DisProTrack: Distributed Provenance Tracking over Serverless Applications

Utkalika Satapathy[1], Rishabh Thakur[1], Subhrendu Chattopadhyay[2], Sandip Chakraborty[1]

IEEE International Conference on Computer Communications 2023 IEEE INFOCOM

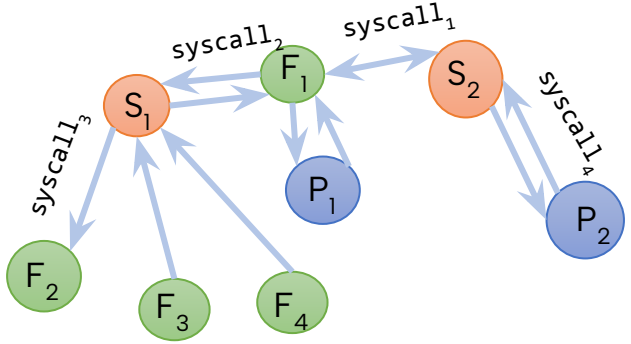
1 UbiNet
IIT Kharagpur, India

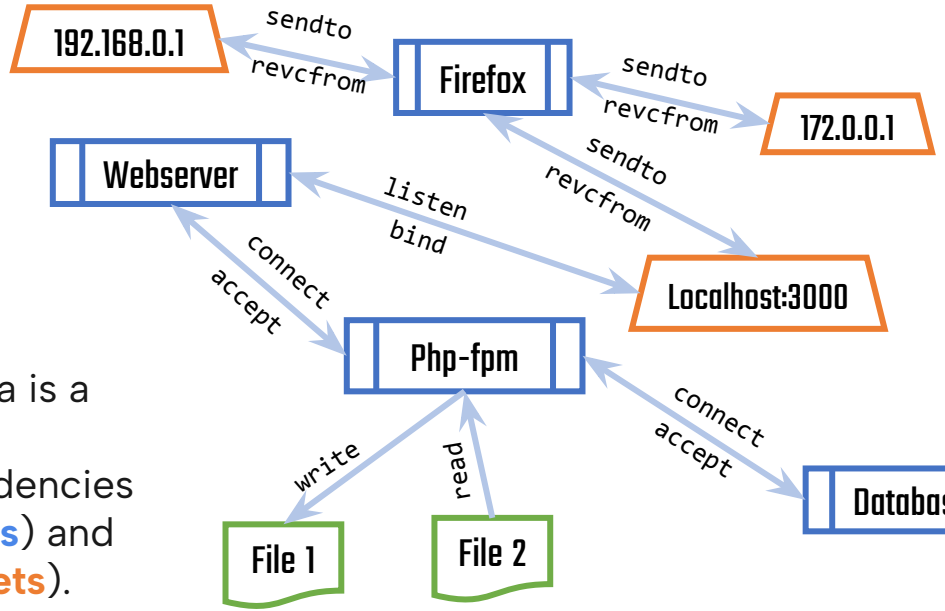
2 IDRBT Explore, Enable, Excel
Network Innovation Lab,
IDRBT, India

Provenance Tracking

- In traditional Monolithic applications Provenance Tracking can help
- **Provenance Data** is the metadata of a process (origin, history of modifications)
- To debug system vulnerabilities
- To find the root causes behind faults, errors, or crashes over a running system
- Universal Provenance Graphs (UPG) ensures observability

Universal Provenance Graph (UPG)

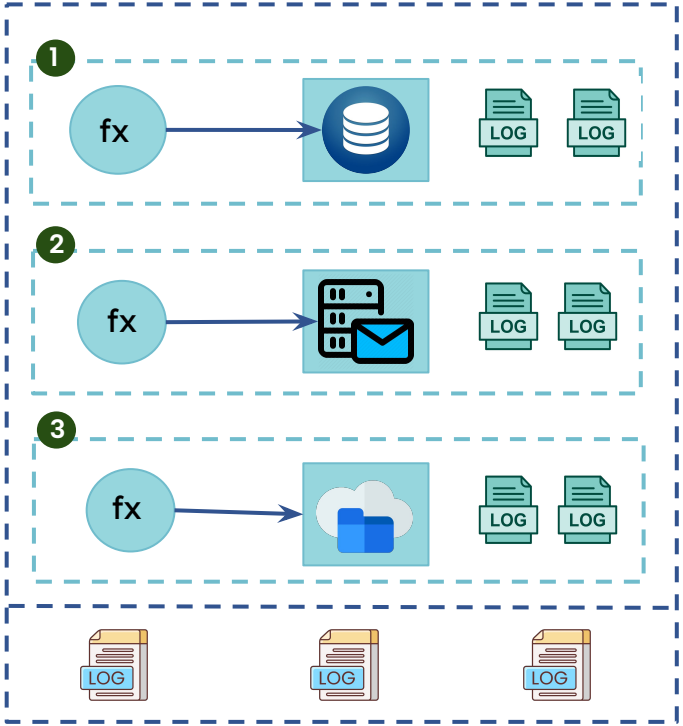




- A graph generated from provenance data is a **Provenance Graph** of a process.
- It's a causal graph that stores the dependencies between system subjects (e.g., **processes**) and system objects (e.g., **files**, **network sockets**).

Serverless Computing (SLC)

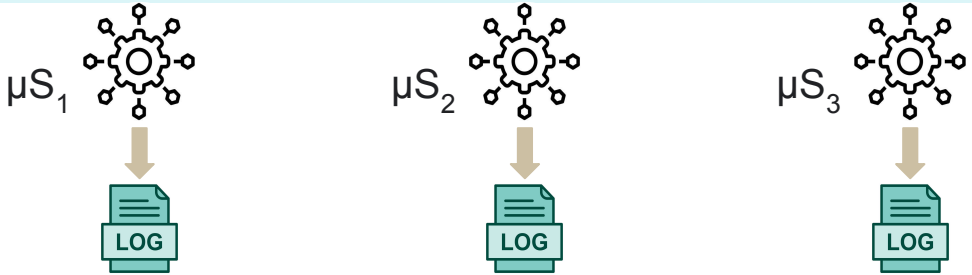
- Provides abstractions of the underlying infrastructure
- Operational expenditure is reduced when service computations are stateless, elastic, and distributed
- Good platform when everything is working fine.
- What happens if something is not right?
  - Too distributed
  - Observability and debugging is challenging



