The S2E Platform

From a research prototype to a commercial product

Vitaly Chipounov
Cyberhaven, Inc

https://s2e.systems

Ready-for-use docker image, demos, tutorials, source code, documentation

Vitaly Chipounov

- 2008: started PhD at EPFL, Switzerland - DSLAB, George Candea
- Reverse engineering device drivers by tracing them in QEMU
- Got a pre-release version of KLEE in 2008 => combined it with QEMU
- 2010: reverse engineering + automated testing of device drivers
- 2011: released the S2E platform
- 2014: graduated with PhD, co-founded Cyberhaven
- 2014-2016: malware scanner for office documents
- 2014-2016: finalists at the DARPA CyberGrandChallenge
- Released all of our S2E work to the public
- Tracing dataflows in enterprises for insider threat prevention

Extensible
Write your own tools

On real OSes, with
real apps, libraries, drivers

S2E is a platform for in-vivo multi-path analysis of software systems

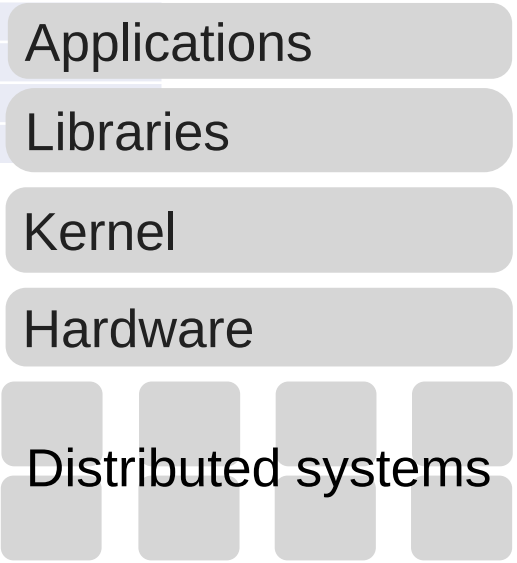
Symbolic execution
Concolic execution
State merging…

Bug finding
Verification
Testing
Security checking...

Pretty much anything that
runs on computers

Automatic firmware emulation	USENIX SEC'21
Finding buggy configurations that cause slowdowns	ODSI'20
Binary lifting and recompilation	EUROSYS'20
Exploitation of tarpit vulnerabilities in malware	SP'19
Exploiting uninitialized memory in the Linux kernel	NDSS'17
Symbolic fault injection in USB drivers	WOOT'17
Bug finding in Windows system components	USENIX'17
Bug finding in the BIOS	WOOT'15
Verifying software router dataplanes	NSDI'14
Testing device firmware	NDSS'14
Symbolic execution for interpreted languages	ASPLOS'13
Finding trojan message vulnerabilities in distributed systems	ASPLOS'13
Testing file systems	EUROSYS'12
Bug finding in Linux device drivers	OSDI'12
Testing distributed systems	WRIPE'12
Bug finding in Windows device drivers	USENIX'11
Reverse engineering device drivers	EUROSYS'10

x86
ARM



Outline

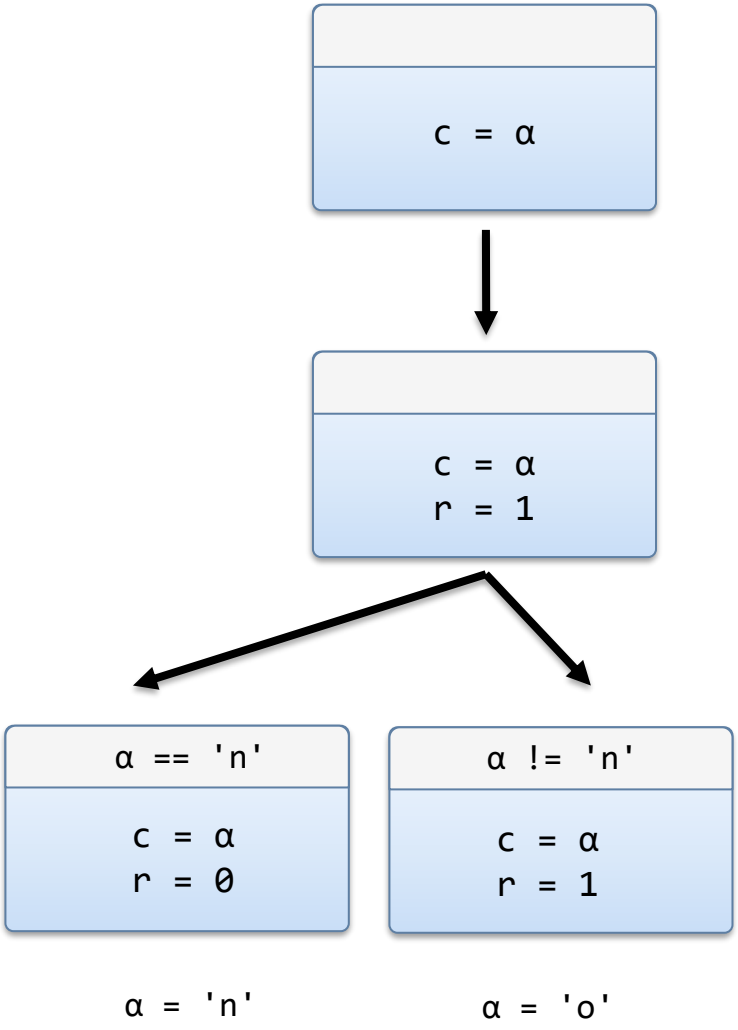
- How does S2E work?
  Scaling symbolic execution to entire VMs

- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Enterprise insider threat detection

- Future of S2E
  Making it 10-100 times faster

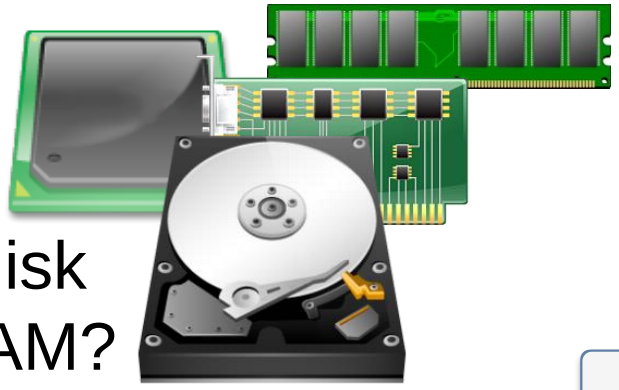
Dynamic Symbolic Execution

```
int func(char c) {
  int r = 1;

  if (c == 'n') {
     r = 0;
  }

  return r;
}
```

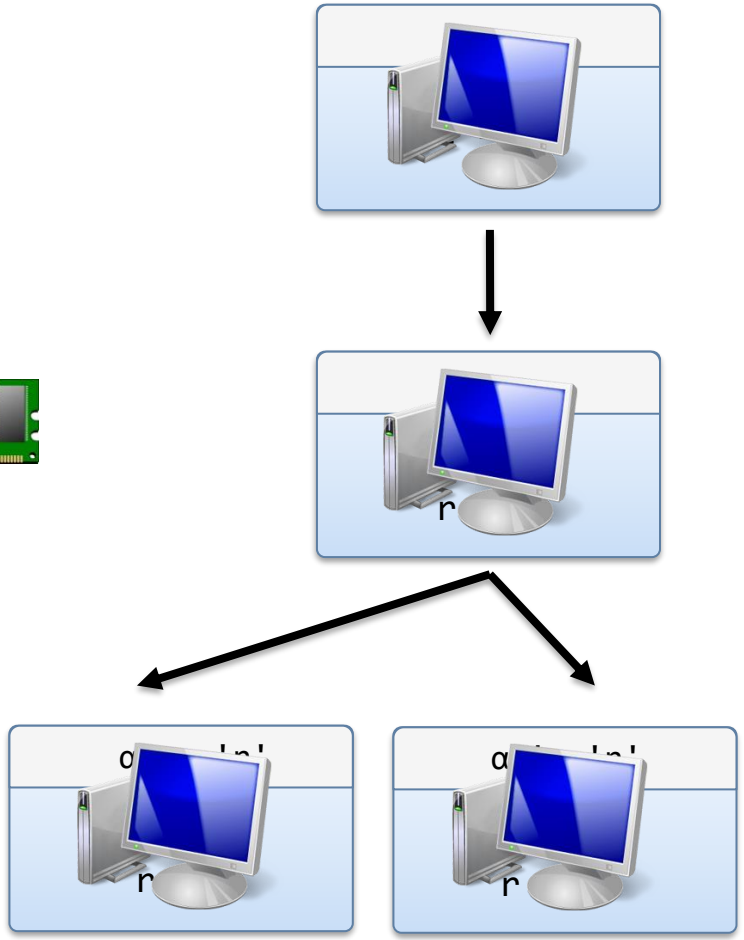


Dynamic Symbolic Execution

```
int func(char c) {
  int r = 1;

  if (c == 'n') {
     r = 0;
  }

  return r;
}
```

