As an aside, running GTK applications with XDISPLAY turned off doesn't give as many violations. Although, without XDISPLAY there aren't many threads/ function calls being made either.

```
================================================
This application is instrumented by sanket_pintool_tls
Generating analysis report for Backward-Edge CFI violations
================================================
================================================================================

Thread ID: 0
                                                Function Name |     Count
--------------------------------------------------------------------------------

                                        g_type_create_instance|         5
                                                      dlerror|        53
                                    g_object_notify_by_pspec|         5
                          gtk_distribute_natural_allocation|         3
                                                g_signal_emit|        14
                        gtk_style_properties_lookup_property|         8
                                     g_io_module_get_type|         3
                                                   lt_dlsym|         9
                                  g_io_error_from_errno|         3
                                    g_object_new_valist|         3
                                         g_object_unref|         6
                                             lt_strlcpy|        10
                                        g_module_symbol|        41
                                  ca_cache_store_sound|         8
                       peas_plugin_loader_garbage_collect|         5
                                        g_module_open|        16
                      g_io_extension_point_get_extensions|         3
                                  gtk_model_button_new|         4
                             g_signal_handler_disconnect|        10
                           g_object_new_with_properties|         3
                                        g_closure_unref|        15
                                        _dl_catch_error|        53
                                           g_object_new|         6
                             peas_object_module_get_type|         5
                                     _dl_catch_exception|        55
                                       g_type_module_use|         8
                                  g_signal_emit_valist|        15
                          gtk_container_get_path_for_child|         8
                           gtk_widget_get_preferred_height|        10
                           gtk_widget_get_preferred_height|        10
                                                    dlsym|        51
                            gtk_widget_get_preferred_width|        10
================================================================================

Stack Smashing Detected in program
Total mismatches detected :592
```