Serverless Architecture

UPG Construction in SLC - Challenges

Challenge 1 – Combining application logs from different micro-services




```
127.0.0.1 - - [17/Dec/2021:22:58:37 +0530] "GET /todo/hashgen.php HTTP/1.1" 200 539 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
127.0.0.1 - - [17/Dec/2021:22:58:45 +0530] "GET /todo/hashgen.php?get_hash=RISHABH
HTTP/1.1" 200 594 "http://localhost/todo/hashgen.php" "Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
```

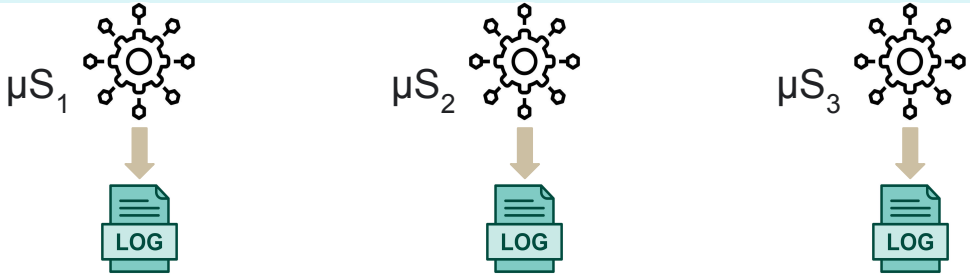
App Log 1

```
172.18.0.1 - - [13/Jul/2022:07:49:00 +0000] "GET /login/login.php
HTTP/1.1" 200 1226
pid = 12490, date = [1657698553278104923] 172.18.0.1 - -
[13/Jul/2022:07:49:13 +0000] "POST /login/login.php HTTP/1.1" 302
-
```

App Log 2

UPG Construction in SLC - Challenges

Challenge 1 – Combining application logs from different micro-services




```
127.0.0.1 - - [17/Dec/2021:22:58:37 +0530] "GET /todo/hashgen.php HTTP/1.1" 200 539 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
127.0.0.1 - - [17/Dec/2021:22:58:45 +0530] "GET /todo/hashgen.php?get_hash=RISHABH
HTTP/1.1" 200 594 "http://localhost/todo/hashgen.php" "Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
```

App Log 1

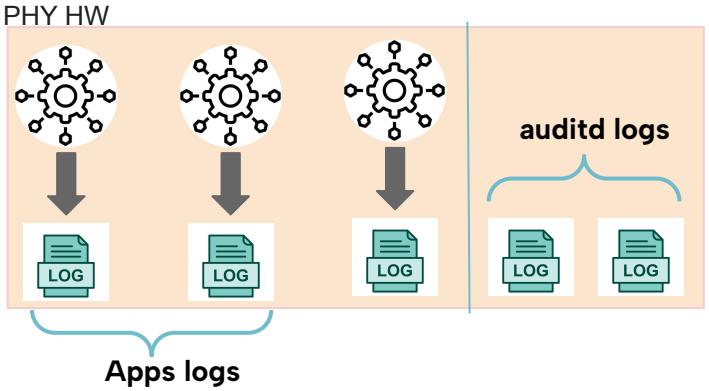
```
172.18.0.1 - - [13/Jul/2022:07:49:00 +0000] "GET /login/login.php
HTTP/1.1" 200 1226
pid = 12490, date = [1657698553278104923] 172.18.0.1 - -
[13/Jul/2022:07:49:13 +0000] "POST /login/login.php HTTP/1.1" 302
-
```

App Log 2

Log messages formatting, timestamp formatting, process descriptors vary

UPG Construction in SLC - Challenges

Challenge 2 – Combining the system log with the application logs

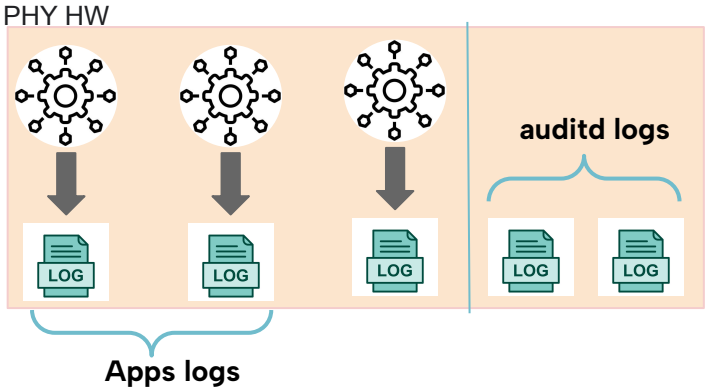



```
{'srn': '23228', 'ts': '1639762166.872', 'type': 'PROCTITLE',
'data': {'proctitle':
'2F62696E2F7368002F7573722F7362696E2F736572766963650061756469746400
073746F70'}}
```

```
127.0.0.1 - - [17/Dec/2021:22:58:37 +0530] "GET /todo/hashgen.php
HTTP/1.1" 200 539 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:95.0) Gecko/20100101 Firefox/95.0"
127.0.0.1 - - [17/Dec/2021:22:58:45 +0530] "GET
/todo/hashgen.php?get_hash=RISHABH HTTP/1.1" 200 594
"http://localhost/todo/hashgen.php" "Mozilla/5.0 (X11; Ubuntu;
Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
```

