PoLPer: Process-Aware Restriction of Over-Privileged Setuid Calls in Legacy Applications

Yuseok Jeon Junghwan Rhee Chung Hwan Kim
Zhichun Li Mathias Payer Byoungyoung Lee Zhenyu Wu

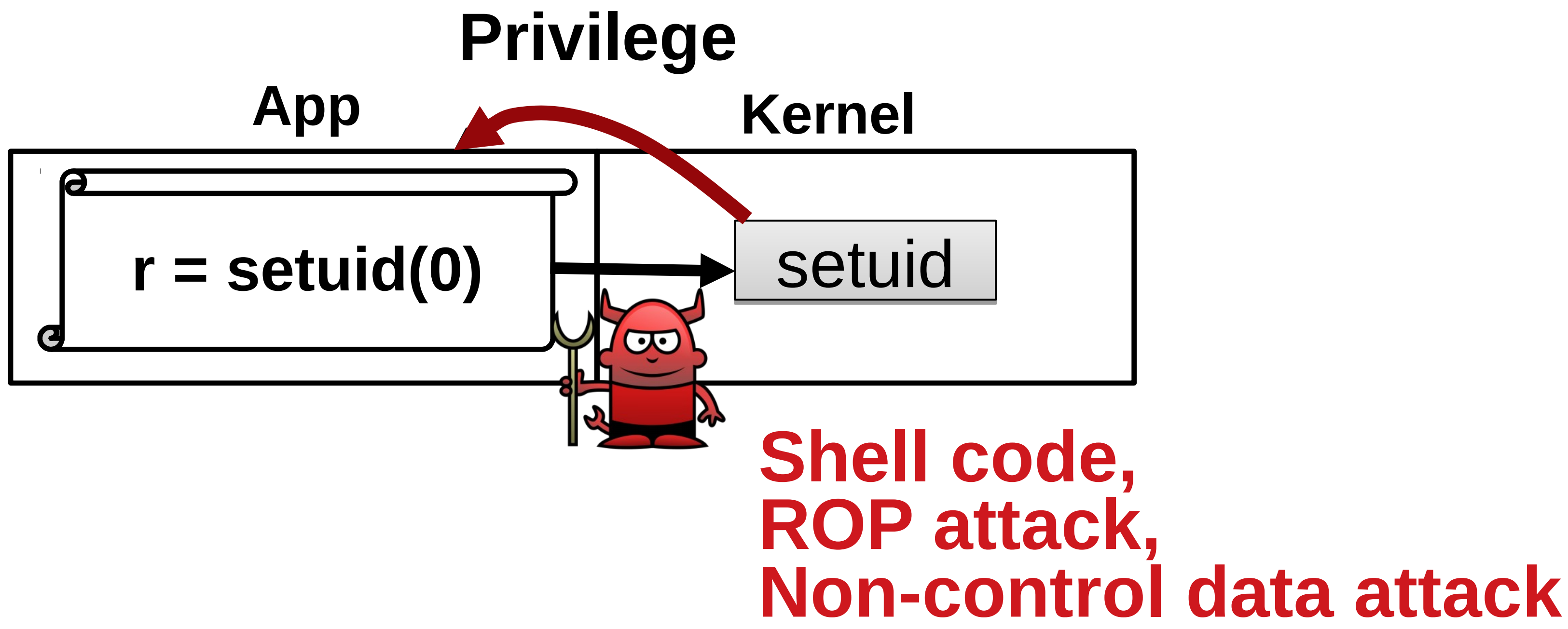
hexhive NEC Laboratories America Relentless passion for innovation PURDUE UNIVERSITY EPFL

Outline

- **Motivation**
- Background
- PoLPer
- Evaluation
- Conclusion

Motivation

- Setud calls
  - Manage privileges
  - Key function for the principle of least privilege (PoLP)
  - Active target of attack



Motivation

- Previous solutions still have limitation

Approaches	Limitations
CFI	Data modification attack detection
DFI	High overhead
System call context check	Over approximated rule (only handle call and data contexts)
Setuid semantic Inconsistency check	Control flow hijacking and data modification attack detection

CFI: control flow integrity
DFI: data flow integrity

Outline

- Motivation
- **Background**
- PoLPer
- Evaluation
- Conclusion

Principle of Least Privilege (PoLP)

