Explore deficiencies in the state-of-the-art automatic software vulnerability mining technologies

Wish Wu

Security Expert, Ant Financial Tian Qiong Security Lab

Target

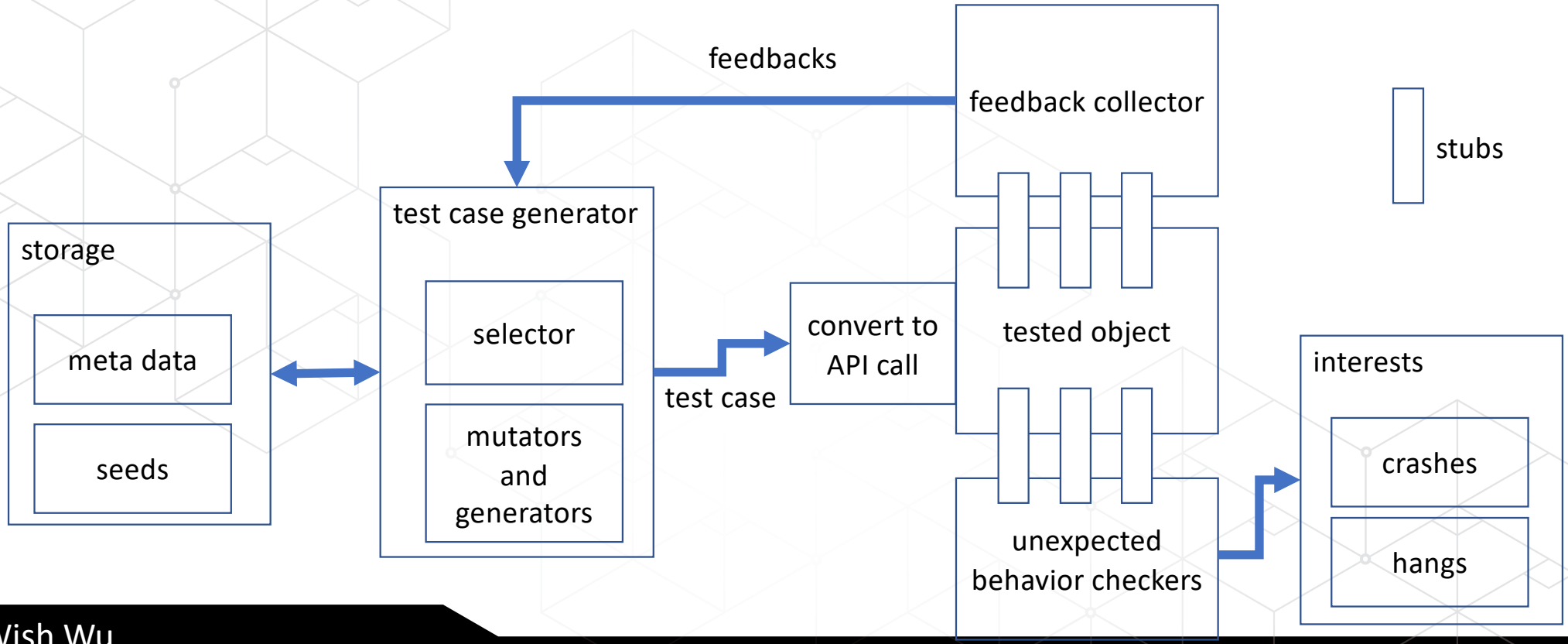
Vulnerability mining is completely done by machine and efficiency reaches or exceeds manual.

Current Reality

1. Find the magic numbers or keywords in the code to construct the dictionary.
2. Remove codes that prevent "effective testing", such as checksum() in libpng.
3. Prepare a large number of seed files that can run to different code blocks.
4. Write programs that use random numbers to generate "valid data".
5. Call the API selectively to ensure that the specified code can be tested.

...

