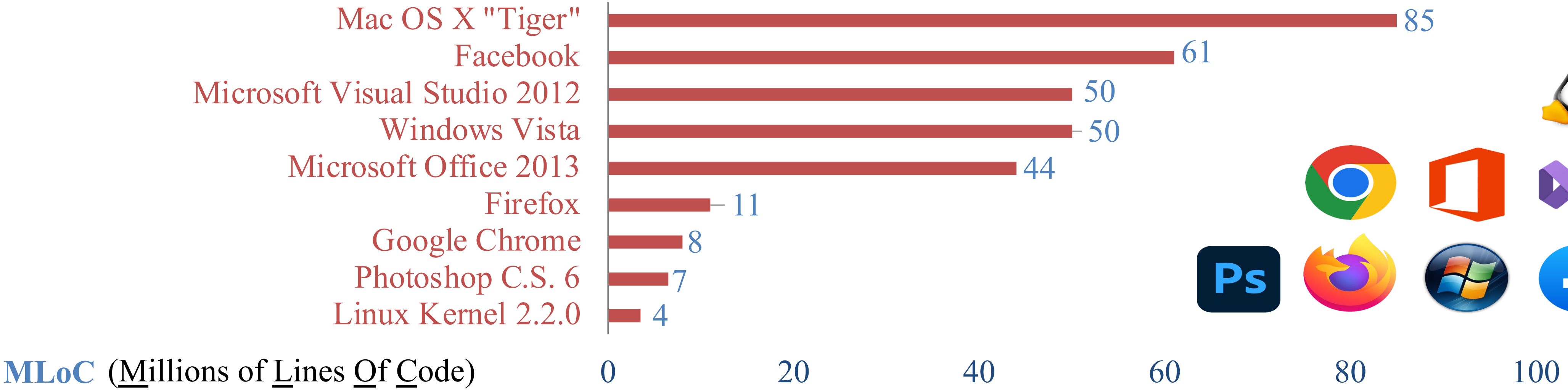
Place Your Locks Well: Understanding and Detecting Lock Misuse Bugs

Yuandao Cai, Peisen Yao, Chengfeng Ye, Charles Zhang
The Hong Kong University of Science and Technology

Increasingly Complex Modern Software

- Massive codebases and high concurrency



- Poor software quality costs the US economy **$2.41 trillion** annually

Cost of Poor Software Quality Report 2022 Update

it-cisq.org

CISQ

Consortium for Information & Software Quality™

Ooops, your files have been encrypted!

What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

Payment will be raised on
5/16/2017 00:47:55
Time Left
02:23:57:37

Your files will be lost on
5/20/2017 00:47:55
Time Left
06:23:57:37

About bitcoin
How to buy bitcoins?
Contact Us

bitcoin ACCEPTED HERE

Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

Check Payment
Decrypt

DIRTY COW

Synchronization Primitives against Harmful Races

- Concurrency bugs are hard-to-avoid and extremely harmful!

- Synchronization primitives are in place to synchronize concurrent code
    - preventing various concurrency bugs and vulnerabilities

- Existing works focus on **concurrency bug detection** (insufficient synchronization)
    - data races[AI Thokair POPL’23] [chabbi et.al., PLDI’22]…
    - concurrency memory corruption bugs [Yuan et.al., Security’23] [Cai et.al., PLDI’21]…
    - concurrency typestate bugs [ASPLOS’11]…

- Our work focuses on the misuses of synchronization APIs themselves
    - currently focusing on locks
    - also causing serious reliability and security issues

Research Goal and Contributions

1. Understanding the common misuses of locks
    - through a CVE-ID-based empirical study

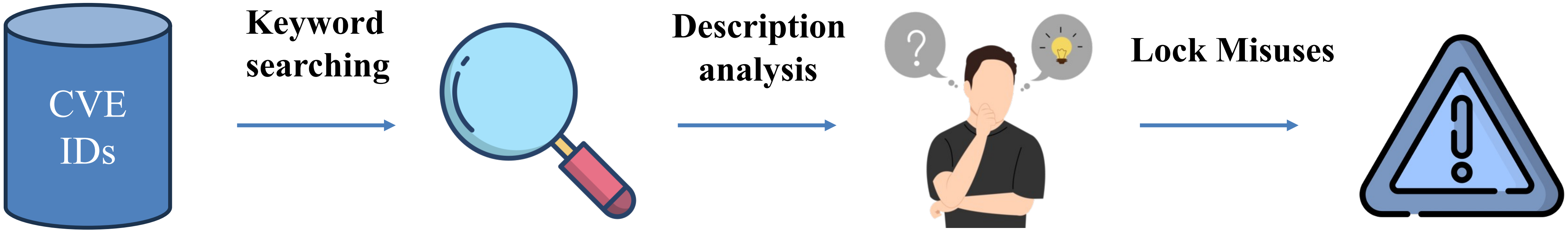
2. Designing techniques to detect the lock misuses

