DisProTrack: Distributed Provenance Tracking over Serverless Applications

Utkalika Satapathy[1], Rishabh Thakur[1], Subhrendu Chattopadhyay[1], Sandip Chakraborty[2]
[1]*IIT Kharagpur, Kharagpur, India 721302*
[2]*IDRBT, Hyderabad, India 500057*
This work is accepted in Infocom 2023

Abstract

Provenance tracking, crucial for debugging system vulnerabilities, is widely employed. Existing models focus on monolithic applications, but modern DevOps-based service architectures pose challenges. This paper introduces a novel Universal Provenance Graph (UPG) approach, utilizing a Loadable Kernel Module (LKM) for runtime unit identification and a log optimization method for effective provenance tracking in serverless architectures. Evaluation with various benchmarked serverless applications demonstrates the model's effectiveness.

Provenance Graphs

Provenance data is the metadata of a process capturing origin and modification details throughout its lifecycle. The resulting Provenance Graph is a causal graph depicting dependencies between system subjects (e.g., processes) and objects (e.g., files, network sockets).

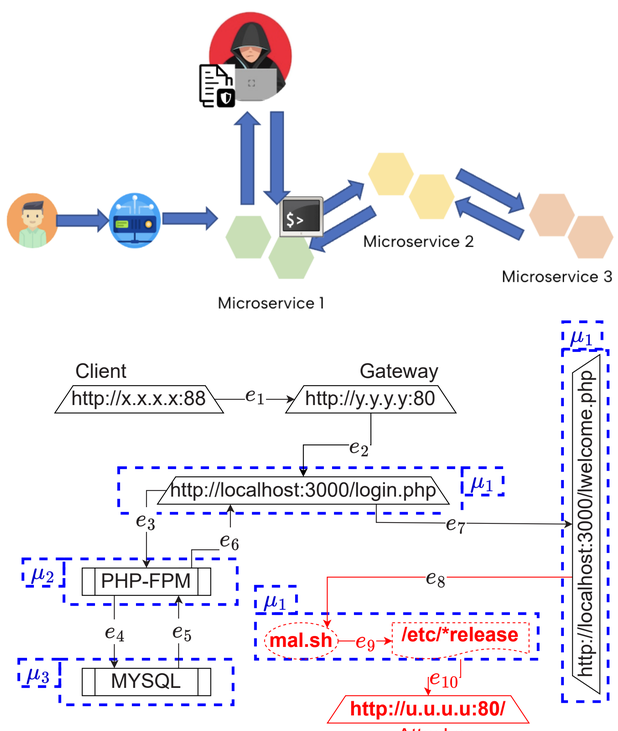


Figure 1. The Top Figure shows an Adversarial Model for Confidential Data Theft, and the bottom figure shows the UPG generated for the same.

Log Message Generating Function

It is a library function used for printing a LMS in either terminals, specific log store files, or log databases. For example, consider the following C code snippet with a popular logging library `Log4C`.

```
log4c_category_log(NULL,
LOG4C_PRIORITY_ERROR,"Hello World!");
```

Here the LMS is `Hello World!` and the LMGF is `log4c_category_log`.

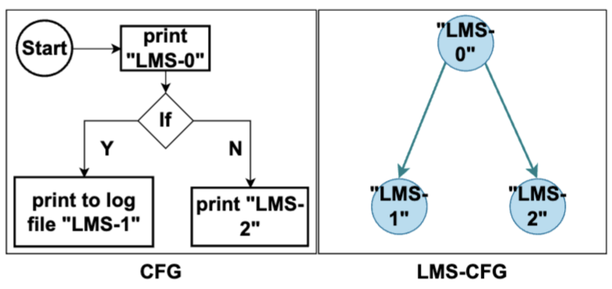


Figure 2. Example of CFG and LMS-CFG

UPG Construction - Challenges

1. Combining application logs from different micro-services: Log messages formatting, timestamp formatting, process descriptors vary
2. Combining the system log with the application logs: Container-based sandboxing shares same pid namespace
3. Identification of execution units
4. Dependency explosion and handling confounding root causes: Reverse query results more than one root cause