Feedback-driven Genetic Algorithm



Core of GA

feedbacks:

trace-pc, trace-cmp, trace memcmp() …

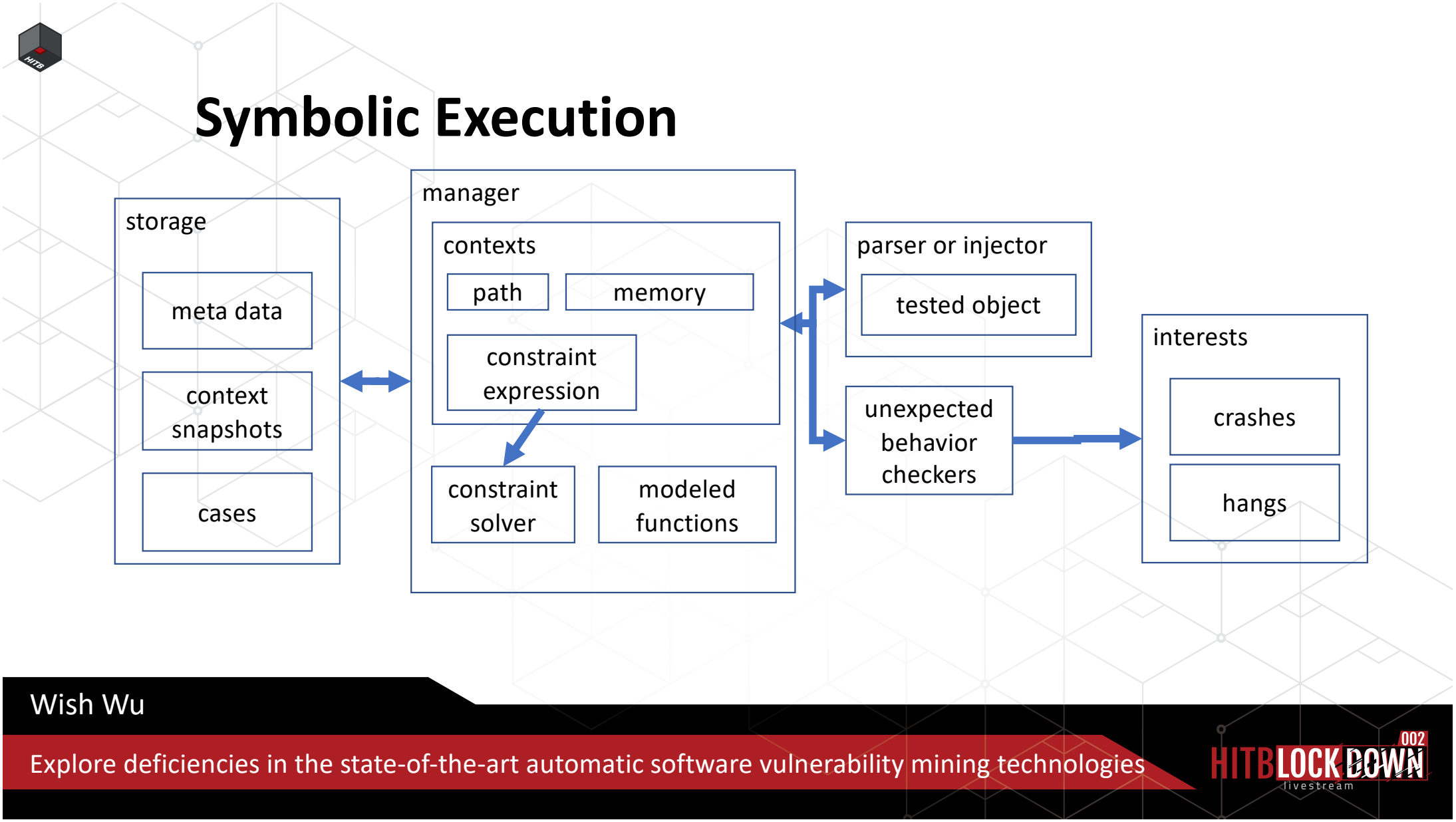
selector & mutators & generators:

insert, delete, replace, dictionary, grammar …

unexpected behavior checkers:

address sanitizer, thread sanitizers …

…


Symbolic Execution
storage
meta data
context snapshots
cases
manager
contexts
path
memory
constraint expression
constraint solver
modeled functions
parser or injector
tested object
unexpected behavior checkers
interests
crashes
hangs

Block AFL

```
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    uint32_t *num = (uint32_t *)Data;
    if (Size < sizeof(uint32_t))
        return 0;
    if (*num == 0xa1b2c3d4u)
        *((volatile uint8_t *)0) = 0;
    return 0;
}
```

Block libFuzzer and AFL

```
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    uint32_t *num = (uint32_t *)Data;
    if (Size < sizeof(uint32_t) * 2)
        return 0;
    //num[0] = 0x00621a27u; num[1] = 0x00c01752u;
    if (num[0] > 0x003e9ef4u && num[0] < 0x00649689u) {
        if (num[1] > 0x00b10797u && num[1] < 0x00f2deebu) {
            if ((num[0] * num[1]) == 0x00621a27u * 0x00c01752u) {
                *((volatile uint8_t *)0) = 0;
            }
        }
    }
    return 0;
}
```