3. Evaluating and advancing the state-of-the-art bug-finding tools

An Empirical Study: Setup

- Locks are common synchronization primitives
  - with explicit disciplines for initialization, use, and destruction.
- Study Question 1: What are **the common lock misuses**?
- Study Question 2: What are **the common causes** of those lock misuses?

- **Study Dataset:** 32 CVE IDs assigned between 2010-2021
  - search keywords: e.g., mutex, lock
  - manual validation for CVE ID description



CVE IDs

Keyword searching

Description analysis

Lock Misuses

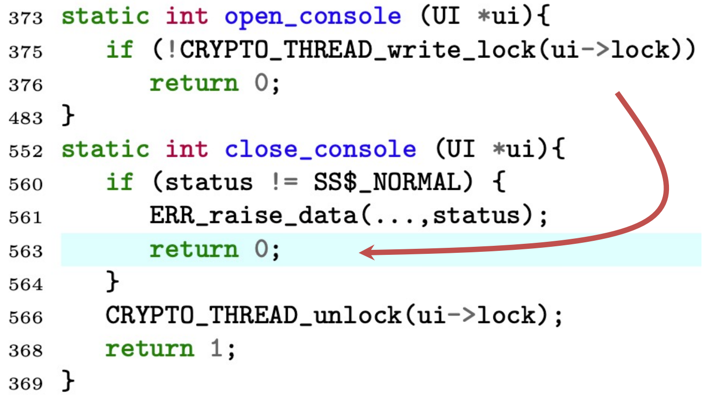
An Empirical Study: Finding I

1. **Identifying** five general locking discipline violations
    - under both sequential and concurrent circumstances
    - covering a single thread and multiple threads

2. **Defining** the bug patterns by revealing their characteristics

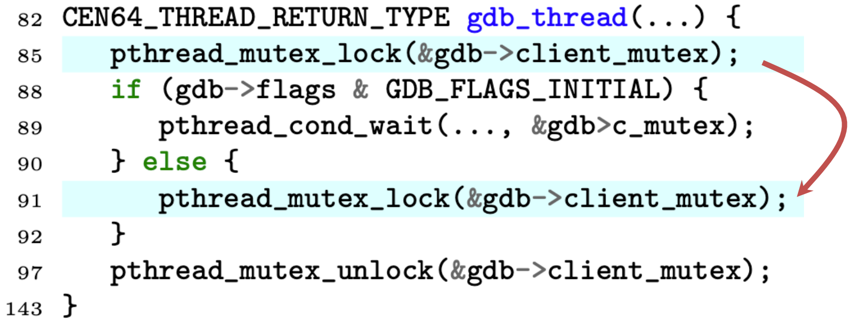
No.	Misuse Pattern	Bug Description	Concurrency
①	Missing lock releases	A lock is not released after its effective lifetime.	
②	Double locking	A lock is acquired twice.	
③	Using uninitialized locks	A lock is not initialized before using it. A concurrency error occurs when the lock is initialized non-deterministically.	✓
④	Releasing unacquired locks	A lock is released without acquiring it first. A concurrency error occurs when there is another thread holding the lock.	✓
⑤	Cyclic lock acquisitions	Different locks are not acquired in the same order. A concurrency error occurs when each thread in a set waits for the other to release a lock.	✓

