QueryX: Symbolic Query on Decompiled Code for Finding Bugs in COTS Binaries

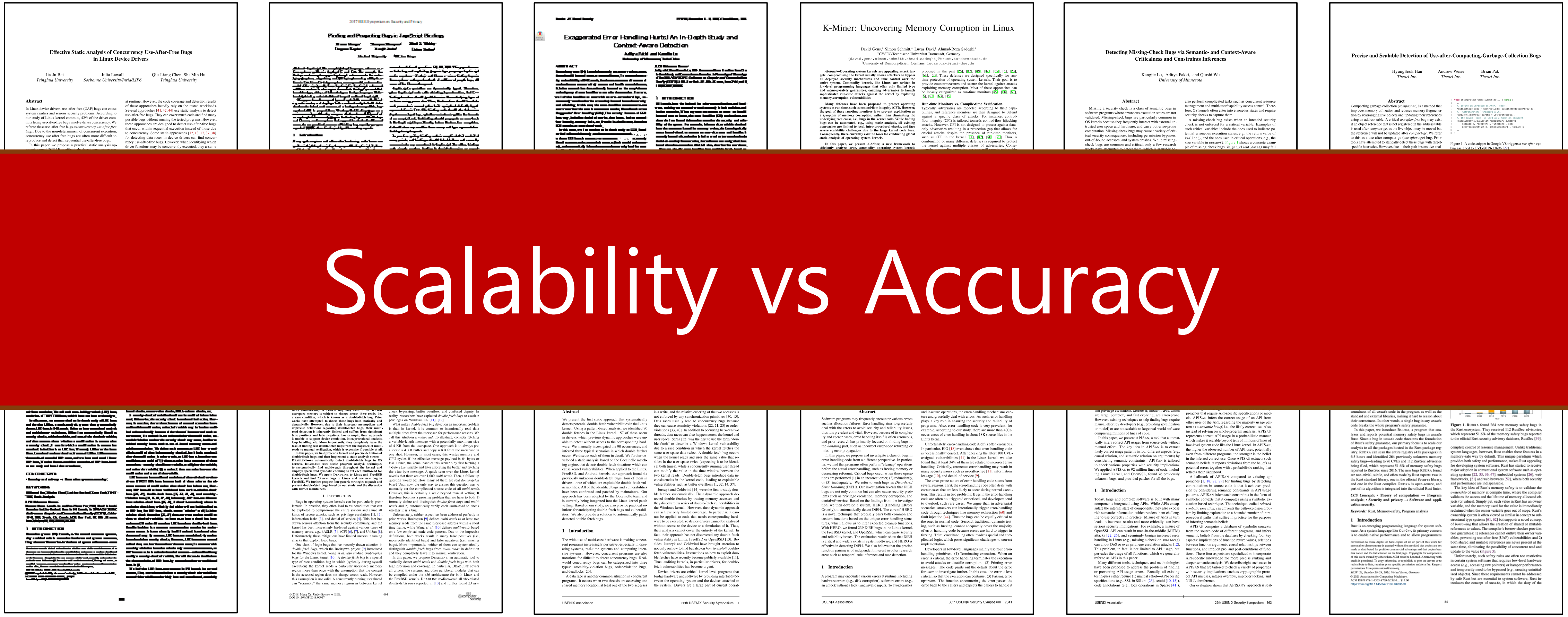
HyungSeok Han[†*], JeongOh Kyea[†], Yonghwi Jin[†], Jinoh Kang[†], Brian Pak[†], Insu Yun*

[†]Theori Inc. [*]KAIST

(Currently postdoc in Georgia Tech)