Block QSYM and KLEE

```
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    if (Size < sizeof(uint32_t) * 16)
        return 0;
    uint8_t flag = 0;
    uint32_t *num = (uint32_t *)Data;
    //num[0] = 0x00621a27; num[1] = 0x00c01752;
    if (num[0] > 0x003e9ef4 && num[0] < 0x00649689) {
        if (num[1] > 0x00b10797 && num[1] < 0x00f2deeb) {
            if ((num[0] * num[1]) == 0x00621a27 * 0x00c01752) {
                flag |= (uint8_t)0x1;
            }
        }
    }
    //num[2] = 0x013520fa; num[3] = 0x018d6191;
    if (num[2] > 0x0112bc98 && num[2] < 0x01c16abd) {
        if (num[3] > 0x01596565 && num[3] < 0x01be1786) {
            if ((num[2] * num[3]) == 0x013520fa * 0x018d6191) {
                flag |= (uint8_t)(0x1 << 1);
            }
        }
    }
    //num[4] = 0x025c6ef7; num[5] = 0x02145f29;
    if (num[4] > 0x024bde68 && num[4] < 0x0266302e) {
        if (num[5] > 0x0201deb3 && num[5] < 0x026191e9) {
            if ((num[4] * num[5]) == 0x025c6ef7 * 0x02145f29) {
                flag |= (uint8_t)(0x1 << 2);
            }
        }
    }
    //num[12] = 0x0681b201; num[13] = 0x0629a9d9;
    if (num[12] > 0x067fd111 && num[12] < 0x0691d629) {
        if (num[13] > 0x06209857 && num[13] < 0x06d93676) {
            if ((num[12] * num[13]) == 0x0681b201 * 0x0629a9d9) {
                flag |= (uint8_t)(0x1 << 6);
            }
        }
    }
    //num[14] = 0x074fd355; num[15] = 0x075e1841;
    if (num[14] > 0x073f66a5 && num[14] < 0x07f04124) {
        if (num[15] > 0x07414558 && num[15] < 0x078e3e98) {
            if ((num[14] * num[15]) == 0x074fd355 * 0x075e1841) {
                flag |= (uint8_t)(0x1 << 7);
            }
        }
    }
    if (flag == (uint8_t)0xff) {
        *((volatile uint8_t *)0) = 0;
    }
    return 0;
}
```

Stutter Fuzzers

https://github.com/arcslab/StutterFuzzers.git

	AFL	libFuzzer	KLEE	QSYM
stutter_AFL.c	Yes			
stutter_libFuzzer_and_AFL.c	Yes	Yes		
stutter_All_for_klee.c			Yes	
stutter_All_for_QSYM_libFuzzer_AFL.c	Yes	Yes		Yes

Inapproximable Constraint

libFuzzer and AFL have their own methods to deal with condition statement.

libFuzzer:

Compile with "–fsanitize-coverage=trace-cmp"

if (A < B) ➔ trace_cmp(A, B); if (A < B)

Use a variety of distance algorithms to calculate the similarity between A and B

Improved AFL:

```
if ( A == constNumber )
➔ if (A[0:8] == constNumber[0:8]) {
        trace_pc();
        if (A[8:16] == constNumber[8:16]) {
                trace_pc();
                …
```

Unable to solve inapproximable problems

Feedback of libFuzzer

clang -g -O2 -fno-omit-frame-pointer -fsanitize=fuzzer -c stutter_libFuzzer_and_AFL.c

```
 if (num[0] > 0x003e9ef4u && num[0] < 0x00649689u) {
753d0:       41 8b 1e                 mov    (%r14),%ebx
753d3:       bf f4 9e 3e 00           mov    $0x3e9ef4,%edi
753d8:       89 de                    mov    %ebx,%esi
753da:       e8 01 63 fd ff           callq  4b6e0 <__sanitizer_cov_trace_const_cmp4>
753df:       81 fb f4 9e 3e 00        cmp    $0x3e9ef4,%ebx
753e5:       76 6f                    jbe    75456 <LLVMFuzzerTestOneInput+0xb6>
753e7:       bf 89 96 64 00           mov    $0x649689,%edi
753ec:       89 de                    mov    %ebx,%esi
753ee:       e8 ed 62 fd ff           callq  4b6e0 <__sanitizer_cov_trace_const_cmp4>
753f3:       81 fb 89 96 64 00        cmp    $0x649689,%ebx
753f9:       73 64                    jae    7545f <LLVMFuzzerTestOneInput+0xbf>
```