Contact Me

DisProTrack GitHub:`https://github.com/usatpath01/DisProTrack`
My Homepage: `https://usatpath01.github.io/`

DisProtrack Oveview

Contributions:
1. Design of the UPG from application and system logs: Static analyzer module generates the application-specific Log Message String-Control Flow Graph (LMS-CFG) from the application binaries which provides a profile of the application.
2. Runtime execution unit identification: Developed a Linux LKM that can intercept the system calls generated during execution time to identify the semantic relationship between the system logs and the application logs.
3. Utilization of Regular Expression to improve search efficacy: Instead of storing the raw log messages in the UPG, we propose conversion and storage of an equivalent regular expression.

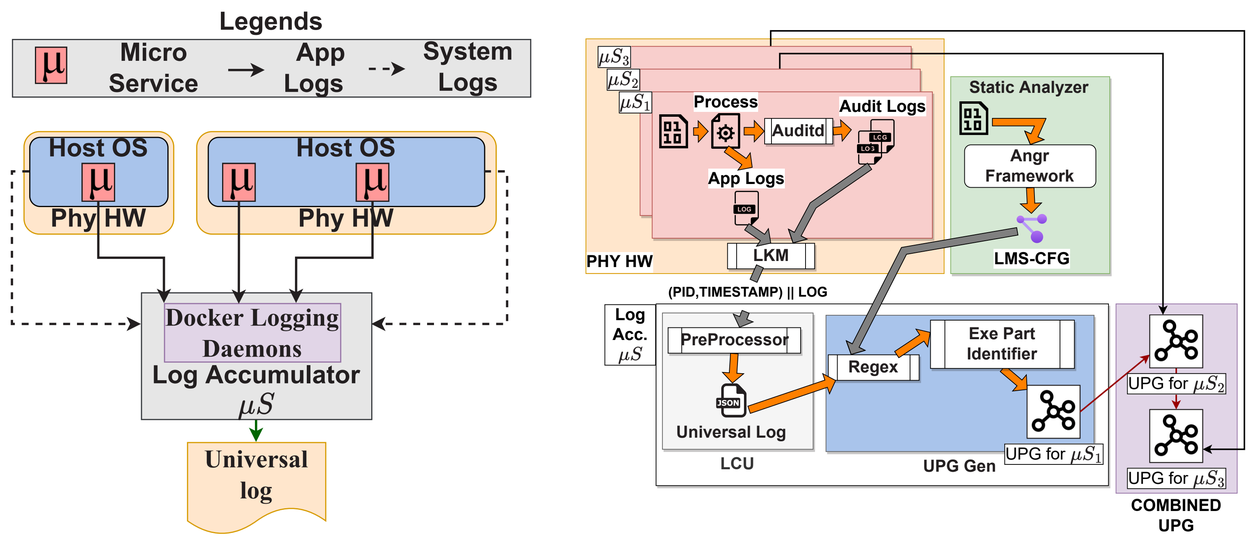


Figure 3. Left side, Environment Setup. Right side, DisProTrack Execution Flow.

Results and Analysis

1) Since DisProTrack is targeted for serverless applications; resource overhead is a major concern. Therefore, experimentally we want to understand the resource overhead of the framework.