UPG Construction in SLC - Challenges

Challenge 2 – Combining the system log with the application logs



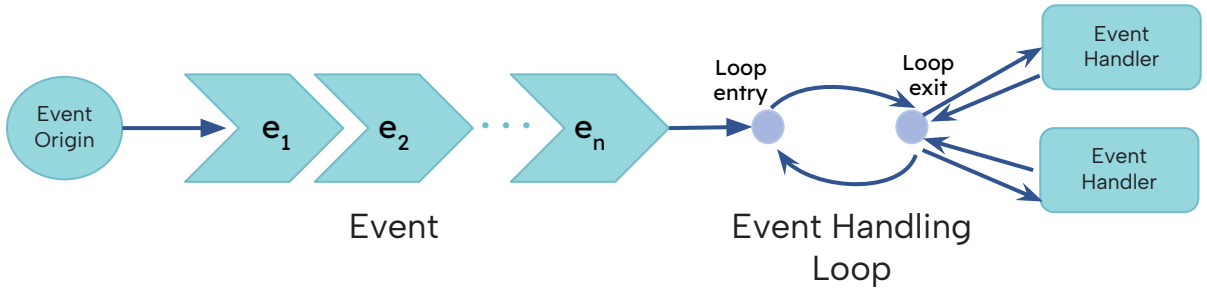

```
{'srn': '23228', 'ts': '1639762166.872', 'type': 'PROCTITLE',
'data': {'proctitle':
'2F62696E2F7368002F7573722F7362696E2F736572766963650061756469746400
073746F70'}}
```

```
127.0.0.1 - - [17/Dec/2021:22:58:37 +0530] "GET /todo/hashgen.php
HTTP/1.1" 200 539 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:95.0) Gecko/20100101 Firefox/95.0"
127.0.0.1 - - [17/Dec/2021:22:58:45 +0530] "GET
/todo/hashgen.php?get_hash=RISHABH HTTP/1.1" 200 594
"http://localhost/todo/hashgen.php" "Mozilla/5.0 (X11; Ubuntu;
Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
```

Container-based sandboxing shares same pid namespace

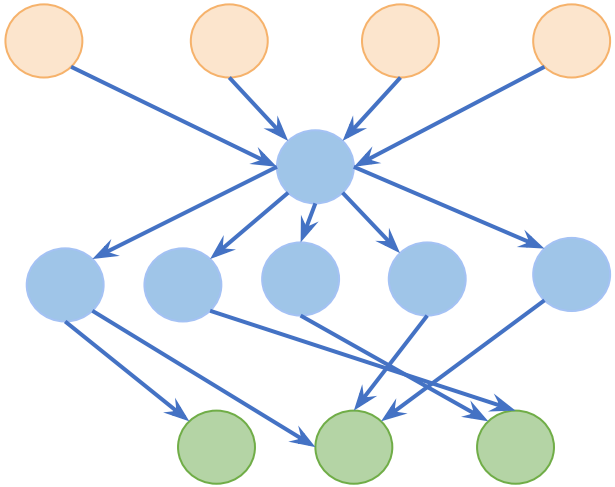
UPG Construction in SLC - Challenges

Challenge 3 – Identification of execution units



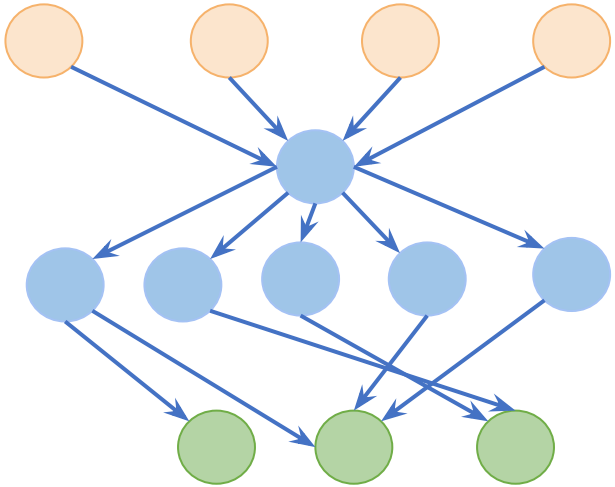
UPG Construction in SLC - Challenges

Challenge 4 – Dependency explosion and handling confounding root causes



UPG Construction in SLC - Challenges

Challenge 4 – Dependency explosion and handling confounding root causes



Reverse query results more than one root cause

Contribution

1. Design of the UPG from application and system logs

2. Runtime execution unit identification:

3. Utilization of Regular Expression to improve search efficacy

4. Implementation and evaluation

Contribution

1. Design of the UPG from application and system logs

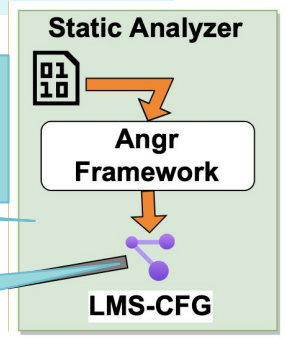
Contribution

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.

The LMS-CFG provides a profile of the application.

Provides a novel approach for constructing a UPG from application logs and system logs using the LMS-CFG profiles for different application micro-services.



Generation of LMS-CFG

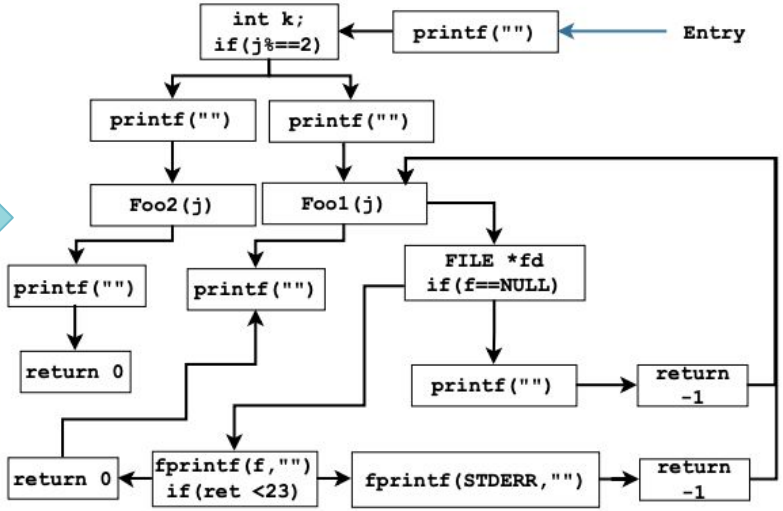
```c
#include<unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define Foo2(int j) ({printf("Empty function\n");})

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;
}
```