Feedback of libFuzzer

```
ATTRIBUTE_INTERFACE
ATTRIBUTE_NO_SANITIZE_ALL
void __sanitizer_cov_trace_pc_indir(uintptr_t Callee) {
  uintptr_t PC = reinterpret_cast<uintptr_t>(GET_CALLER_PC());
  fuzzer::TPC.HandleCallerCallee(PC, Callee);
}
```

Trace code block

```
ATTRIBUTE_INTERFACE
ATTRIBUTE_NO_SANITIZE_ALL
ATTRIBUTE_TARGET_POPCNT
void __sanitizer_cov_trace_cmp8(uint64_t Arg1, uint64_t Arg2) {
  uintptr_t PC = reinterpret_cast<uintptr_t>(GET_CALLER_PC());
  fuzzer::TPC.HandleCmp(PC, Arg1, Arg2);
}
ATTRIBUTE_INTERFACE ATTRIBUTE_NO_SANITIZE_MEMORY
void __sanitizer_weak_hook_memcmp(void *caller_pc, const void *s1,
                                  const void *s2, size_t n, int result) {
  if (!fuzzer::RunningUserCallback) return;
  if (result == 0) return;  // No reason to mutate.
  if (n <= 1) return;  // Not interesting.
  fuzzer::TPC.AddValueForMemcmp(caller_pc, s1, s2, n, /*StopAtZero*/false);
}
```

Trace comparison

https://github.com/llvm/llvm-project.git compiler-rt/lib/fuzzer/FuzzerTracePC.cpp

Distance Algorithm of libFuzzer

```
template <class T>
ATTRIBUTE_TARGET_POPCNT ALWAYS_INLINE
ATTRIBUTE_NO_SANITIZE_ALL
void TracePC::HandleCmp(uintptr_t PC, T Arg1, T Arg2) {
  uint64_t ArgXor = Arg1 ^ Arg2;
  if (sizeof(T) == 4)
      TORC4.Insert(ArgXor, Arg1, Arg2);
  else if (sizeof(T) == 8)
      TORC8.Insert(ArgXor, Arg1, Arg2);
  uint64_t HammingDistance = Popcountll(ArgXor);  // [0,64]
  uint64_t AbsoluteDistance = (Arg1 == Arg2 ? 0 : Clzll(Arg1 - Arg2) + 1);
  ValueProfileMap.AddValue(PC * 128 + HammingDistance);
  ValueProfileMap.AddValue(PC * 128 + 64 + AbsoluteDistance);
}
```

Make dictionary

Get hamming distance and absolute distance

https://github.com/llvm/llvm-project.git compiler-rt/lib/fuzzer/FuzzerTracePC.cpp

Massive Bug-free Paths

```
flags = 0;
…
if( A ) flags |= 1;
…
if( B ) flags |= 1 << 1;
…
if( C ) flags |= 1 << 2;
…
if( D ) flags |= 1 << 3;
…
…
if( G ) flags |= 1 << 7;
…
if (flags == 0xff)
                bug();
```

1. Vulnerability exists only in very specific or unique path.
2. There are many conditions for the vulnerability.

Discovery

1. Coverage is losing its effectiveness.
2. Selecting path is better than traversing.
3. Constraint solver is necessary.

Code Review

```
//num[12] = 0x0681b201; num[13] = 0x0629a9d9;
if (num[12] > 0x067fd111 && num[12] < 0x0691d629) {
    if (num[13] > 0x06209857 && num[13] < 0x06d93676) {
        if ((num[12] * num[13]) == 0x0681b201 * 0x0629a9d9) {
            flag |= (uint8_t)(0x1 << 6);
        }
    }
}
//num[14] = 0x074fd355; num[15] = 0x075e1841;
if (num[14] > 0x073f66a5 && num[14] < 0x07f04124) {
    if (num[15] > 0x07414558 && num[15] < 0x078e3e98) {
        if ((num[14] * num[15]) == 0x074fd355 * 0x075e1841) {
            flag |= (uint8_t)(0x1 << 7);
        }
    }
}
if (flag == (uint8_t)0xff) {
    *((volatile uint8_t *)0) = 0;
}
return 0;
}
```