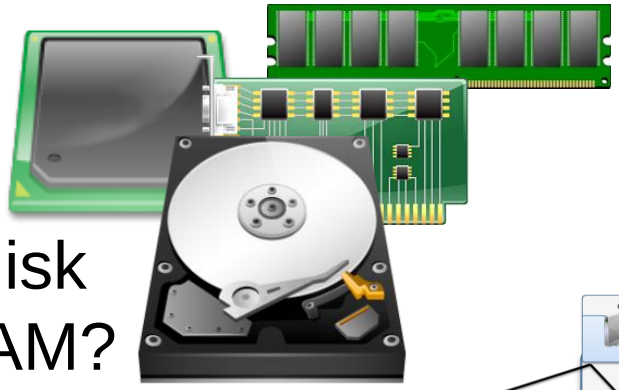


30 GB disk
4 GB RAM?

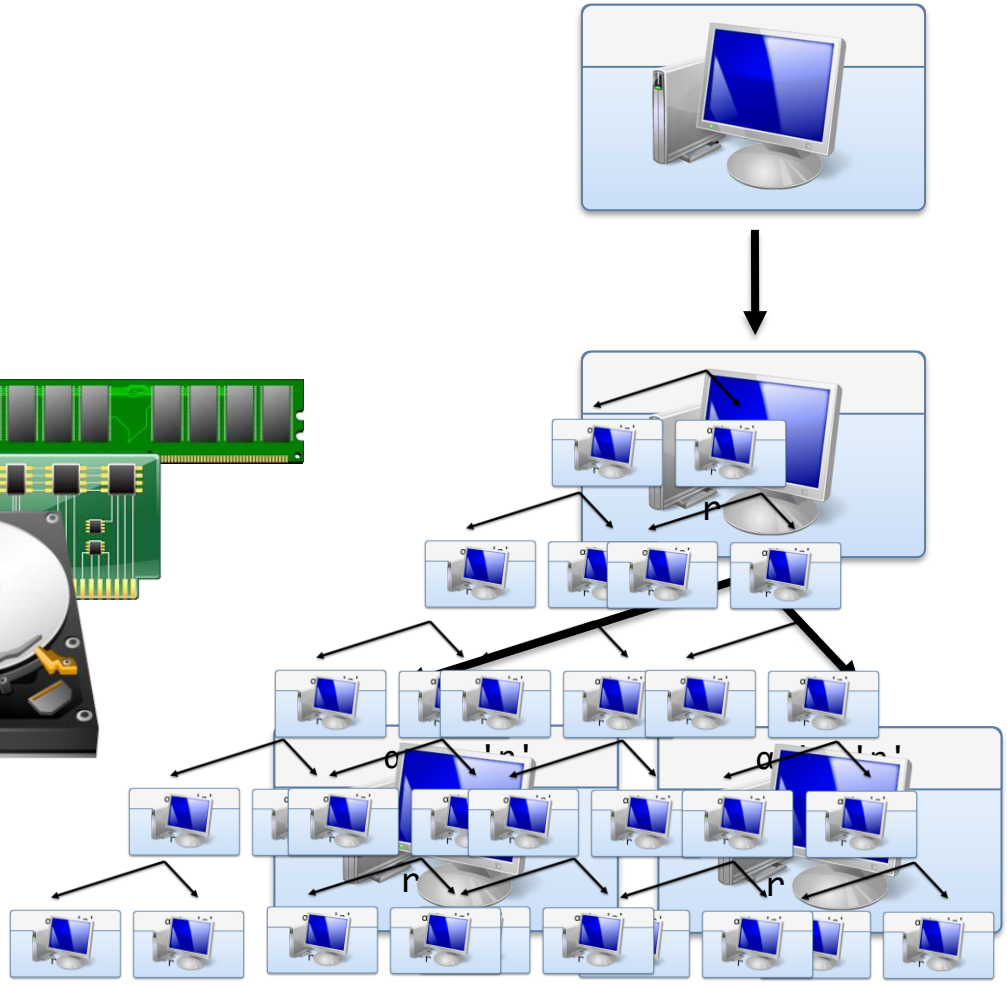


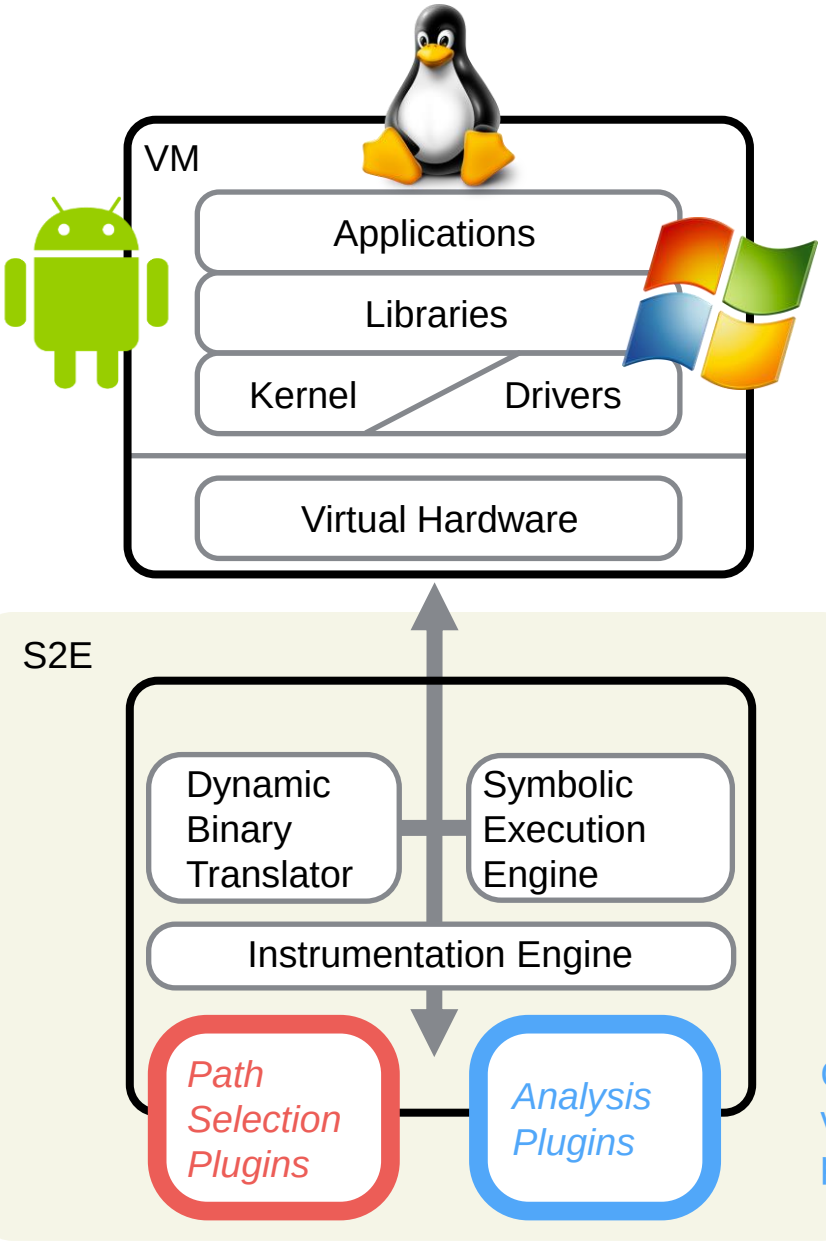
Dynamic Symbolic Execution

```
int func(char c) {
  int r = 1;

  if (c == 'n') {
     r = 0;
  }

  return r;
}
```