PoC Program for Accuracy Analysis

Generation of LMS-CFG

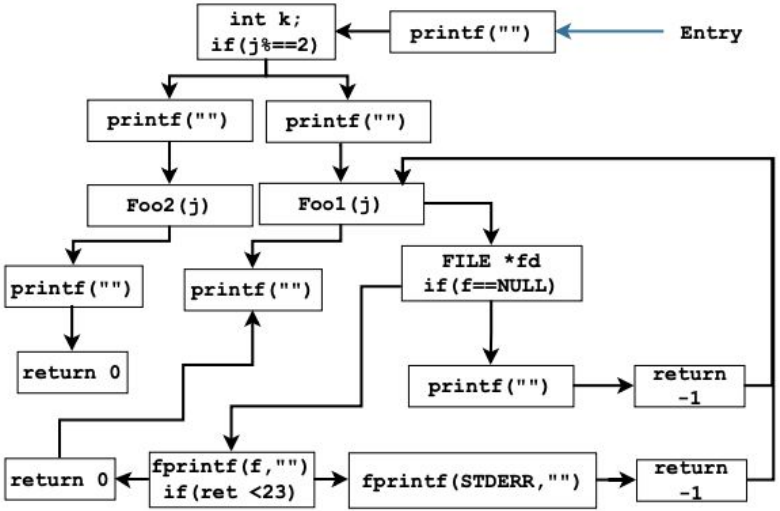
```c
#include<unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define Foo2(int j) ({printf("Empty function\n");})

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;
}
```

PoC Program for Accuracy Analysis

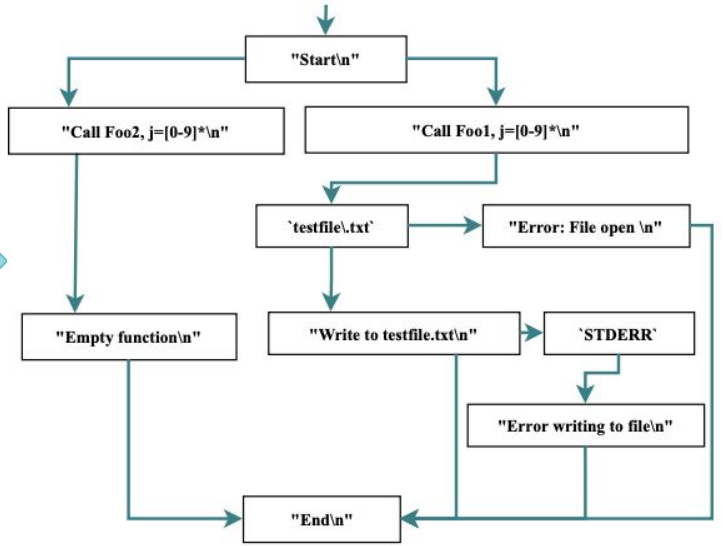


CFG for Test Program

Generation of LMS-CFG



CFG for Test Program

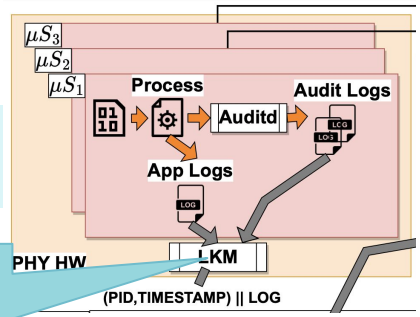


LMS-CFG for Test Program

Contribution

2. Runtime execution unit identification:

We develop a Linux LKM which can intercept the system calls generated during execution time to identify the semantic relationship between the system logs and the application logs.



Contribution

2. Runtime execution unit identification:

- We develop a Linux LKM which can intercept the system calls generated during execution time to identify the semantic relationship between the system logs and the application logs.

```
pid = 3545, date = [1639807813132979139] type=SYSCALL msg=audit(1639807813.130:23446): arch=c000003e
syscall=3 success=yes exit=0 a0=b a1=7f1d34aeb0a0 a2=7f1d35644090 a3=7f1d35628110 items=0ppid=3586
pid=3590 auid=4294967295 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none)
ses=4294967295 comm="apache2"exe="/usr/sbin/apache2" subj==unconfined key=(null)ARCH=x86_64
SYSCALL=close AUID="unset" UID="www-data" GID="www-data" EUID="www-data" SUID="www-data"
FSUID="www-data" EGID="www-data" SGID="www-data" FSGID="www-data"
```

```
pid = 3586, date = [1639807817707045812] 127.0.0.1 - - [18/Dec/2021:11:40:17 +0530] "GET /todo/
HTTP/1.1" 302 461 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0"
```

Contribution

2. Runtime execution unit identification:

- We develop a Linux LKM which can intercept the system calls generated during execution time to identify the semantic relationship between the system logs and the application logs.
- We propose a heuristic which identifies the syscalls to mark the application's event handling loops by tracing back the application binaries.
- These event handling loops can be refereed during the runtime partitioning of execution units across the micro-services.

Contribution

2. Runtime execution unit identification:


```
[
        "[Sat Dec 18 11:39:23.251164 2021] [watchdog:debug[pid 3586] mod_watchdog.c(452): AH010033: Watchdog: Running with WatchdogInterval
1000ms",
        "[Sat Dec 18 11:39:23.251219 2021] [watchdog:debug[pid 3586] mod_watchdog.c(461): AH02974: Watchdog: found parent providers."
    ],
    [
        "[Sat Dec 18 11:39:23.251226 2021] [watchdog:debug[pid 3586] mod_watchdog.c(507): AH02977: Watchdog: found child providers."
    ],
    [
        {
            "srn": "23632",
            "exe": "/usr/sbin/apache2",
            "ts": 1639807818599000064,
            "path_name": null,
            "syscall_id": "56",
            "pid": "3586",
            "sock_path": null,
            "sock_laddr": null,
            "sock_lport": null,
            "exit": "3958",
            "arg0": "18874385",
            "syscall_name": "clone"
        },
        "[Sat Dec 18 11:41:27.864426 2021] [core:info[pid 3586] AH00096: removed PID file /var/run/apache2/apache2.pid (pid=3586)"
    ],
    [
        "[Sat Dec 18 11:41:27.864472 2021] [mpm_prefork:notice[pid 3586] AH00169: caught SIGTERM, shutting down"
    ],
]
```

Contribution

3. Utilization of Regular Expression to improve search efficacy

Contribution

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.
- This method improves the matching accuracy of log messages during the investigation phase and reduces the runtime search complexity by providing a faster response time.
- This method also reduces dependency explosion by decreasing the number of nodes in the generated UPG

```
fprintf(stderr, "AH00526: Syntax error on line  %d of %s:");
```

```
"AH00526: Syntax error on line  -?[0-9]+ of .*:",
```