- Require minimal privileges
  - Minimized attack surface
  - Limited malware propagation
  - Better stability

- Login programs and daemon launchers
  - Switch their IDs from root to the user ID
  - Setuid calls are used for this change of IDs

Setuid Family System Calls

Use three user IDs as parameters

- Real user ID (real uid, or ruid)
- Effective user ID (effective uid, or euid)
- Saved user ID (saved uid, or suid)

User ID (4 types)	setuid()	seteuid()	setreuid()	setresuid()
Group ID (4 types)	setgid()	setegid()	setregid()	setresgid()

Outline

- Motivation
- Background
- **PoLPer**
- Evaluation
- Conclusion

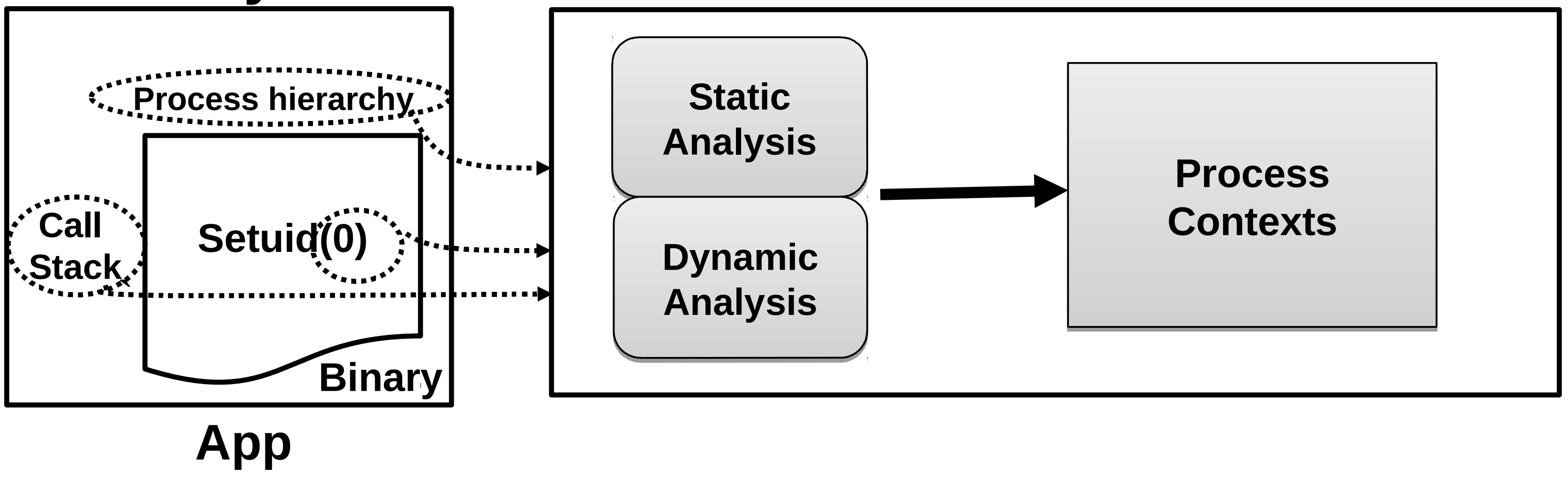
PoLPer

- Focus on process contexts of a setuid call
  - Extracts accurate context information
  - Enforces precise least privileges