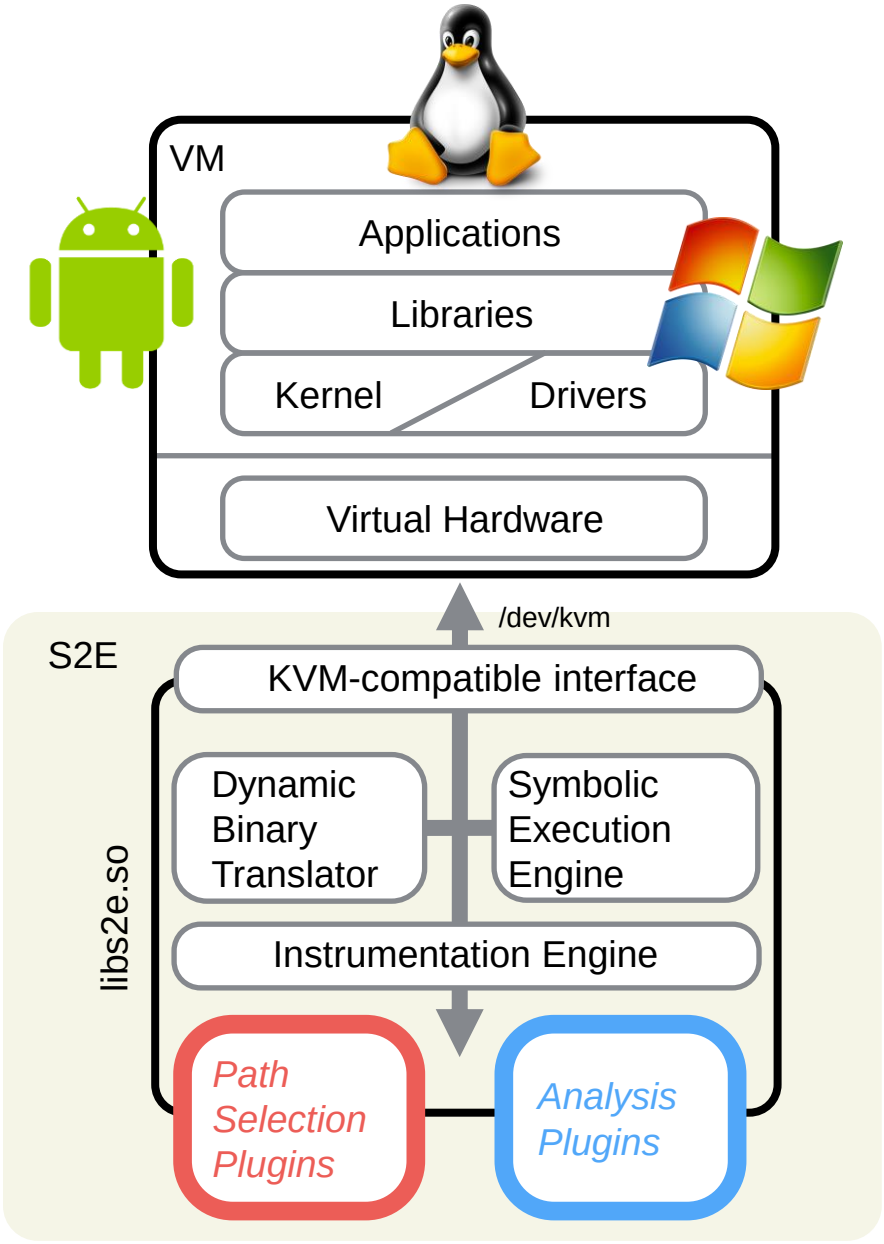
30 GB disk
4 GB RAM?





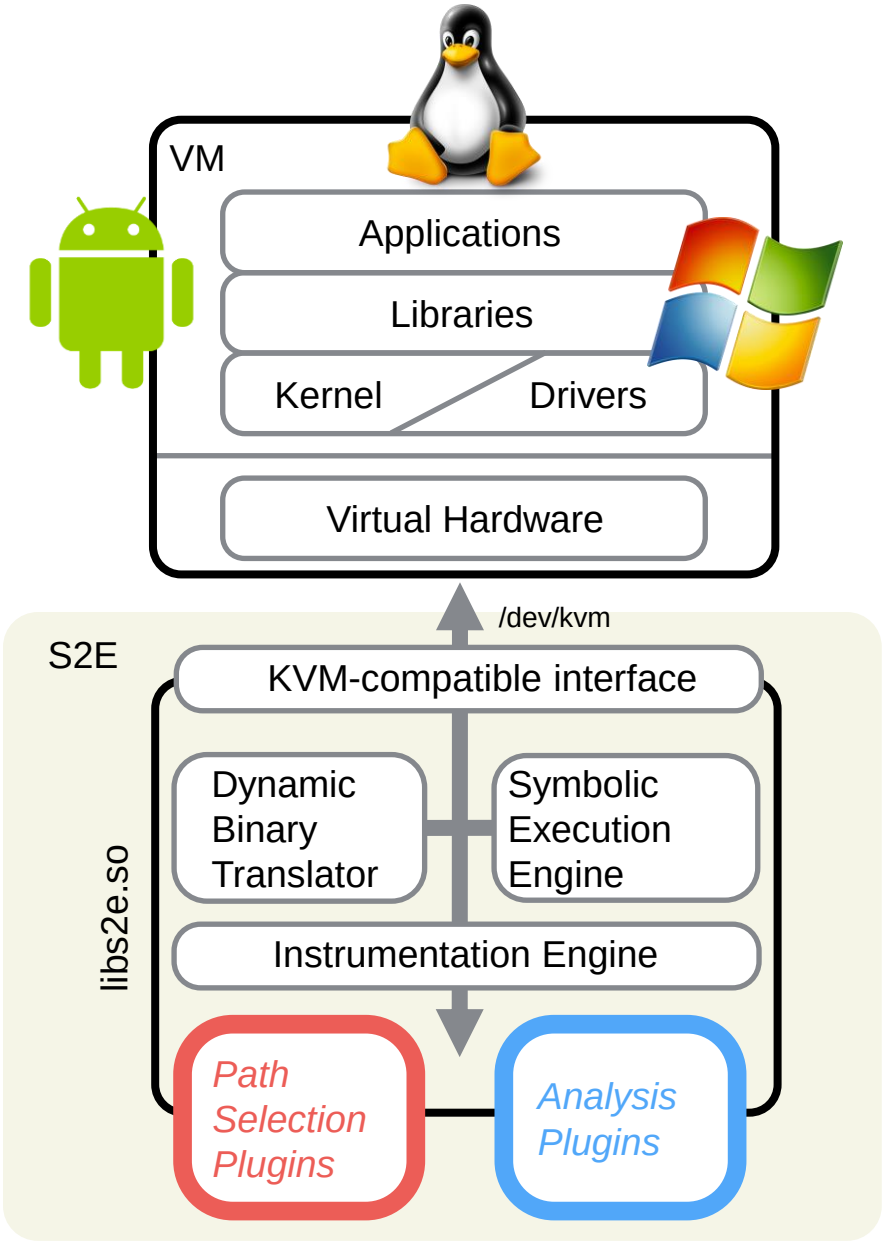
What input to make symbolic
What input to make concrete
Search heuristics

Check for crashes, vulnerability conditions, performance metrics, etc.



KVM Extensions for Symbolic Execution

- S2E uses QEMU
- S2E and QEMU are decoupled
- S2E is contained in libs2e.so
- libs2e.so intercepts and replaces /dev/kvm functionality
- Need a few simple KVM extensions to intercept DMA, disk R/W, and device state snapshotting
- You don’t have to use QEMU with S2E



Modular Architecture

- We refactored QEMU's translator to make it standalone
- libcpu, libtcg: code translation and generation libraries
- libs2ecore, libs2eplugins, klee, libvmi, etc.
- You can reuse these in your own projects
- You can swap out the symbolic execution engine with your own if you want

Dynamic Binary Translation

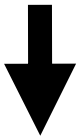
```
while(true) {
  tb = translate(cpu->pc)
  tb->func(cpu);
}
```

Host-independent micro-operations

Host instructions (x86, arm, etc.)