Contribution

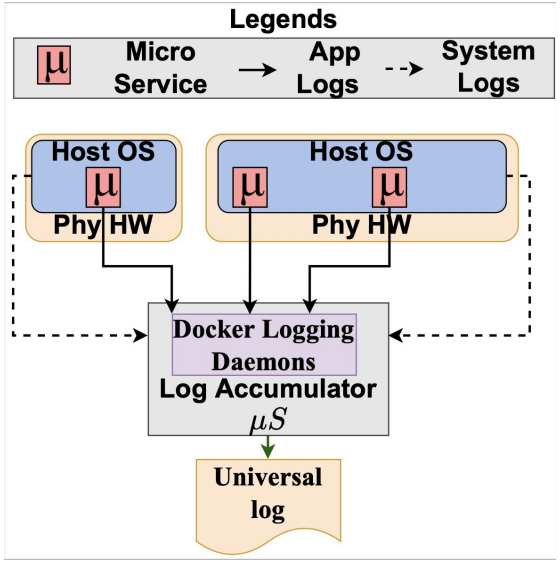
4. Implementation and evaluation

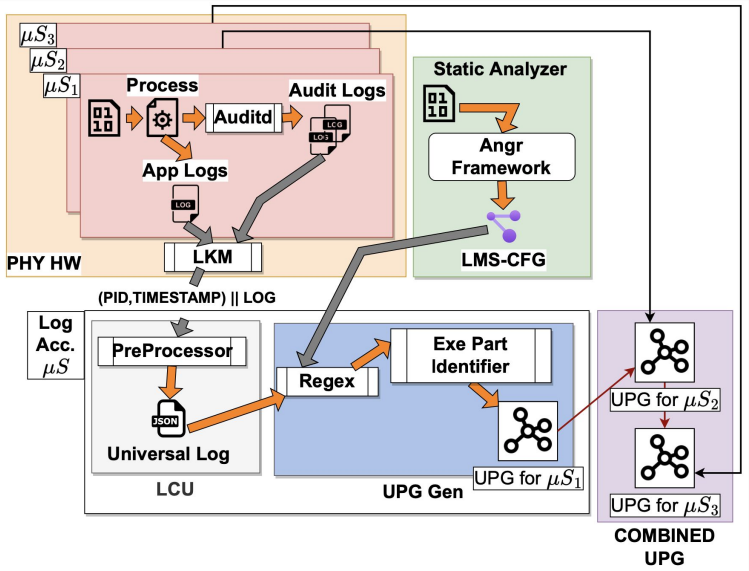
Contribution

4. Implementation and evaluation

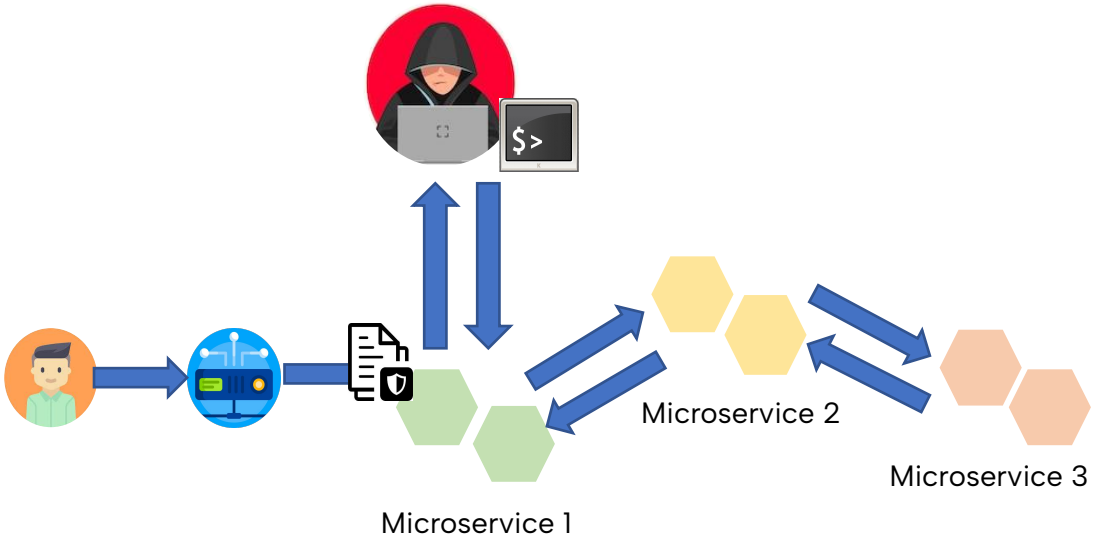
- DisProTrack can be deployed as a microservice on top of the SLC without instrumenting the source code of the applications
- Implementation is open-sourced
- DisProTrack has a minimal memory footprint (~ KB) & responds within 20s-30s.