An Empirical Study: Five General Lock Misuses



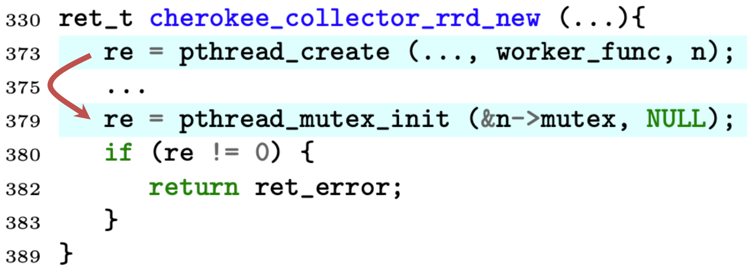
```
static int open_console (UI *ui){
    if (!CRYPTO_THREAD_write_lock(ui->lock))
        return 0;
}
static int close_console (UI *ui){
    if (status != SS$_NORMAL) {
        ERR_raise_data(...,status);
        return 0;
    }
    CRYPTO_THREAD_unlock(ui->lock);
    return 1;
}
```

(1) Missing lock releases (OpenSSL)



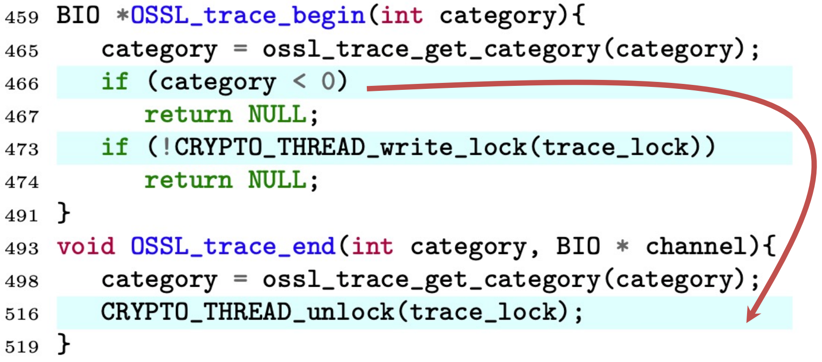
```
CEN64_THREAD_RETURN_TYPE gdb_thread(...) {
    pthread_mutex_lock(&gdb->client_mutex);
    if (gdb->flags & GDB_FLAGS_INITIAL) {
        pthread_cond_wait(..., &gdb>c_mutex);
    } else {
        pthread_mutex_lock(&gdb->client_mutex);
    }
    pthread_mutex_unlock(&gdb->client_mutex);
}
```

(2) Double locking (Cen64)



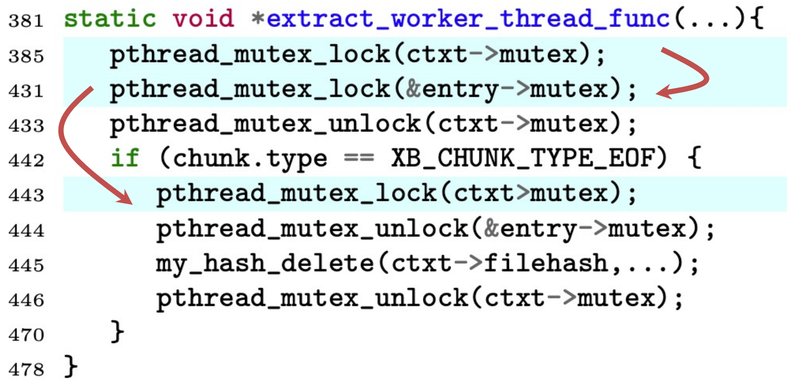
```
ret_t cherokee_collector_rrd_new (...){
    re = pthread_create (..., worker_func, n);
    ...
    re = pthread_mutex_init (&n->mutex, NULL);
    if (re != 0) {
        return ret_error;
    }
}
```

(3) Using uninitialized locks (Cherokee)



```
BIO *OSSL_trace_begin(int category){
    category = ossl_trace_get_category(category);
    if (category < 0)
        return NULL;
    if (!CRYPTO_THREAD_write_lock(trace_lock))
        return NULL;
}
void OSSL_trace_end(int category, BIO * channel){
    category = ossl_trace_get_category(category);
    CRYPTO_THREAD_unlock(trace_lock);
}
```

(4) Releasing unacquired locks (OpenSSL)