+LLVM

```
0x80000000: mov [ebx], eax
```



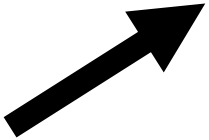
```
void tb_0x80000000(cpu) {
   tmp1 = cpu->regs[EBX];
   tmp2 = cpu->regs[EAX];
   __stl_mmu(tmp1, tmp2);
}
```

```
void tb_0x80000000(cpu) {
   s2e_call_plugins(cpu);
   tmp1 = cpu->regs[R_EBX];
   tmp2 = cpu->regs[R_EAX];
   __stl_mmu(tmp1, tmp2);
}
```

```
translate(pc) {
  do {
     if (s2e_instrument_ins(pc)) {
       emit_uops_s2e();
     }
     ins = disas(pc);
     emit_uops(ins);
     pc += ins.size;
  } while (ins != jmp);
}
```

Dynamic Binary Translation

```
0x80000000: mov [ebx], eax
```



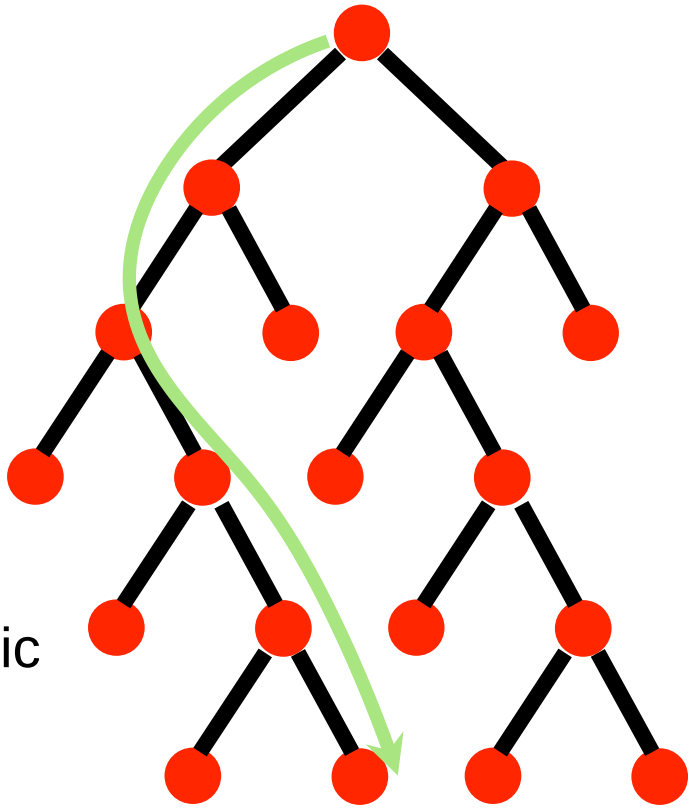
```
define i64 @tb_0x80000000(i64*) #12 {
entry:
  %loc_18ptr = alloca i32
  %loc_19ptr = alloca i32
  %1 = getelementptr i64, i64* %0, i32 0
  %2 = load i64, i64* %1
  %eax_ptr = getelementptr %struct.CPUX86State, …, i32 0, i32 0
  %ebx_ptr = getelementptr %struct.CPUX86State, …, i32 0, i32 2
  %eax = load i32, i32* %eax_ptr
  %ebx = load i32, i32* %ebx_ptr
  call void @__stl_mmu(i32 %ebx, i32 %eax)
  …
}
```

Symbolic Execution Engine

- Stripped down version of KLEE from ~2009
  20KLOC vs 60KLOC
- Replaced STP with Z3
- Cherry-picked LLVM-related updates from upstream
- Added concolic execution support

Concolic Execution

```
c = (α,'n')

int func(char c) {
  int r = 1;

  if (c == 'n') {
     r = 0;
  }

  return r;
}
```

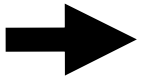


Use golden seeds to guide symbolic execution towards deeper paths

KLEE Improvements

Immutable Expressions

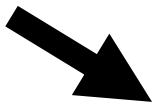
```
class ExtractExpr {
public:
  ref<Expr> expr;
  unsigned offset;
  Width width;
  ...
}
```



```
class ExtractExpr {
private:
  ref<Expr> expr;
  unsigned offset;
  Width width;
  ...
public:
  ref<Expr> getExpr();
  unsigned getOffset();
  Width getWidth();
}
```

Proper encapsulation

```
class Executor {
  ...
 Cell &getArgumentCell(ExecutionState &state, KFunction *kf, unsigned index);
 Cell &getDestCell(ExecutionState &state, KInstruction *target);
 void bindLocal(ExecutionState &state, KInstruction *target, ref<Expr> value);
 void bindArgument(ExecutionState &state, KFunction *kf, unsigned index, ref<Expr> value);
 void stepInstruction(ExecutionState &state);
 void bindObject(ExecutionState &state, const ObjectStatePtr &os, bool isLocal);  ...
}
```



```
class ExecutionState {
  ...
 Cell &getArgumentCell(KFunction *kf, unsigned index);
 Cell &getDestCell(KInstruction *target);
 void bindLocal(KInstruction *target, ref<Expr> value);
 void bindArgument(KFunction *kf, unsigned index, ref<Expr> value);
 void stepInstruction();
 void bindObject(const ObjectStatePtr &os, bool isLocal);  ...
}
```

KLEE Improvements

- Use smart pointers (almost) everywhere
  No new or delete

- Merged `MemoryObject` and `ObjectState`
  Fewer memory allocations

Outline

- How does S2E work?
  Scaling symbolic execution to entire VMs

- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Enterprise insider threat detection

- Future of S2E
  Making it 10-100 times faster

CYBER GRAND_CHALLENGE

The World's First All-Machine Hacking Tournament

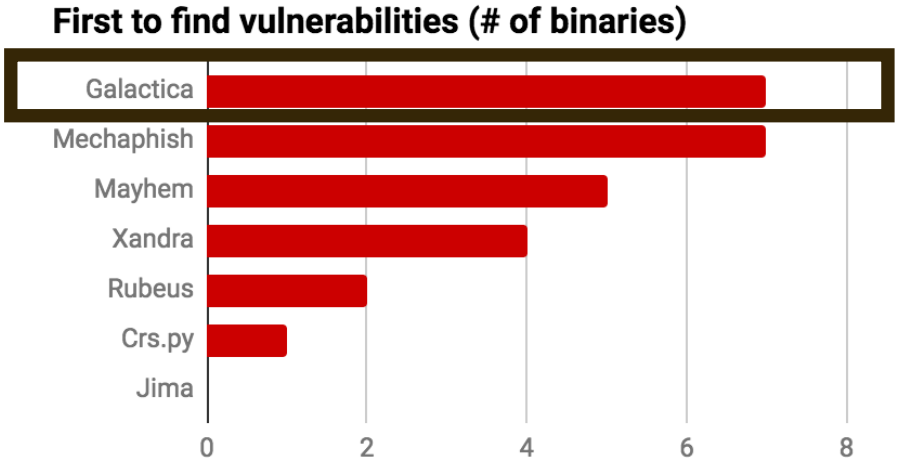
- Evaluate software for vulnerabilities (**Attack**)
- Defend software against attacks (**Defend**)
- Keep software running and available (**Availability**)