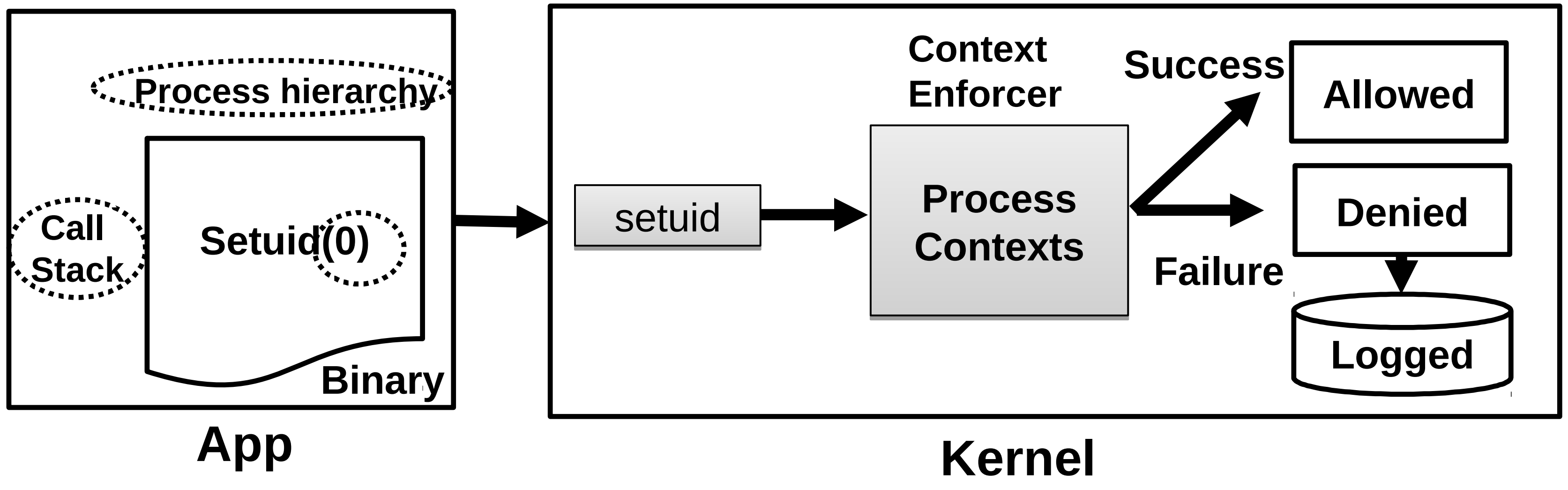
- We propose PoLPer
  - Identifies important process contexts
  - Implements automated context extractor
  - Implements run-time enforcer

PoLPer Overview

Context Analyzer

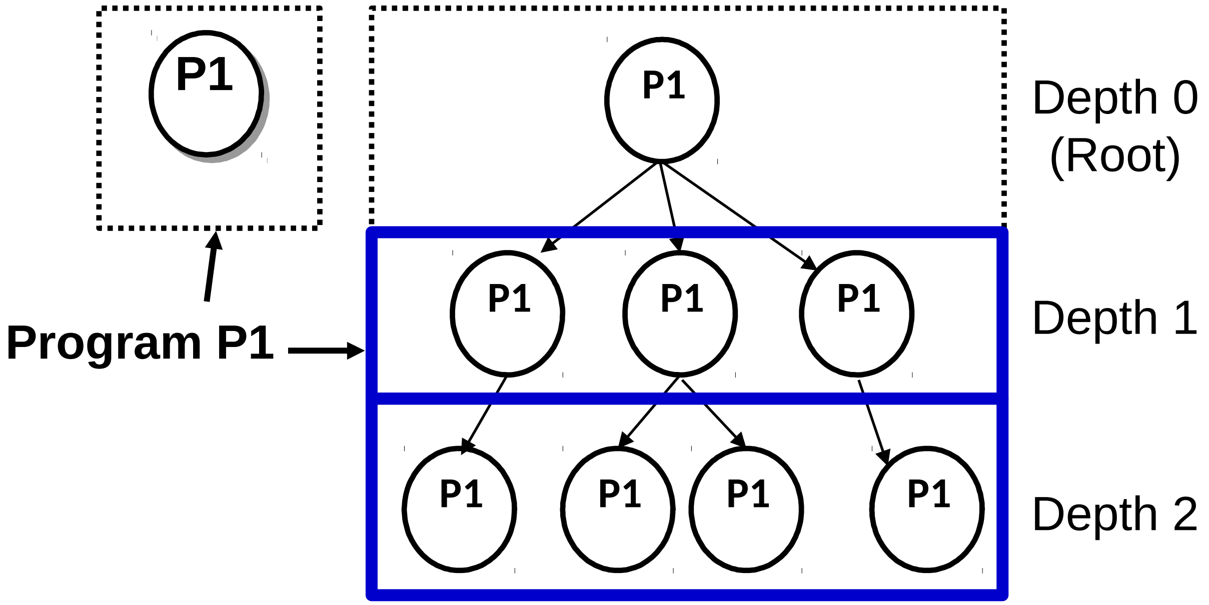


Context Enforcer



Process Hierarchy Context

- Leverage different units of execution to decompose functionalities.