DisProTrack

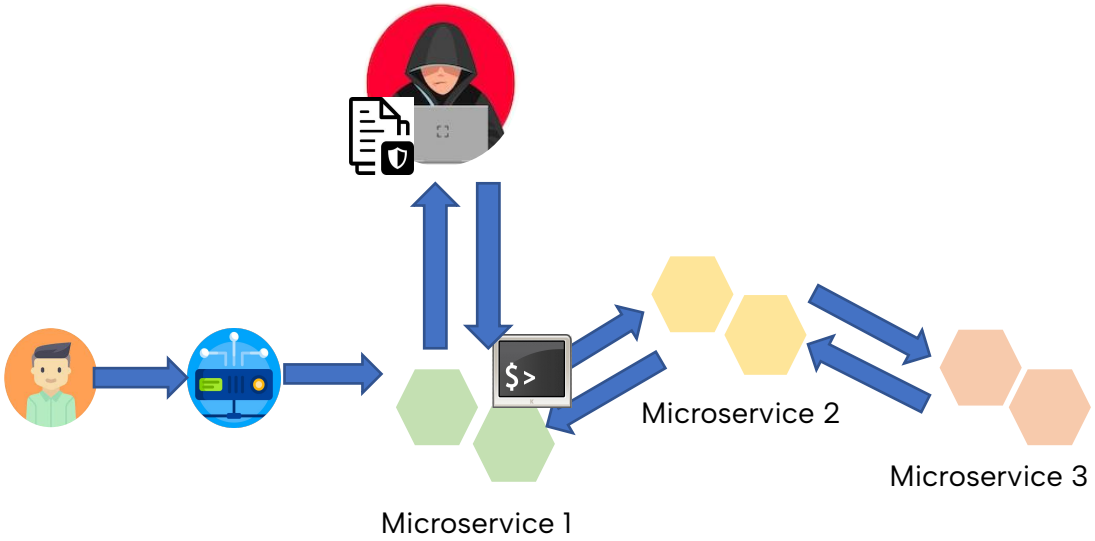




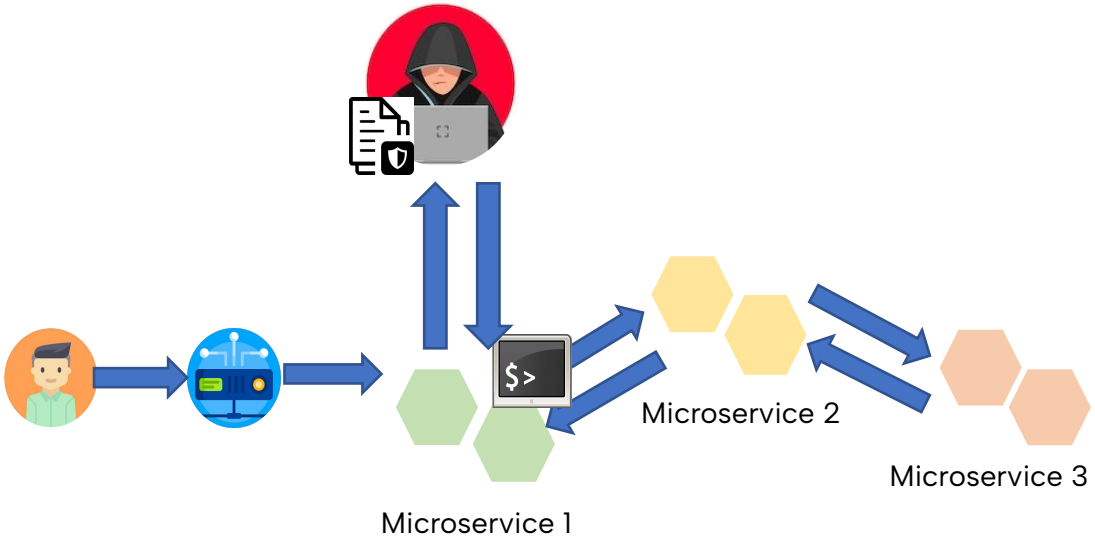
Adversarial Model

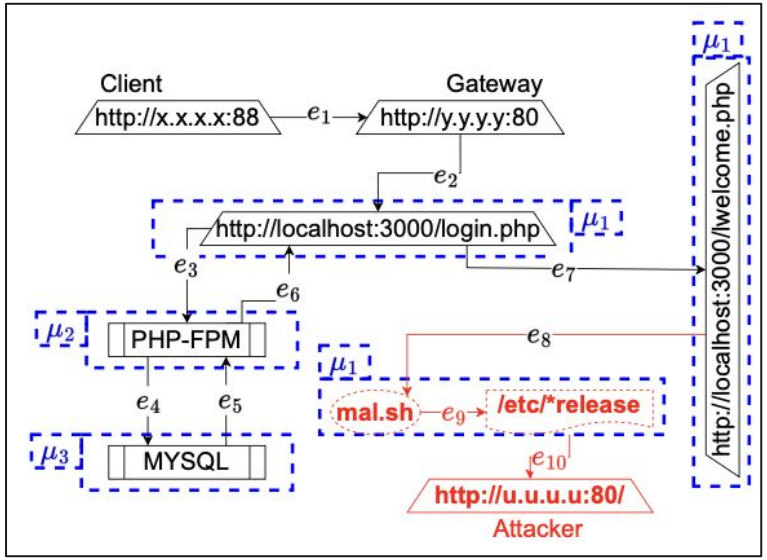


Adversarial Model



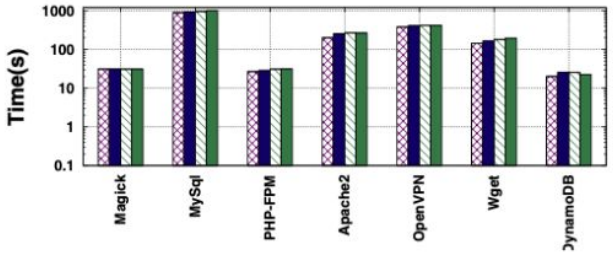
Adversarial Model



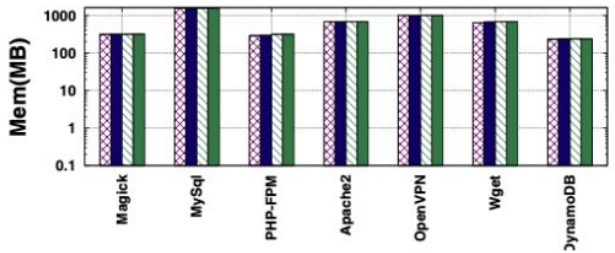


UPG for Confidential Data Theft

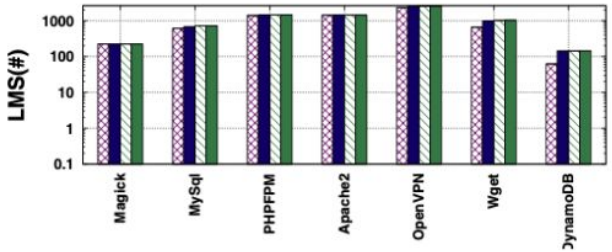
Performance Evaluation - Static Analysis