Two teams used S2E

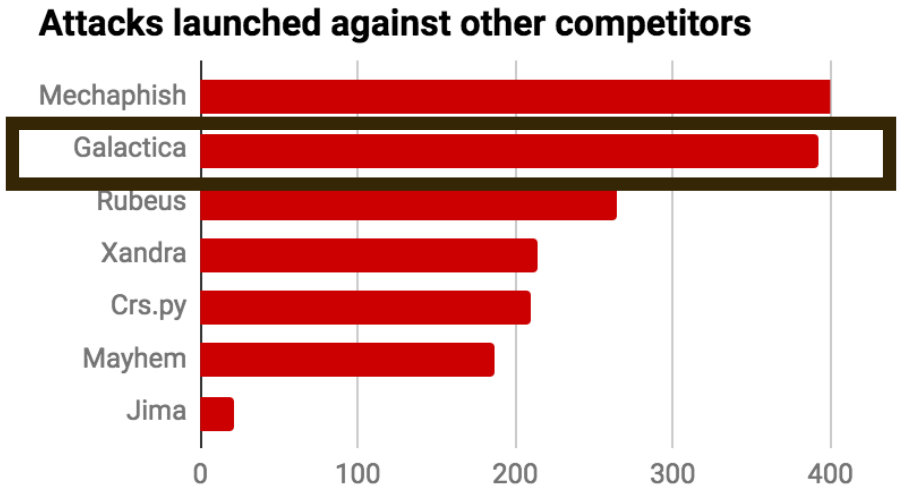
Team CodeJitsu

Cyberhaven
UC Berkeley
Syracuse University

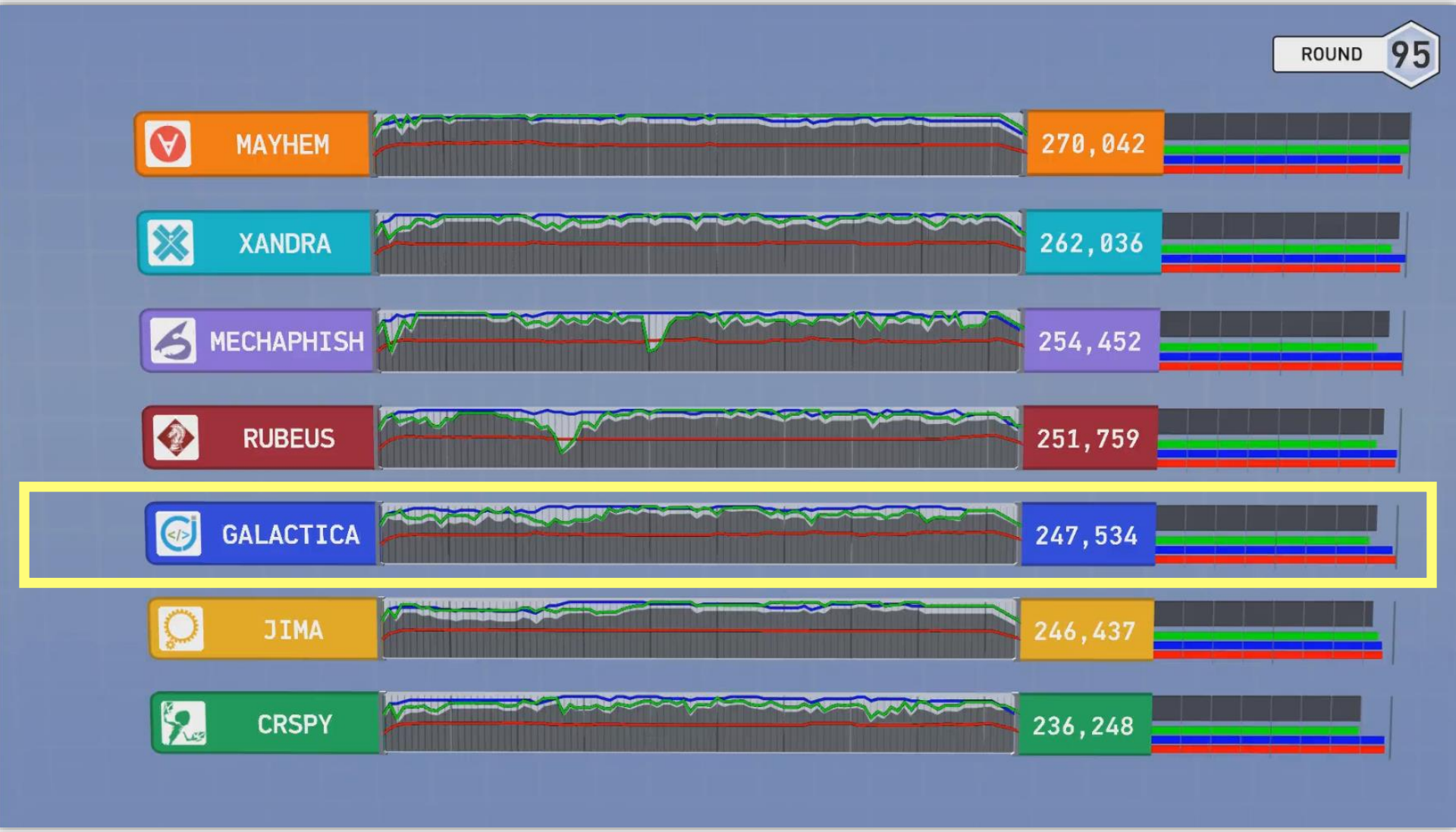
Team Disekt



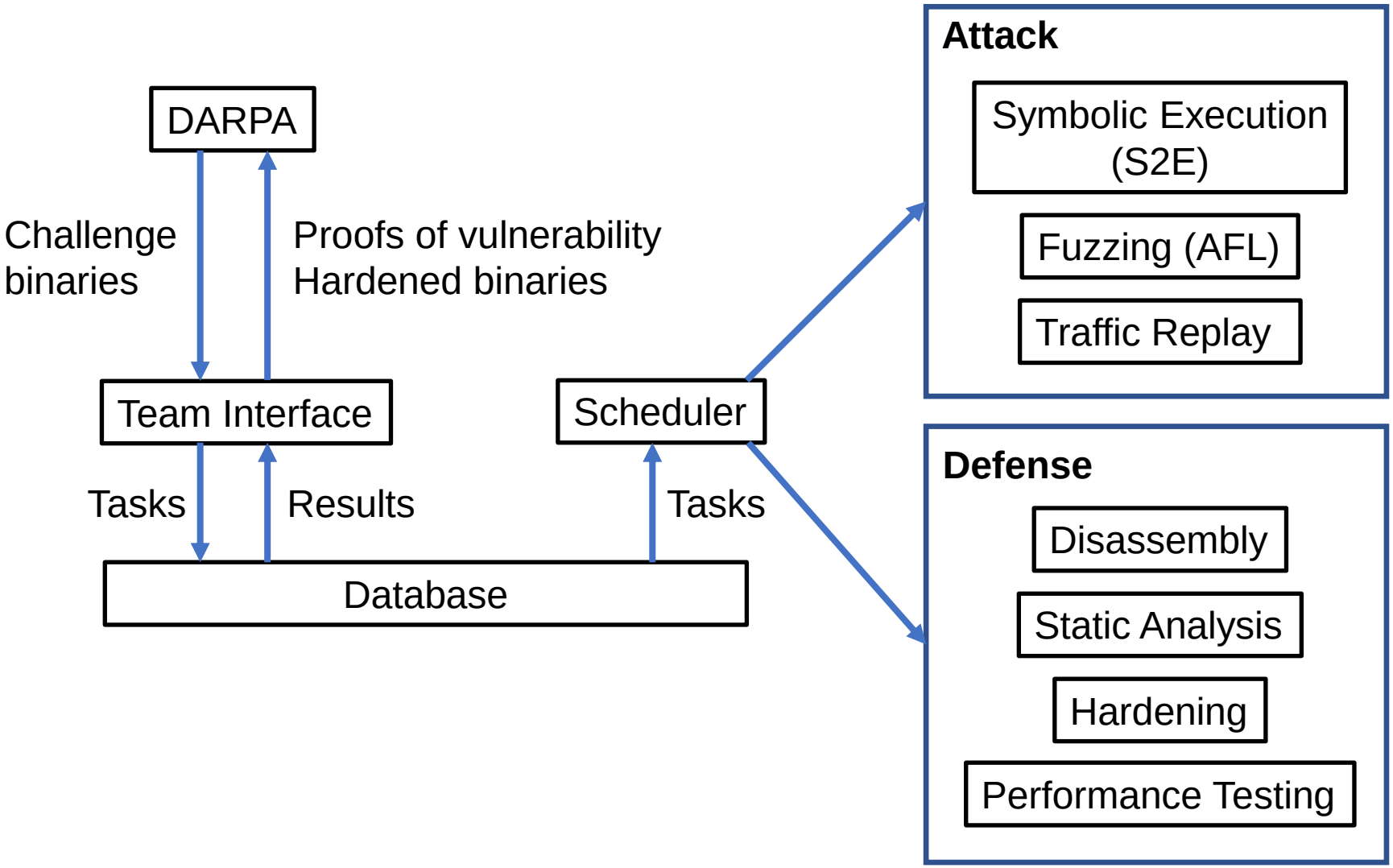
#1 fastest to attack: first to find and exploit vulnerabilities



#2 most effective: 392 successful attacks (1st place launched 402 attacks, 3rd got 265)


ROUND 95
MAYHEM
270,042
XANDRA
262,036
MECHAPHISH
254,452
RUBEUS
251,759
GALACTICA
247,534
JIMA
246,437
CRSPY
236,248

Architecture



CGC Cluster

- 64 nodes * 20 cores * 256GB mem * 1TB disk
- Scheduling resources for symbolic execution, fuzzing, hardening, and management tasks
- Component integration
- Reliability is top priority

CGC Cluster

- Shared storage
  postgres + glusterfs
- Automated deployment
  ansible
- Containerized apps
  docker
- Resource scheduling
  mesos
- Health monitoring and automated recovery
  monit

Cyberhaven Binary Analysis Engine

- Fully open source: https://s2e.systems/
- Documentation and tutorials
- Demo

```
docker run --rm -ti -w $(pwd) -v $HOME:$HOME \
cyberhaven/s2e-demo /demo/run.sh $(id -u) $(id -g) /demo/CADET_00001
```

