Project 1: Dissecting Malware

1. Context

You are an administrator of a system. After you find out your system is compromised, you find ***a malware binary*** left in the system (i.e., *timebomb2*). As you were curious about the malware, you run the program. As usual, the malware just terminates without doing anything (see Figure 1).

```
~ > class > cf ./timebomb2
This program is legitimate.
 - Source: Dude-Trust-Me.
```

Figure 1. Timebomb2 program doing nothing

Can you figure out how to run the malware, so that it would do (or expose) malicious activities?

2. Description

You are given a *fake malicious program* ("timebomb2" hereafter). It does *suspicious tasks*, but it *does not harm your system*.

> Note that it is ***not recommended to run this program on machines managed by the university***. While benign eventually, the activity of this sample program might be considered as *suspicious/malicious by some programs or administrators*. If you are running it on one of the university-owned machines, and if you are contacted by anyone regarding this sample program, you should tell me as soon as possible.

Given the "timebomb2" program, you need to create your own ***pintool*** to analyze the program. Specifically, you need to identify (1) ***what are the suspicious actions*** (2) ***what are the specific conditions*** "timebom2" expects. You are allowed to use the following tools.

- **Debuggers** such as gdb
- **Disassemblers** such as IDA, Ghidra
- **Any pintool available on the Internet**

Here are some specifications for the program. *Note that in practice, you will never get such information, and it is up to you to figure out that.*

- **Compiler**: The binary is **statically** compiled by **gcc**. This means that it would not make a library call.
- **Packer:** The sample program is packed by the **UPX** (https://upx.github.io/). If you do not know what a "packer" is, please read https://en.wikipedia.org/wiki/UPX.

3. What to do

1. Download the Pin and install (i.e., compile) it – check the lecture slides for this.
2. Download the sample file: "timebom2".
3. Make your own pintool to analyze the sample.
4. Write the report as described below.

4. Hints and Extra credit

Modification Preventing UPX Decompression. UPX supports the "decompress" function. However, I made a slight modification on the "timebomb2" binary so that you can't simply use the decoding function of UPX to decompress. Figuring out how to modify the "timebomb2" binary so

that it can be "decompressed" (using "upx -d" command) and writing down what was the key modification that prevents the decompression will give some extra credit (e.g., +5).

Warning (Cheating related). If you have submitted the document explaining how you modify the binary to decompress, you can use the decompressed (i.e., the original) binary for your analysis. If you cannot submit the document, you are not allowed to use the decompressed binary. Moreover, if you use the decompressed binary without the submission, I would consider that as ***cheating*** (because if you did not decompress it by yourself, how did you even get the file?)

Warning. *Your pintool **should not change** the timebomb2's **instructions** (e.g., some replace functions or remove some instructions).* Please only use Pin APIs that change the data of memory/registers. If you change instructions, a certain deduction of your score can be applied.

5. Guide

1. **Play with UPX to understand it.** You can use UPX to pack a simple program to understand what UPX does and how (Note that UPX can't compress a very small binary. For example, if you simply compile a hello world program, it is too small. That's why I compiled it with "-static" option).
2. **Logging.** Logging instructions generated by UPX would be useful in identifying the original binary's instructions.
3. **Branches/Conditions.** Focus on branches and their conditions.
4. **Unexplored Paths.** Focus on branches that are not taken.
5. When the condition meets, you will see a *(very slightly) different output*. The output (in the console) will have ***one empty line before the message*** (i.e., the "This program is legitimate.\n - Source: Dude-Trust-Me.\n").

6. What to submit?

1. Your Pintool code. (**Submit a single .cpp file, please**)
2. A report that includes
   (1) high-level descriptions of how your pintool works (around **0.5 page**) (15%),
   (2) what are the triggering conditions that the sample expects (there are four conditions) (around **0.5 page**) (25%),
   (3) what is the suspicious activity that the sample would do (hint: it will do some operations related to files) (around **1 page** with the details such as instructions for the activities) (40%),
   (4) how did you deal with the packer (around **0.2 page**) (5%),
   (5) how did you analyze the sample (around **0.5 page**) (15%).
   - Please submit only two files: **(1) .cpp file**, and **(2) .pdf file**.