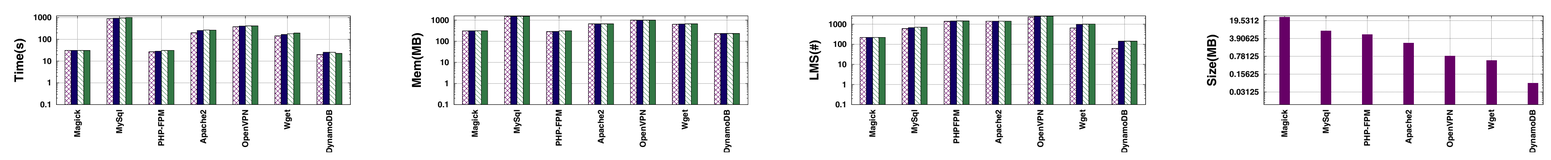
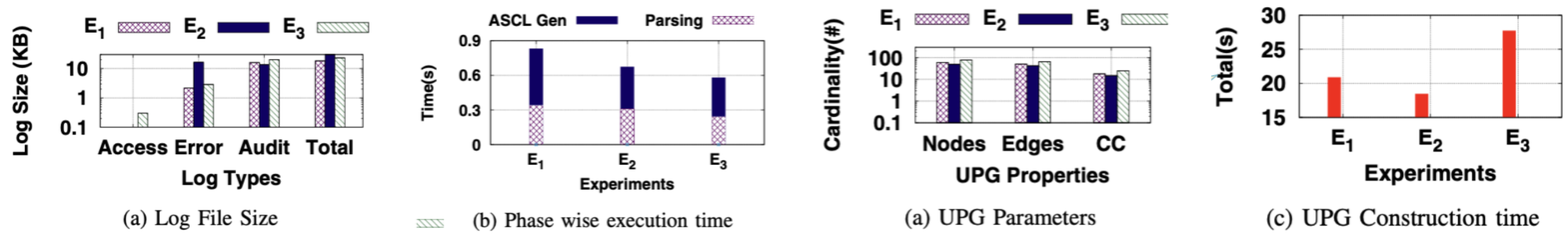


Figure 4. Performance Evaluation - Static Analysis – The Y-axes are in logarithmic scale



(a) Log File Size (b) Phase wise execution time (a) UPG Parameters (c) UPG Construction time

Figure 5. Performance Evaluation - Runtime Analysis

2) We also want to understand how effective DisProTrack is for identifying malicious activities

```c
#include<unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define Foo2(int j) ({printf("Empty function\n");})

int Foo1(int j){
    FILE *f = fopen("testfile.txt", "a");
    if (f == NULL) {
        printf("Error: File open\n");return -1;
    }
    if (fprintf(f, "Write to testfile.txt\n") < 23) {
        fprintf(stderr, "Error writing to file\n");
        return -1;
    }
    return(0);
}
/***********************/
int main(void){
    printf("Start\n");int j = rand();
    if(j%2==0){
      printf("Call Foo1(), j = %d\n", j);Foo1(j);
    }else{
      printf("Call Foo2(), j = %d\n", j);Foo2(j);
    }
    printf("End\n");
    return 0;
}
```

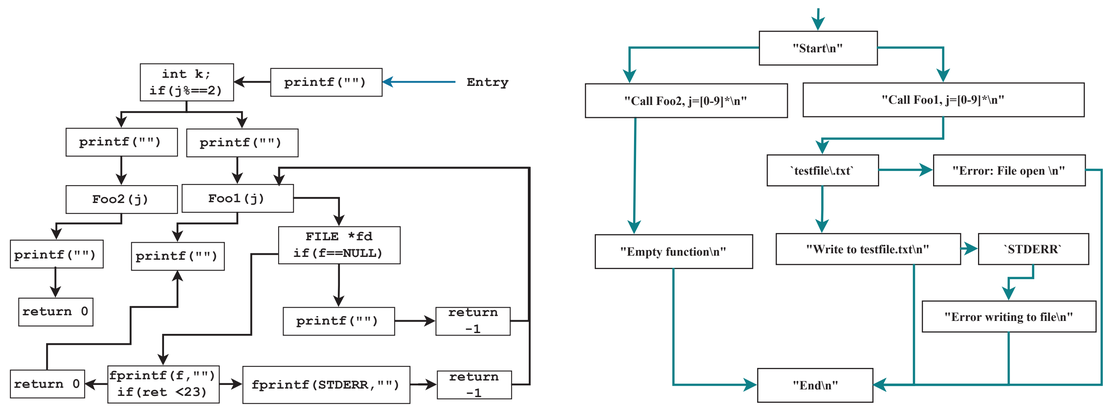


Figure 6. Left Side: PoC Program, Middle and Right Side: A PoC Case Study to Analyze the Accuracy of DisProTrac

Conclusion

1. DisProTrack can be deployed as a microservice on top of the SLC without instrumenting the source code of the applications
2. Implementation is open-sourced
3. DisProTrack has a minimal memory footprint (∼KB) & responds within 20s-30s.

Acknowledgement
I would like to thank the "Infocom 2023" organizers for awarding me with the student travel grant