Outline

- How does S2E work?
  Scaling symbolic execution to entire VMs

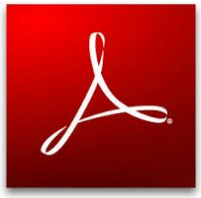
- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Enterprise insider threat detection

- Future of S2E
  Making it 10-100 times faster

Building a Malware Scanner

- Control flow integrity checking

- Diverse software stacks
  - Office 2007-2014
  - Acrobat Reader + Foxit Reader
  - Windows XP, 7, 8



- Cluster architecture
  - Ansible, Postgres, Django, Docker, Mesos

Control Flow Integrity Checker

```
func:
    0x00: push ebp
    0x01: mov ebp, esp
    0x03: mov eax, func1
    0x08: push param
    0x0E: call eax
    0x10: add esp, 4
    0x13: leave
    0x14: ret
```

```
func1:
    push ebp
    mov ebp, esp
    ...
    ; Overwrite EIP on the stack
    ...
    leave
    ret
```

Stack
ebp
param
0x10
ebp

Stack	Shadow stack
ebp	
param	
0x13371337	0x10
0xabcdef	

Shadow stack mismatch
CFI violation

Implementation

- CFI checker plugin (<600 LOCs)
- Supporting plugins
  WindowsMonitor, ProcessExecutionDetector, MemoryMap, ModuleMap, ExecutionTracer, UserSpaceTracer
- Automated disk image builder
  - 80 combinations of OSes and applications
- Automated GUI clicker (1.8 KLOC)
  - Dismiss any popups
  - Scroll documents
  - Decide when to stop the analysis

Challenges

- System code stack pointer manipulation
- JITed code
- Self-modifying code
- Identifying valid call targets
- No support for indirect jumps
- Single path, no symbolic execution

Performance

- 1’057’204 Office+PDF files analyzed
  - Diverse set of files, many corner cases
- 4’110’210 analyses
  - ~4 stacks per file
  - 5 - 15 min per file per stack
- 198 dangerous files detected
  - Many of them undetected by AVs
  - Bypassed all other security defenses deployed

Lessons Learned

- Too slow, cannot do inline scanning
  Malicious emails have to be deleted later
- Limited threat coverage
  One more solution to manage
- Existing antiviruses deemed good enough
  Windows Defender is built-in
- Strong competition
  Machine learning / AI
- Could take a lot of time before finding a threat
  Hard to demonstrate value quickly

Outline

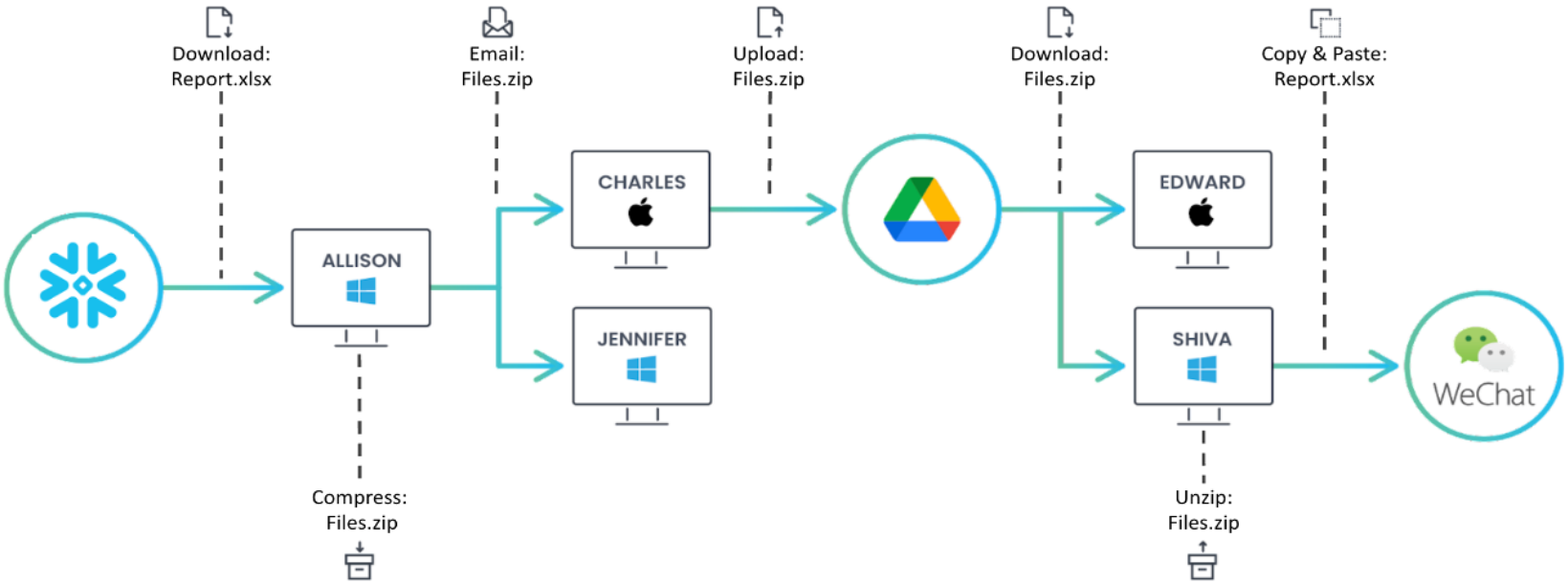
- How does S2E work?
  Scaling symbolic execution to entire VMs

- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Enterprise insider threat detection

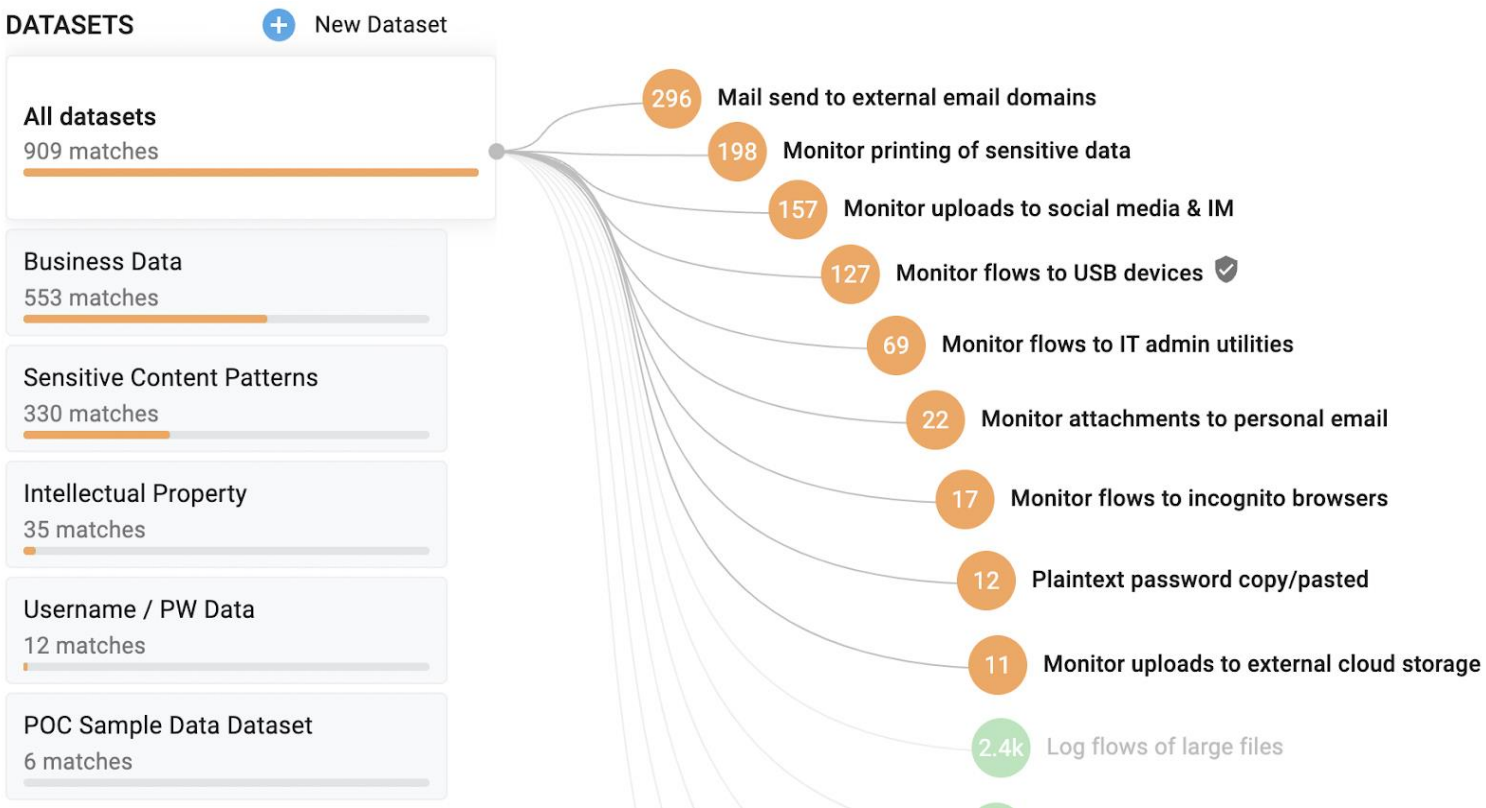
- Future of S2E
  Making it 10-100 times faster