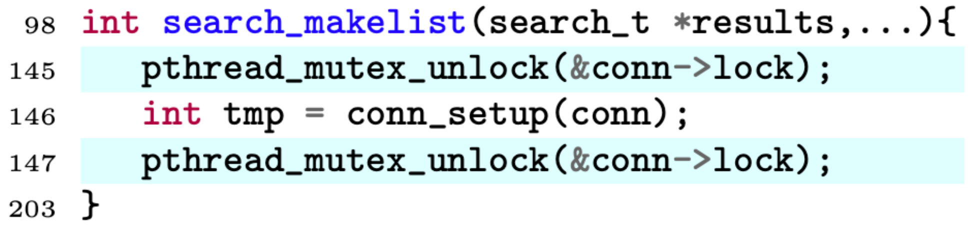
```
static void *extract_worker_thread_func(...){
    pthread_mutex_lock(ctxt->mutex);
    pthread_mutex_lock(&entry->mutex);
    pthread_mutex_unlock(ctxt->mutex);
    if (chunk.type == XB_CHUNK_TYPE_EOF) {
        pthread_mutex_lock(ctxt>mutex);
        pthread_mutex_unlock(&entry->mutex);
        my_hash_delete(ctxt->filehash,...);
        pthread_mutex_unlock(ctxt->mutex);
    }
}
```

(5) Cyclic lock acquisitions (MariaDB)

An Empirical Study: Finding II

- Wreaking severe havoc by triggering lock misuses
    - **denial-of-service with system hang** (concurrent cyclic acquisitions, double locking)
        - CVE-2013-4553, CVE-2014-8131, CVE-2019-14763, CVE-2021-41213,…
    - **memory exhaustion with memory leak** (missing lock releases)
        - CVE-2004-2650, CVE-2018-14660, CVE-2020-12658,…
    - **memory corruption; system crash** (releasing unacquired locks, using uninitialized locks)
        - CVE-2014-1453, CVE-2015-8767, CVE-2017-6353, CVE-2020-10573,…
    - **even privilege escalation and other unidentified issues**
        - CVE-2010-4210, CVE2014-9748, …

- Relating to other security bugs
    - **atomicity violations** (CVE-2020-10573)
    - **use-after-free** (CVE-2019-14034)
    - **double free** (CVE-2017-6353)



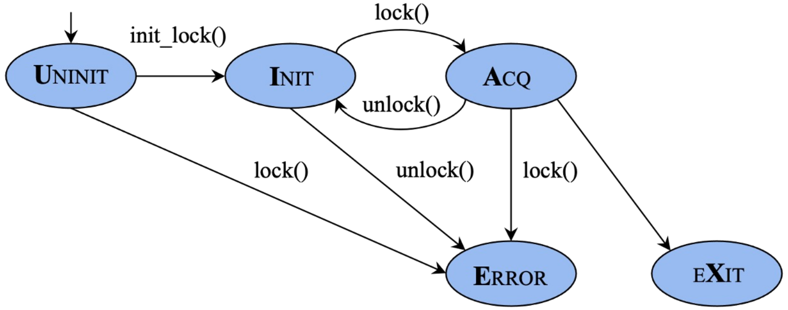
```
int search_makelist(search_t *results,...){
  pthread_mutex_unlock(&conn->lock);
  int tmp = conn_setup(conn);
  pthread_mutex_unlock(&conn->lock);
}
```

Releasing unacquired locks that leads to **atomicity violations** (Axel)

Detecting the Five Lock Misuses with Lockpick

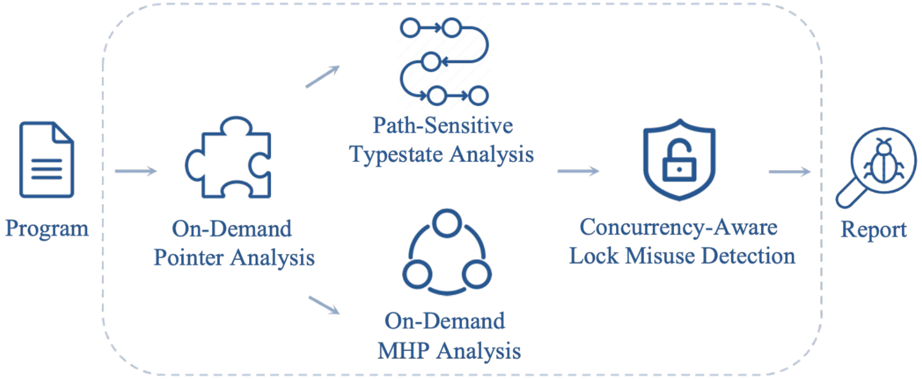
- **Lock misuse formulation:** characterizing lock misuses with **a finite-state machine (FSM)**