Process Hierarchy Context

SUDO

```
static char *sudo_askpass() {
 static char buf[SUDO_MAX];
    ...
    if ((pid = fork()) == -1) …
    if (pid == 0) {
        ...
        If(setgid(u_details.gid))
         {...}
        ifi(setuid(u_details.uid))
         {...}
        ...
        execl(askpass, ...);
        ...
    }
    ...
(void)sigaction(SIGPIPE,&sa,...);
```

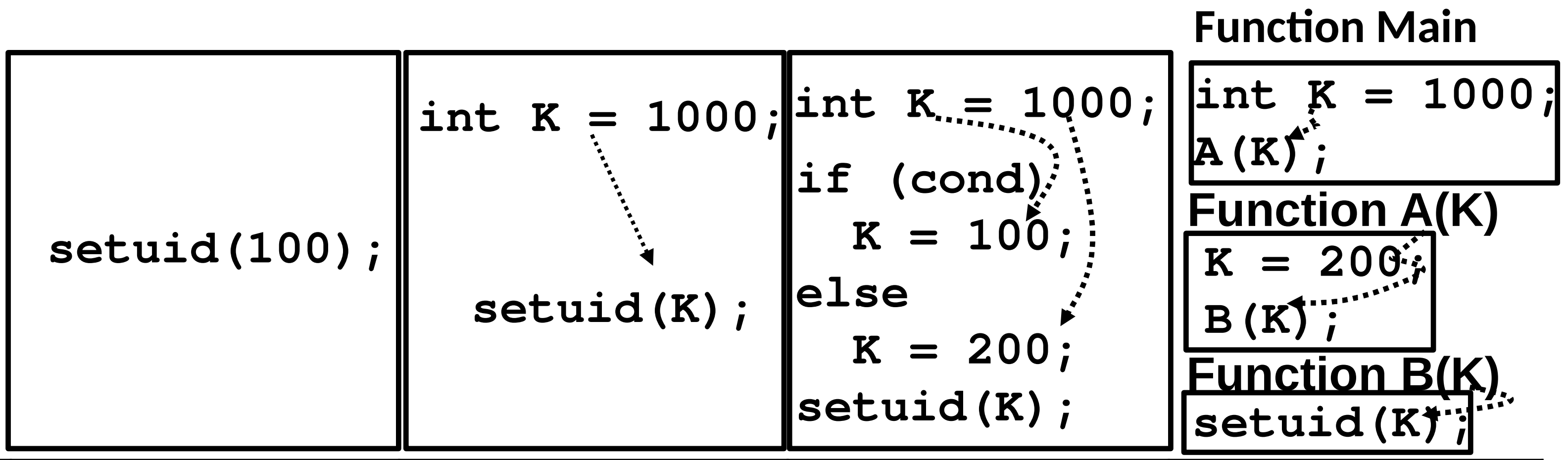
SSHD

```
pid_t subprocess() {
FILE *f;
...
switch ((pid = fork())) {
    case 0:
    ...
    if(setresgid(pw->pw_gid,…))
      {...}
    if(setresuid(pw->pw_uid,…))
      {...}
    ...
    execve(av[0],...);
    _exit(127);
    ...
    default:
    break;
}
```

- Only child process can access setuid calls

Process Data Context

- Need to handle various parameter setting patterns



```
setuid(100);
```

```
int K = 1000;

setuid(K);
```

```
int K = 1000;
if (cond)
  K = 100;
else
  K = 200;
setuid(K);
```

Function Main

```
int K = 1000;
A(K);
```

Function A(K)

```
K = 200;
B(K);
```

Function B(K)

```
setuid(K);
```

Case	(1)	(2)	(3)	(4)
Type	Constant	Variable	Variable	Variable
Value	100	1000	100, 200	200