CYBERHAVEN

Tracing dataflows in enterprises



CYBERHAVEN


DATASETS
New Dataset
All datasets
909 matches
Business Data
553 matches
Sensitive Content Patterns
330 matches
Intellectual Property
35 matches
Username / PW Data
12 matches
POC Sample Data Dataset
6 matches
296 Mail send to external email domains
198 Monitor printing of sensitive data
157 Monitor uploads to social media & IM
127 Monitor flows to USB devices
69 Monitor flows to IT admin utilities
22 Monitor attachments to personal email
17 Monitor flows to incognito browsers
12 Plaintext password copy/pasted
11 Monitor uploads to external cloud storage
2.4k Log flows of large files

- We do not use S2E anymore
  Open sourced everything we built with it
- Built new technology from scratch
  After listening to customers this time
- Scalable dataflow tracing on the backend
  Handle graphs with billions of nodes, 100k+ endpoints per customer
- Windows and MacOS endpoints
  Use all possible sources of events

Outline

- How does S2E work?
  Scaling symbolic execution to entire VMs

- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Insider threat detection in enterprises

- Future of S2E
  Making it run 10-100x faster

Typical user experience (in 2011)

- Download and build S2E
- Install guest OS
- Try to boot it in S2E
- Take a snapshot
- Write a config file from scratch
- ...
- Two weeks later, still

Everything was manual
No tooling, no doc

Typical user experience (in 2022)

- Build and run a demo in less than one hour
- It works! Let me try it on my own programs
- ...
- Why doesn’t it run faster?

Making S2E Fast

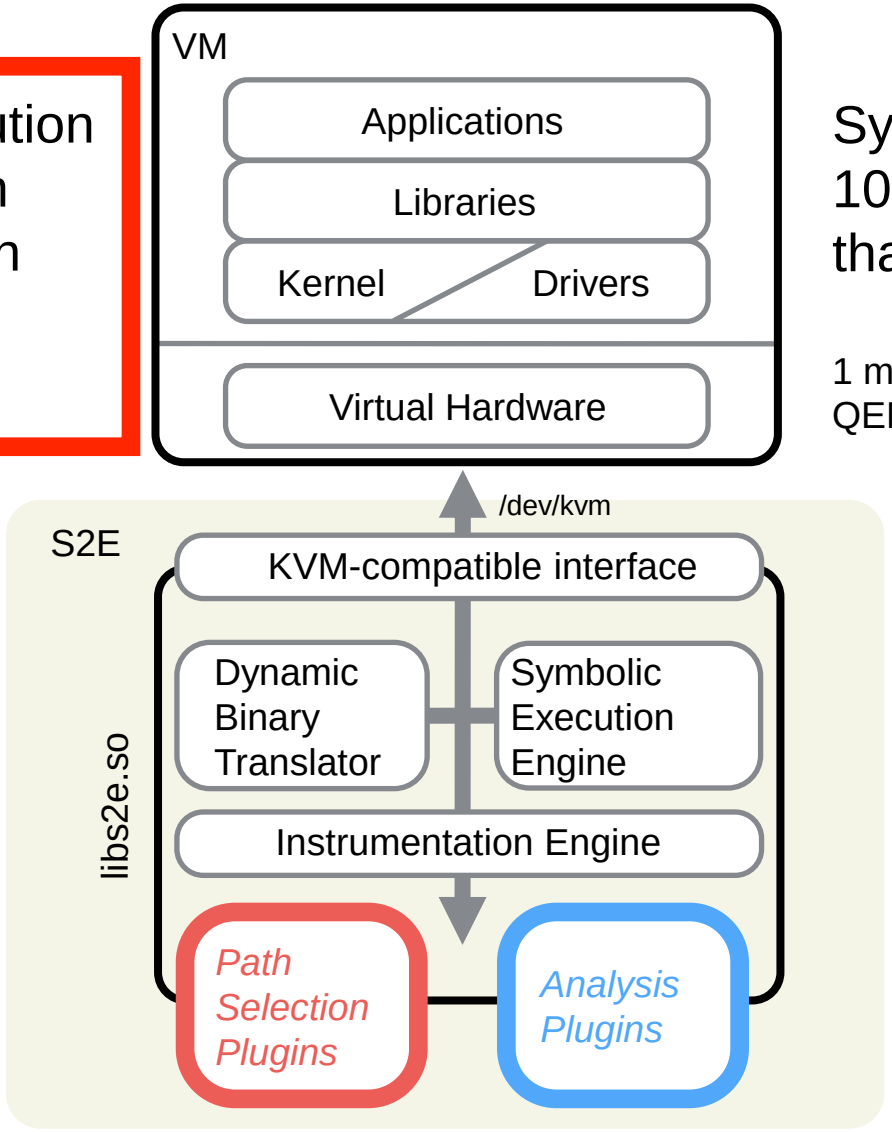
- Optimizing single-path execution
  Accommodate large software stacks

- Optimizing multi-path exploration
  Integrate state-of-the-art program analysis techniques