1. Model the states of lock objects using typestates
2. Capture the state transitions of lock objects with a new FSM
3. Capture the lock misuses by tracking the state transitions



- **Lock misuse detection:** detecting lock misuses with **several customized techniques**

1. Path-sensitively track the typestates of locks
2. Reason about the MHP relations of statements
3. Flag the lock misuses based on typestate violations



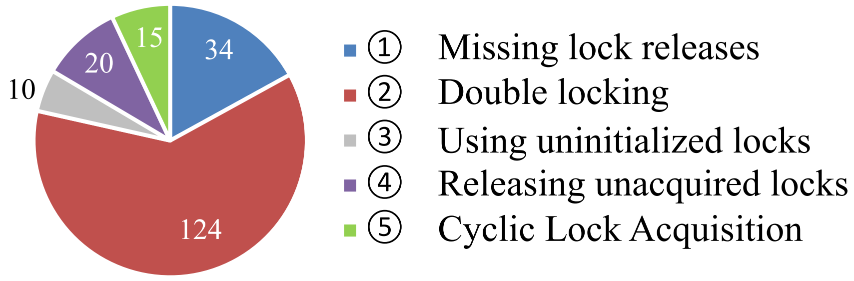
Implementation and Experiment Setup

- Lockpick is built upon the LLVM infrastructure and the Z3 SMT solver
  - a **soundy** implementation to reach both high efficiency and precision
    - unrolling loops twice, ignoring inline assembly, pointer arithmetic
  - a value-flow-based pointer analysis
    - on-demand flow-, context-sensitive pointer analysis
  - path conditions are encoded as first-order logic formulae over bit-vectors

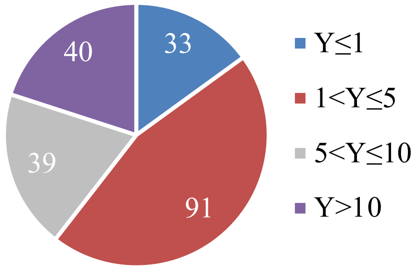
- Question 1: How **effective and practical** is Lockpick at uncovering lock misuses in mature open-source software systems?
- Question 2: How does Lockpick perform **compared to the state-of-the-art tools**?

(1) Highlights: Effectiveness on Bug Finding

- Finding **203 developer-confirmed bugs** in over 80 well-checked software programs
  - **184** of them have been fixed (at the time of publication)
  - finding various kinds of bugs
  - hiding for an average of **7.4** years