Static Analysis

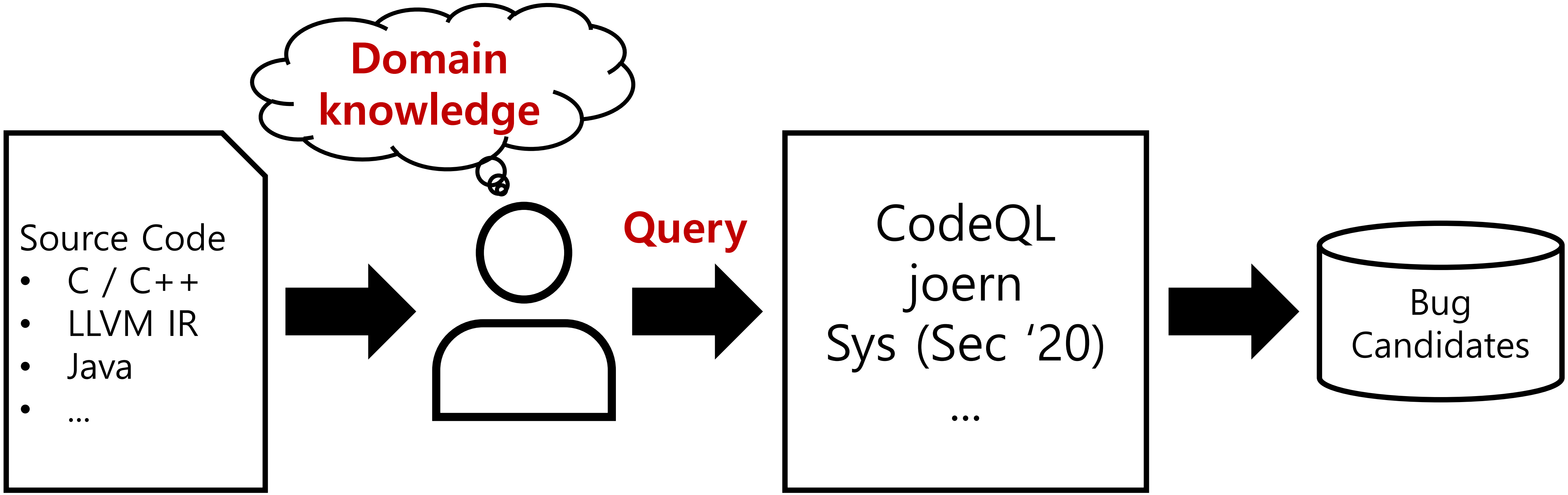
- One of the most popular techniques for automatic bug finding



Scalability vs Accuracy

Extensible Static Checking Tools

- Static analysis **+ Domain knowledge (Query)**



Extensible Static Checking Tools

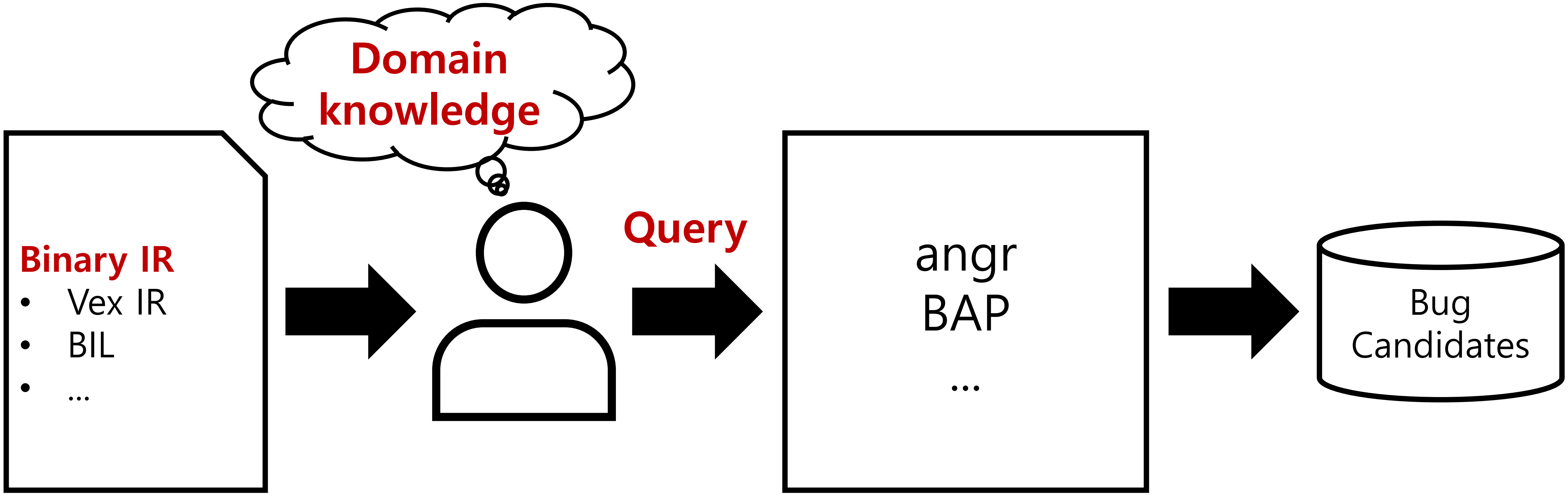
- Static analysis + Domain knowledge (Query)

	Input	Analysis objects	Syntactic	Data-flow	Symbolic
CodeQL [22]	Source	Source	✓	✓	×
joern [76]	Source	Source	✓	✓	×
Sys [7]	Source	LLVM IR	✓	✓	✓

Binary ?????