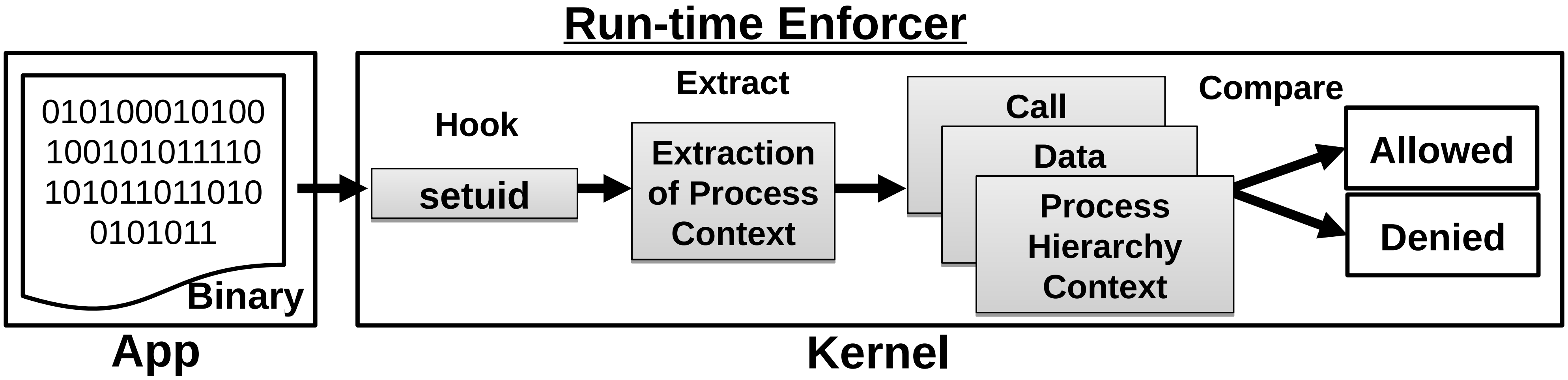
- Use backward data-flow analysis
- Record together with the process hierarchy context

Process Call Context

- Identify code location to identify setuid call
- Dynamic analysis for high accuracy call context
- Record together with the process hierarchy context

Run-time Enforcement

- Use Kprobes, a kernel-based probing mechanism
  - Hooks on the entry points of setuid calls
  - Extracts process hierarchy, data, and call contexts
  - Compares with the profile that was previously extracted