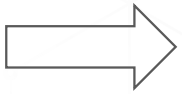
Find path to satisfy constraints

Sufficient and necessary constraints

```
if (flags == 0xff)
        *((volatile uint8_t *)0) = 0;
```

flags == 0xff

```
memcpy(dst, src, size);
```



size > allocated_size(dst)
|| size > allocated_size(src)

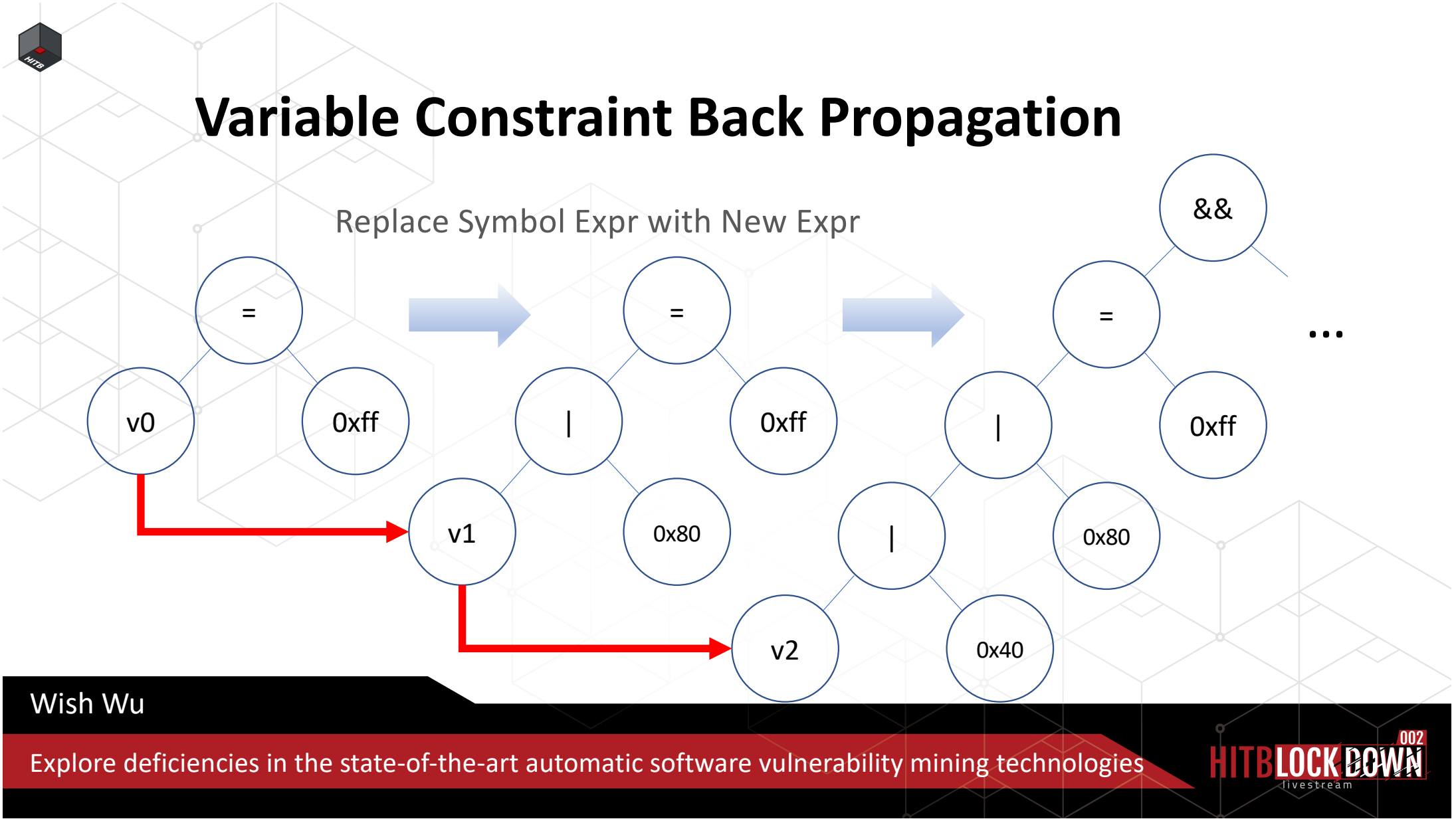
vulnerability is a set of sufficient and necessary constraints

Code Review

1. Assume constraints that make the vulnerability exist can be satisfied.

2. Backpropagate constraints until all variables are input.

3. Check the solvability of constraints during backpropagation.



Variable Constraint Back Propagation

Replace Symbol Expr with New Expr

=
v0
0xff
=
|
0xff
v1
0x80
=
|
0xff
v2
&&
=
|
0xff
0x80
0x40
...