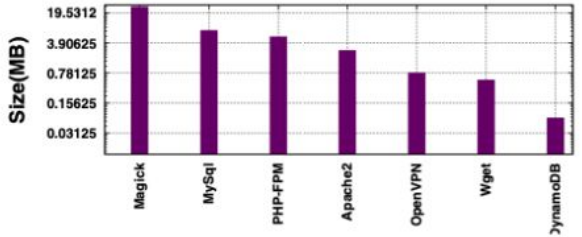
(a) Required Time



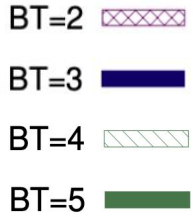
(b) Memory Consumption



(c) LMS Found

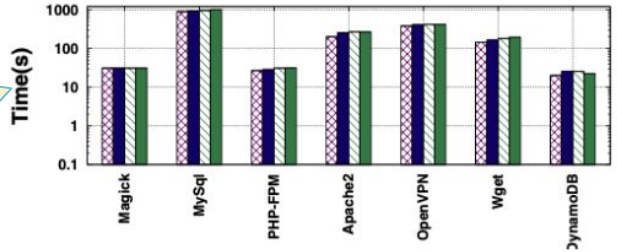


(d) Application Size

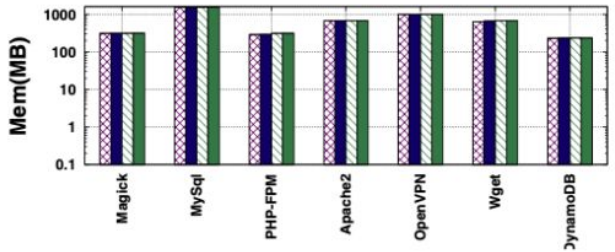


Performance Evaluation - Static Analysis

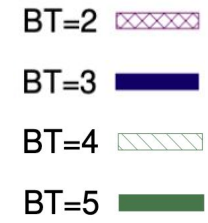
Depends on nature of the program control instructions used by developers



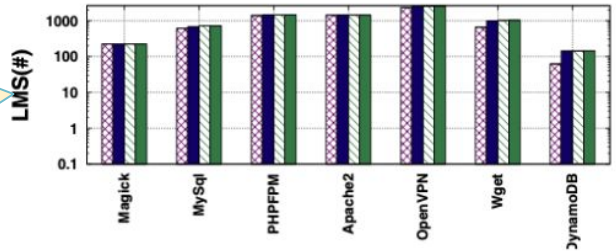
(a) Required Time



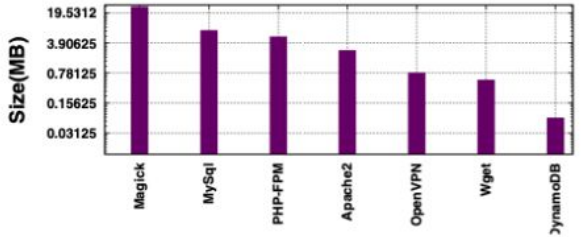
(b) Memory Consumption



Increase in the number of backtraces can identify more number of LMSes

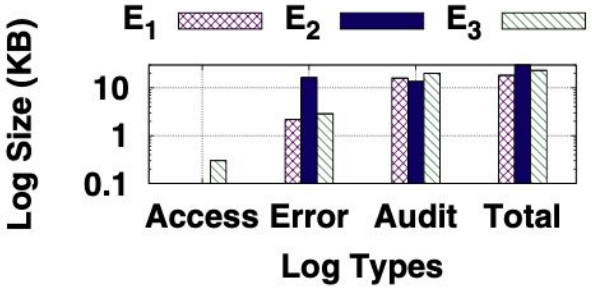


(c) LMS Found

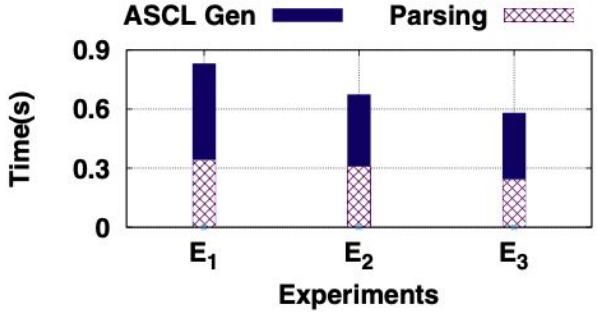


(d) Application Size

Performance Evaluation - Runtime Analysis

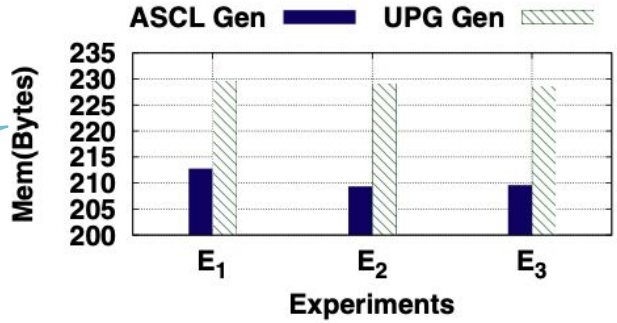


(a) Log File Size



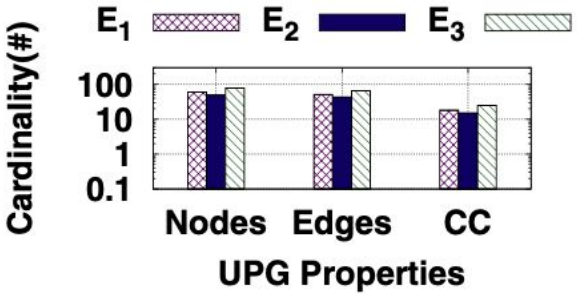
(b) Phase wise execution time

Memory utilization is significantly lower for the modules of the runtime engine

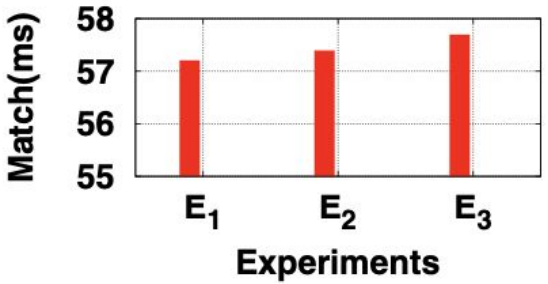


(c) Phase wise Memory utilization

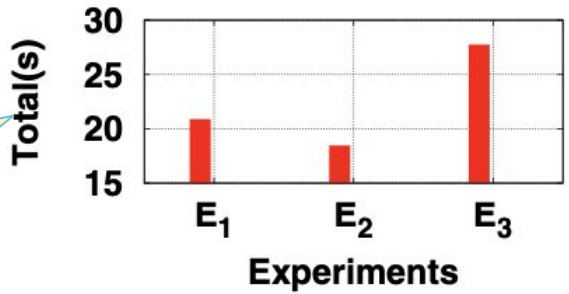
Performance Evaluation - Runtime Analysis



(a) UPG Parameters



(b) RegEx Identification Time



(c) UPG Construction time

Parse time of the log files, merging them to create an ASCL and generation of UPG during provenance builder is directly related to the total amount of logs generated

Conclusion

- DisProTrack is non-invasive causality analysis framework, for provenance tracking over distributed serverless applications.
- It is capable of adversarial attack analysis by identifying the root causes effectively.
- It can be deployed on top of the SLC as a microservice
- We have demonstrated a PoC analysis of DisProTrack which shows its efficiency and efficacy in detecting attack instances for an SLC application.