Bottlenecks

Concrete execution
10x slower than
native execution

5 min to boot Win7



Symbolic interpreter
100-1000x slower
than native execution

1 minute to boot
QEMU BIOS in KLEE

Dynamic Binary Translation

Guest code

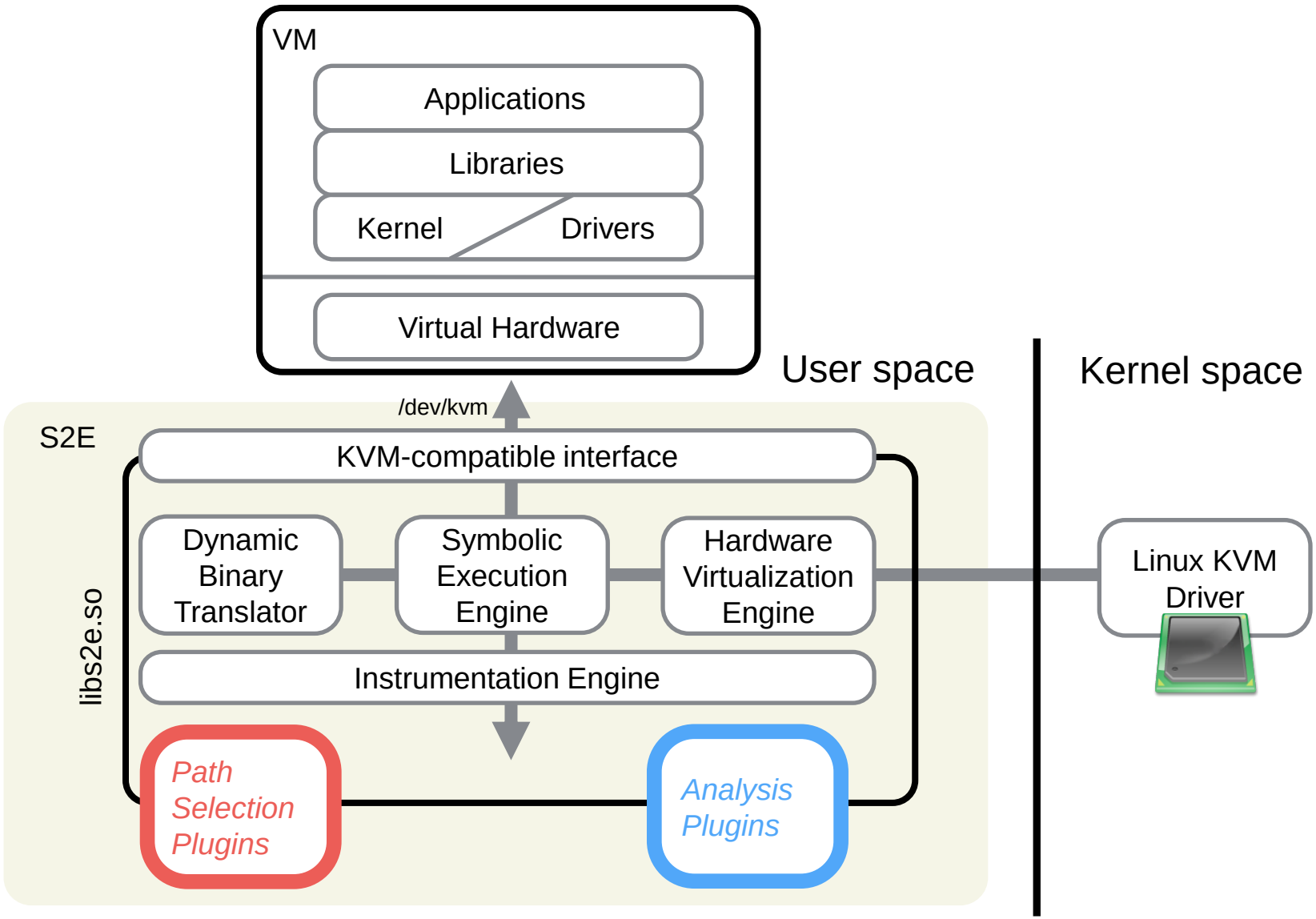
```
cmp    dl, 0x25
je     addr
```

23 instructions

Host code

```
mov    $0x7fbc565baf88,%rbp
mov    %rbp,0x518(%r14)
mov    0x10(%r14),%rbp
sub    $0x25,%rbp
mov    %rbp,%rbx
movzbl %bl,%ebx
mov    $0xe,%r12d
mov    %r12d,0x80(%r14)
mov    $0x25,%r12d
mov    %r12,0x88(%r14)
mov    %rbp,0x90(%r14)
…
```

Hardware Virtualization



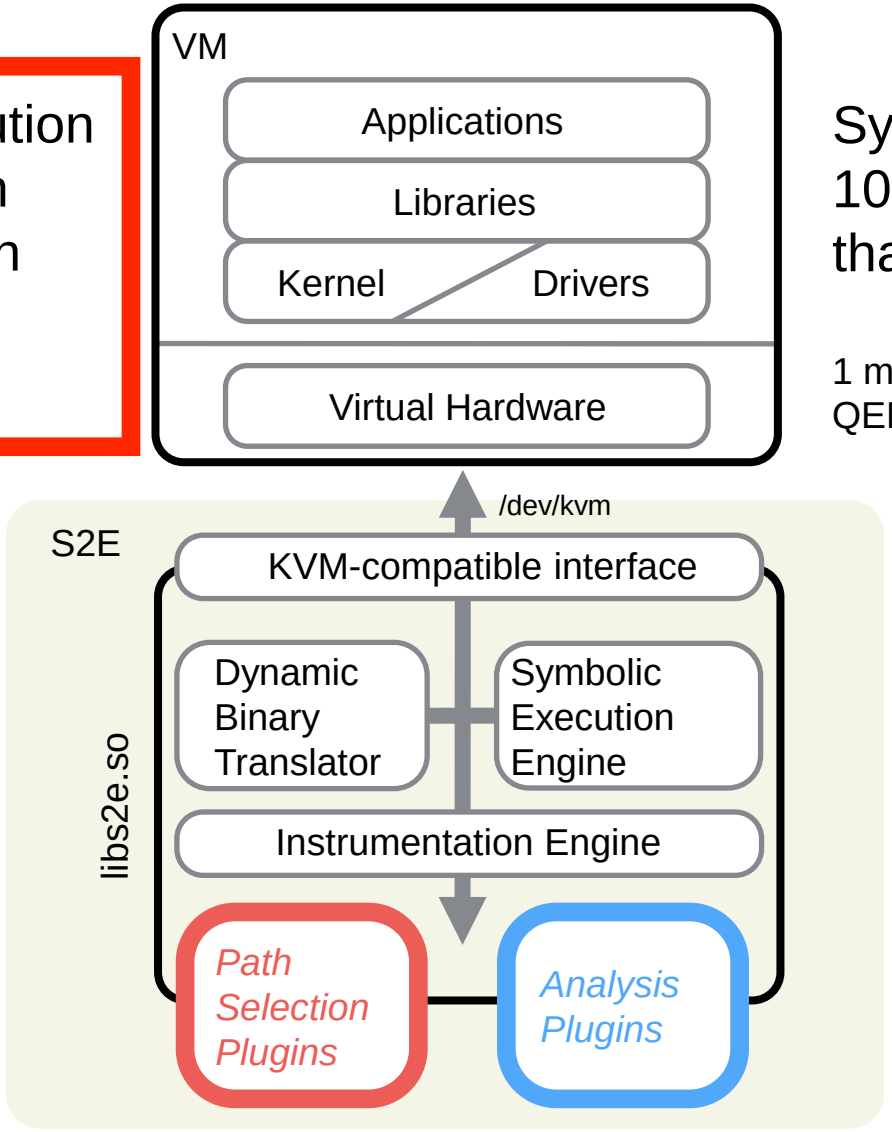
Hardware Virtualization Challenges

- Efficiently switching between DBT/KVM/KLEE
- Instrumenting code running in KVM

Bottlenecks

Concrete execution
10x slower than
native execution

5 min to boot Win7



Symbolic interpreter
100-1000x slower
than native execution

1 minute to boot
QEMU BIOS in KLEE

Dynamic Binary Translation

Guest code

```
cmp     dl, 0x25
je      addr
```

LLVM code

```
define i64 @tcg-llvm-tb-f27da-5-89-0-b4(i64*) alwaysinline {
entry:
  %1 = getelementptr i64* %0, i32 0
  %env_v = load i64* %1
  %state = inttoptr i64 %env_v to %struct.CPUX86State*
  %2 = getelementptr %struct.CPUX86State* %state, i32 0, i32 5
  %3 = getelementptr %struct.CPUX86State* %state, i32 0, i32 1
  %4 = add i64 %env_v, 1304
  %5 = inttoptr i64 %4 to i64*
  store i64 140373302116232, i64* %5
  store i64 993242, i64* %2
  %6 = getelementptr %struct.CPUX86State* %state, i32 0, i32 0, i32 2
  %rdx_v = load i64* %6
  %7 = getelementptr %struct.CPUX86State* %state, i32 0, i32 2
  store i64 37, i64* %7
  %tmp-18_v = sub i64 %rdx_v, 37
  %8 = getelementptr %struct.CPUX86State* %state, i32 0, i32 3
  store i64 %tmp-18_v, i64* %8
  store i64 993245, i64* %2
  store i64 14, i64* %3
  %9 = bitcast i64* %3 to i32*
  store i32 14, i32* %9
  %tmp4_v = and i64 %tmp-18_v, 255
  %10 = icmp eq i64 %tmp4_v, 0
  br i1 %10, label %label_0, label %11

; <label>:11                                      ; preds = %entry
  %12 = getelementptr i64* %0, i32 0
  %env_v1 = load i64* %12
  store i64 993247, i64* %2
  store i8 0, i8* inttoptr (i64 31703248 to i8*)
  ret i64 140373293713968

label_0:                                          ; preds = %entry
  %13 = getelementptr i64* %0, i32 0
  %env_v2 = load i64* %13
  store i64 993255, i64* %2
  store i8 1, i8* inttoptr (i64 31703248 to i8*)
  ret i64 140373293713969
}
```

Problems with LLVM

- Slow to generate
  45 minutes to boot Windows XP if translating all instructions to LLVM in addition to x86

- Slow to interpret
  Pathological case: tight loop with a million iterations

We need an intermediate representation
that is fast to generate and interpret

Tiny Code Interpreter (TCI)

- QEMU comes with TCI (Tiny Code Interpreter)
- Fast to translate and interpret
- Add symbolic expressions support to TCI
- KLEE will still be used to handle emulation helpers

Outline

- How does S2E work?
  Scaling symbolic execution to entire VMs

- Building commercial products
  Automated vulnerability analysis
  Scanning documents for malware
  Insider threat detection in enterprises

- Future of S2E
  Making it run 10-100x faster

Extensible
Write your own tools

On real OSes, with
real apps, libraries, drivers

S2E is a platform for in-vivo multi-path analysis of software systems

Symbolic execution
Concolic execution
State merging…

Bug finding
Verification
Testing
Security checking…

Pretty much anything that
runs on computers

https://s2e.systems
Ready-for-use docker image, demos,
tutorials, source code, documentation