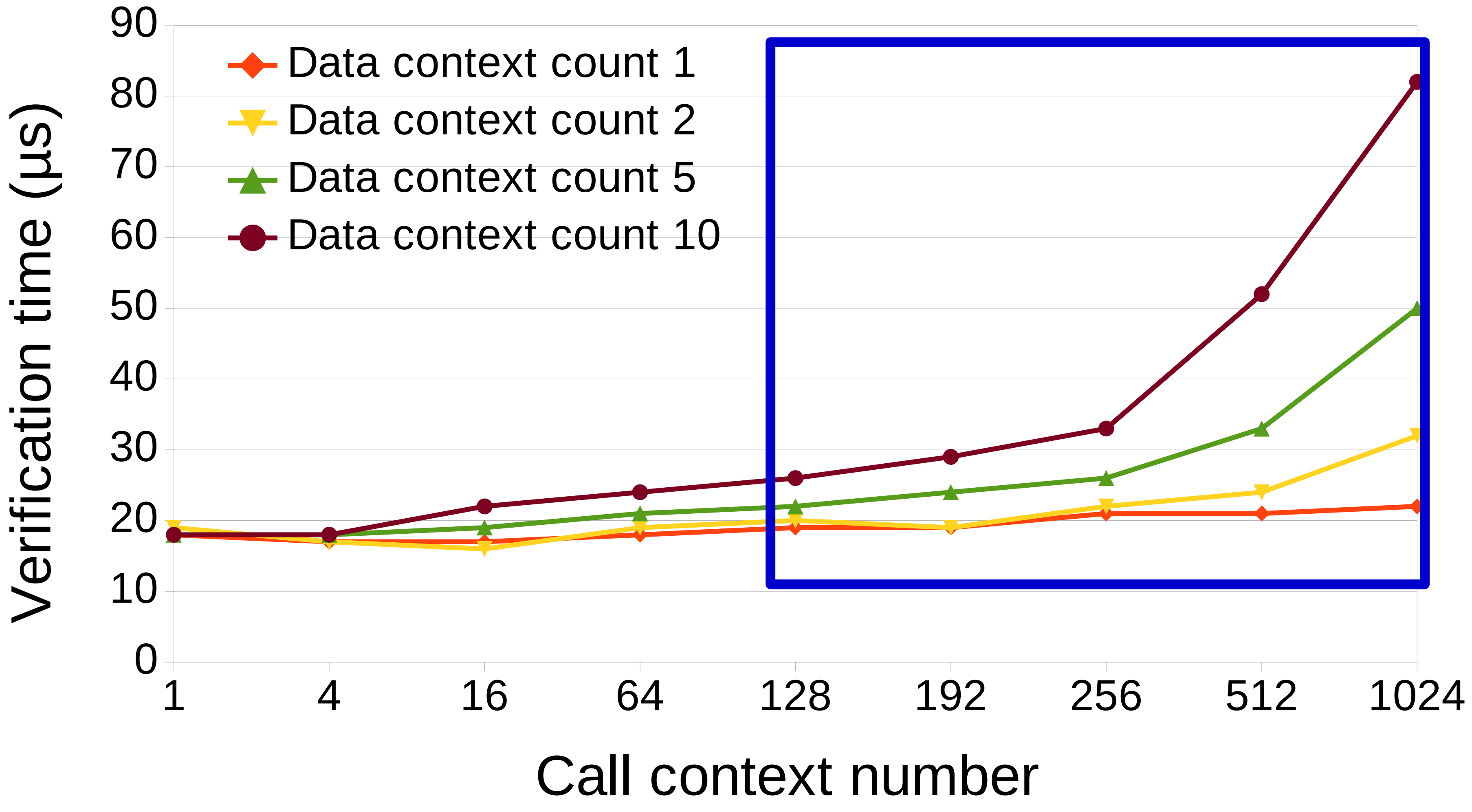
Outline

- Motivation
- Background
- PoLPer
- **Evaluation**
- Conclusion

More Accurate Rule Generation

Programs	Process hierarchy context disabled	Process hierarchy context enabled	Rule cut (%)
Ping	1	1	0
Sudo	352	196	44
Xterm	576	296	49
Cron	2	2	0
Telnet	4	4	0
Telnet-Login	6	3	50
Login	4	2	50
SSH & SCP	182	88	52
WireShark	2	2	0
Apache	2	2	0
Nginx	2	2	0

Micro-benchmark



End-to-end Benchmarks

- Show near zero overhead

Programs	Base (s)	PoLPer (s)	Setuid call (#)	Overhead (%)
Ping	9.0019	9.0039	1	0.02
Nginx	11.522	11.539	0	0.14
Apache	18.250	18.286	0	0.1
Telnet	1.001	1.004	6	0.29
SCP	0.1656	0.1665	28	0.54

Real-world Exploits

Exploit Pattern	Vul. Program	Exploit Name (EDB)	Setuid Syscall Exploited	Detected		
				PoLPer	CFI	NCI
Modify Setuid Parameters	Sudo	(N/A)	setuid	√	X	√
	Wu_ftpd	(N/A)	seteuid	√	X	√
Run setuid call to creat a root shell	Overlayfs	37292-2015	setresuid, setresgid	√	√	X
		39230-2016	setresuid	√	√	X
	Glibc	209-2000	setuid, setgid	√	√	X
	Mkdir	20554-2001	setuid, setgid	√	√	X
	KApplication	19981-2000	setuid, setregid	√	√	X
	Suid_dumpable	2006-2006	setuid, setgid	√	√	X
	Execve/ptrace	20720-2001	setuid	√	√	X
	Splitvt	20013-2000	setuid	√	√	X
	OpernMovieeditor	2338-2006	setuid,setgid	√	√	X

CFI: control flow integrity
NCI: non-control data integrity

Case Study: Sudo

```
struct  user {
  uid_t uid;
  ...
};

struct user ud;
ud.uid = getuid();

//in sudo_debug()
vfprintf (...);

//in sudo_askpass()
setuid (ud.uid);
```

Sudo code example

Depth	1			
Priv. Op.	setuid			
Parameter	(Profile) uid = getuid()			
	(exploit) 0			
Call Stack	#	Offset	File	Function
	21	0x32 + 0xb75f7b44	../libc.so.6	-
	20	0x8053080	../bin/sudo	sudo_askpass
	...			
	1	0x8049dd1	../bin/sudo	-

Outline

- Motivation
- Background
- PoLPer
- Evaluation
- **Conclusion**

Conclusion

- Extracts only the required contexts of setuid calls
- Prevents setuid exploits with negligible overhead
- Enforces PoLP using a combination of different process contexts

THANK YOU!

Q&A