Thanks

Do you have any questions?

For more details refer to our codebase
git: https://github.com/usatpath01/DisProTrack

DisProTrack

To know more about me and my Research Group
Please Visit:
My Homepage: https://usatpath01.github.io/
UbiNet: https://cse.iitkgp.ac.in/resgrp/ubinet/index.html

My Homepage

UbiNet

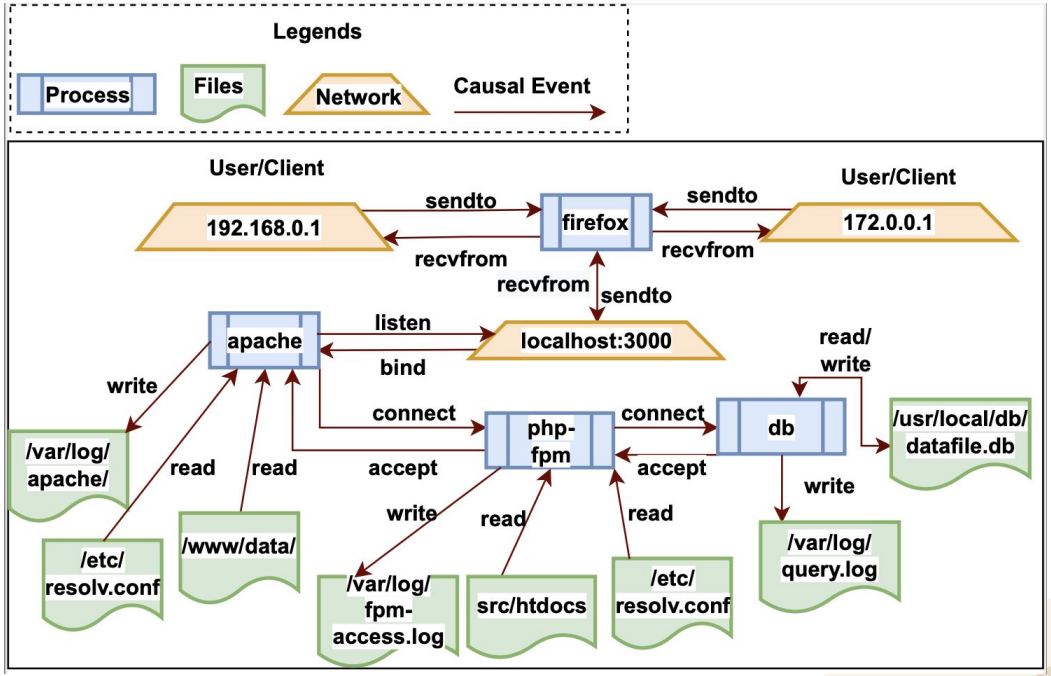
Email: utkalika.satapathy01@gmail.com
utkalika.satapathy01@kgpian.iitkgp.ac.in

Thanks

Do you have any questions?

For more details refer to our paper
Source: https://github.com/usatpath01/DisProTrack

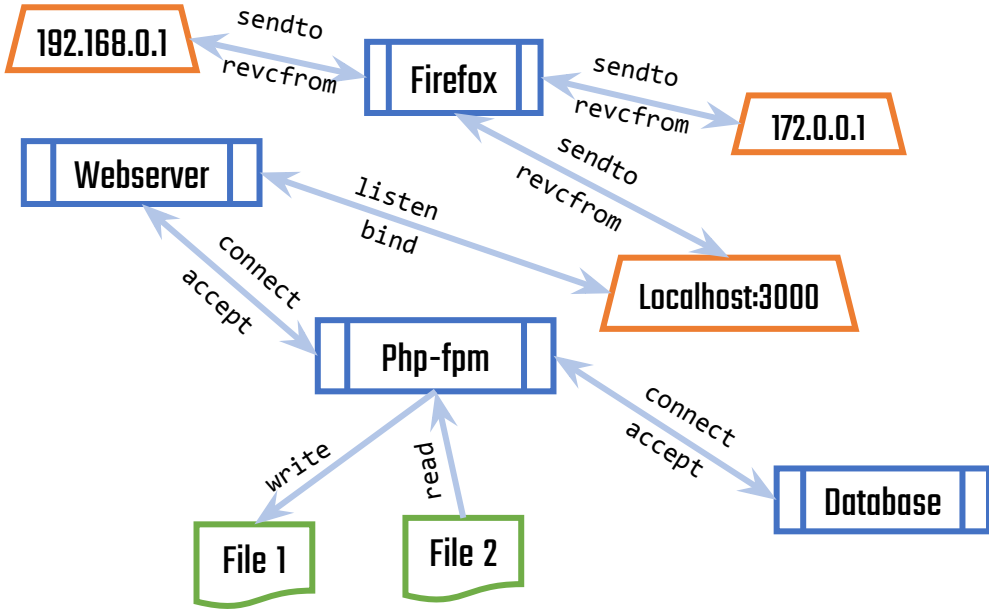
A System Provenance Example



Serverless Computing (SLC)

- Operational expenditure is reduced when service computations are stateless, elastic, and distributed
- Provides abstractions of the underlying infrastructure
- Developer's effort for maintenance and configuration of the environment is reduced
- Good platform when everything is working fine.
- What happens if something is not right?
  - Too distributed

Example of UPG



Motivation

- Existing serverless-specific industry solutions provide limited support for error reporting, execution tracing, and provenance tracking.