The distributions of bug type

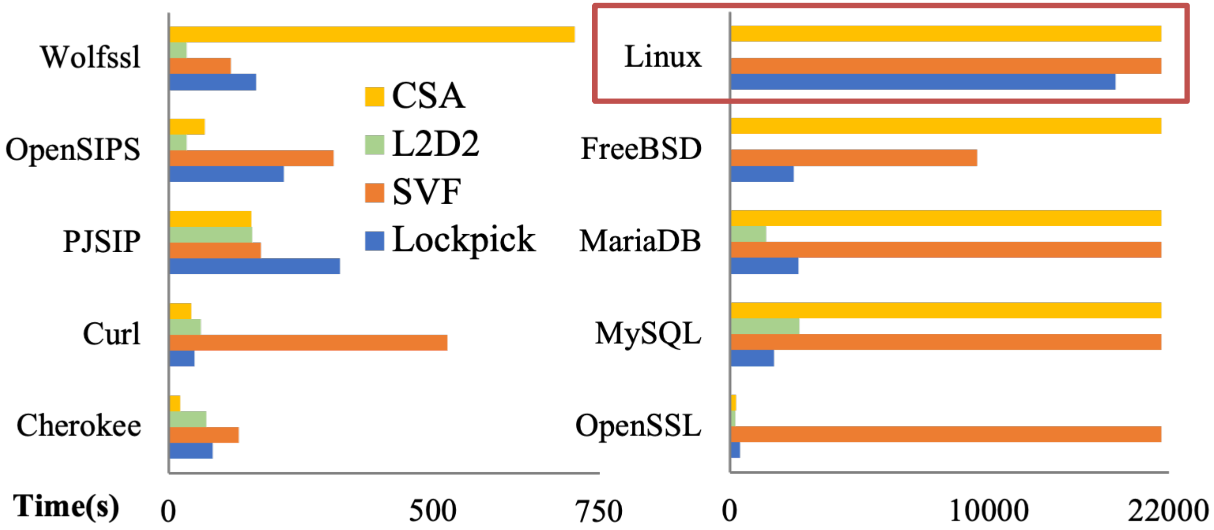


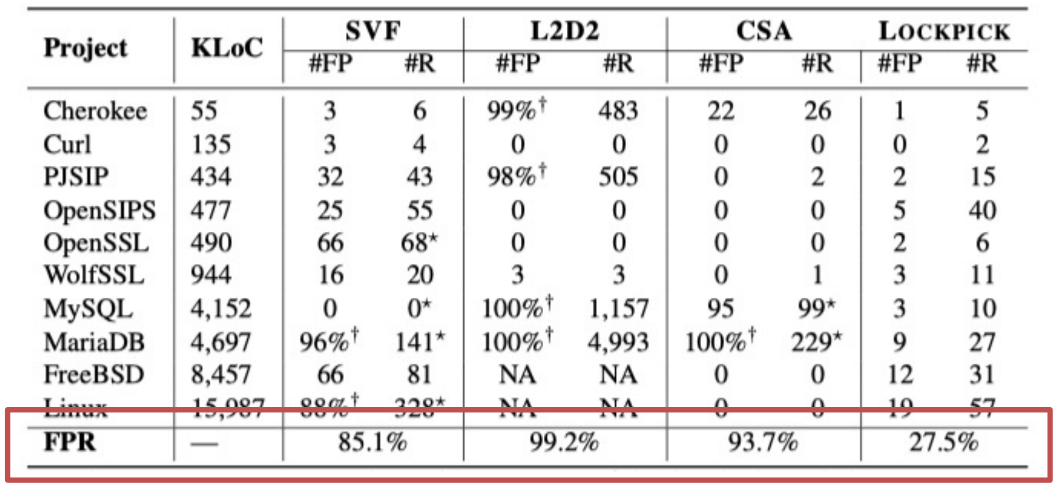
The distributions of hidden time (**Y**ear)

- **16 CVE IDs** have been assigned for multiple bugs with high security impacts
  - CVE-2021-41141, CVE-2021-43429, CVE-2022-31621, CVE-2022-31624, CVE-2022-31623, CVE-2022-31622, CVE-2022-30027, CVE-2022-37869, CVE-2022-37868, CVE-2022-38791, CVE-2022-37874, CVE-2022-37875, CVE-2022-37876, CVE-2022-37871, CVE-2022-37872…

(2) Highlights: Advancement over Previous Tools

- **Baselines:** SVF, L2D2 (built on Infer), Clang Static Analyzer
- **Benchmarks:** ten popular software programs with **35.8 MLoC**
- **Efficiency:** being able to analyze big programs like Linux kernel in about five hours
- **Precision:** embracing better precision than other tools
- **Recall:** being able to discover **26 past CVE IDs** in C/C++ programs (2010-2021)
    - other tools cannot reach





Project	KLoC	SVF		L2D2		CSA		LOCKPICK	
		#FP	#R	#FP	#R	#FP	#R	#FP	#R
Cherokee	55	3	6	99%†	483	22	26	1	5
Curl	135	3	4	0	0	0	0	0	2
PJSIP	434	32	43	98%†	505	0	2	2	15
OpenSIPS	477	25	55	0	0	0	0	5	40
OpenSSL	490	66	68*	0	0	0	0	2	6
WolfSSL	944	16	20	3	3	0	1	3	11
MySQL	4,152	0	0*	100%†	1,157	95	99*	3	10
MariaDB	4,697	96%†	141*	100%†	4,993	100%†	229*	9	27
FreeBSD	8,457	66	81	NA	NA	0	0	12	31
Linux	15,987	88%†	328*	NA	NA	0	0	19	57
FPR	—	85.1%		99.2%		93.7%		27.5%	

Thank you for your listening!

Questions & Answers

More details can be found in our paper:
https://www.usenix.org/system/files/sec23fall-prepub-298_cai-yuandao.pdf
Bug and CVE ID lists can be found:
https://drive.google.com/file/d/1HY7PydeDga-850ZOn3YPACnX7hRws8DG/view