Transformation of constraint expressions

```
int ExprSet::replaceSymbol(void *oldSym, std::shared_ptr<Expr> newExpr) {
    if (newExpr == nullptr) {
        printf("error:replaceSymbol find bad expr\n");
        return -1;
    }
    std::map<void *, std::list<std::shared_ptr<Expr>>>::iterator it;
    it = mSymMap.find(oldSym);
    if (it == mSymMap.end()) {
        printf("error: no such symbole %p\n", oldSym);
        return -1;
    }
    std::list<std::shared_ptr<Expr>> symExprs = it->second;
    mSymMap.erase(it);
    for (const std::shared_ptr<Expr> &expr : symExprs) {
        std::shared_ptr<Expr> parent = expr->mParent.lock();
        std::shared_ptr<Expr> copy = Expr::dupWithSymMap(newExpr, mSymMap);
        copy->mParent = parent;
        if (parent != nullptr) {
            if (parent->mLHS == expr) {
                parent->mLHS = copy;
            } else if (parent->mRHS == expr) {
                parent->mRHS = copy;
            } else {
                printf("error: something wrong!!!!\n");
                return -1;
            }
        } else {
            mRoot = copy;
        }
    }
    return 0;
}
```

Replace Symbol Expr with New Expr

Variable Constraint Back Propagation

```
  %144 = select i1 %142, i8 %143, i8 %128
  br label %145

145:                                              ; preds = %134, %127
  %146 = phi i8 [ %128, %127 ], [ %144, %134 ]
  %147 = icmp eq i8 %146, -1
  br i1 %147, label %148, label %149

148:                                              ; preds = %145
  store volatile i8 0, i8* null, align 536870912, !tbaa !8
  br label %149
```

Back propagate on LLVM bitcode

Back Propagation on LLVM bitcode

```
} else if (isa<BinaryOperator>(val)) {
    const BinaryOperator *BO = dyn_cast<BinaryOperator>(&val);
    Instruction::BinaryOps ops = BO->getOpcode();
    switch(ops) {
    case Instruction::BinaryOps::And: {
        const Value *lhs = BO->getOperand(0);
        const Value *rhs = BO->getOperand(1);
        if (isa<Instruction>(*rhs)) {
            std::shared_ptr<Variable> lhsvar(new Variable((IRVariable)lhs));
            std::shared_ptr<Variable> rhsvar(new Variable((IRVariable)rhs));
            char name[64];
            snprintf(name, sizeof(name) - 1, "%p", lhsvar.get());
            std::shared_ptr<Expr> lhssym = Expr::Sym(lhsvar.get(), name,
                    Expr::Type::BOOLEAN_TYPE, 1);
            snprintf(name, sizeof(name) - 1, "%p", rhsvar.get());
            std::shared_ptr<Expr> rhssym = Expr::Sym(rhsvar.get(), name,
                    Expr::Type::BOOLEAN_TYPE, 1);
            std::shared_ptr<Expr> andop = Expr::And(lhssym, rhssym);
            if (bs.mExpSet.replaceSymbol(var.get(), andop) != 0)
                return -1;
            std::set<std::shared_ptr<Variable>>::iterator it;
            it = bs.mVars.find(var);
            if (it == bs.mVars.end()) {
                LOG("error: no such var\n");
                return -1;
            }
            bs.mVars.erase(it);
            bs.mVars.insert(lhsvar);
            bs.mVars.insert(rhsvar);
        }
        break;
```

Back propagate on bitwise AND

Variable Constraint Back Propagation

Demo

Solve the codes that block QSYM, KLEE, AFL and libFuzzer

Wish Wu

Explore deficiencies in the state-of-the-art automatic software vulnerability mining technologies

Imitate manual code review

1. Make assumptions and initial constraints

   assert(), address sanitizer …
2. Use fuzz tool to get concrete paths

   libFuzzer,AFL…
3. Back propagate constraints over a certain path
4. Use approximation algorithms to satisfy constraints

   Constraint-guided Fuzz ?

Meme

I used reverse symbolic execution combined with an improved genetic algorithm to find an existing bug.

Found 3 new bugs with dumb fuzz.

Thank You!

HITB LOCKDOWN 002
livestream

Wish Wu (@wish_wu)