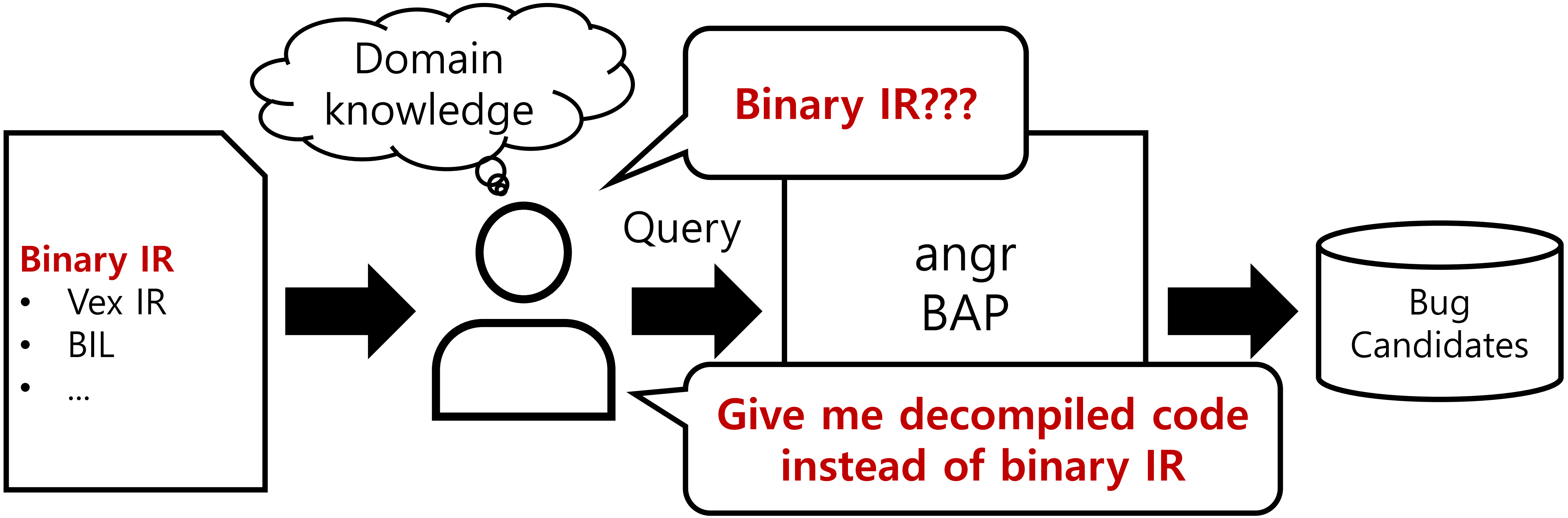
Extensible Static **Binary** Checking Tools

- Query based on **binary IRs**



Extensible Static **Binary** Checking Tools

- Query based on **binary IRs**

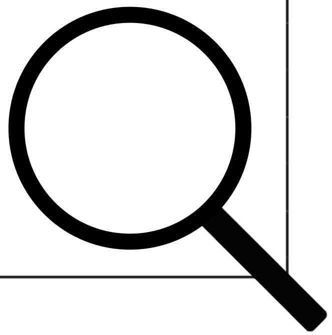


Inconsistency b/w Analyst & Query

- Analysts mostly work with **decompiled code** due to its high-level information such as high-level control flows and types.
- But, queries are based on **binary IRs** because binary IRs are more analyzer-friendly.

```
__int64 __fastcall isPalindrome(const char *a1) {
  int i; // [rsp+18h] [rbp-8h]
  int v3; // [rsp+1Ch] [rbp-4h]

  v3 = strlen(a1);
  for ( i = 0; i < v3 / 2; ++i ) {
    if ( a1[i] != a1[v3 - i - 1] )
      return 0LL;
  }
  return 1LL;
}
```



```
------ IMark(0x40069a, 1, 0) ------
t0 = GET:I64(rbp)
t10 = GET:I64(rsp)
t9 = Sub64(t10,0x0000000000000008)
PUT(rsp) = t9
STle(t9) = t0
------ IMark(0x40069b, 3, 0) ------
PUT(rbp) = t9
------ IMark(0x40069e, 4, 0) ------
t2 = Sub64(t9,0x0000000000000020)
; + 285 more instructions
```

Extensible Static Binary Checking Tools

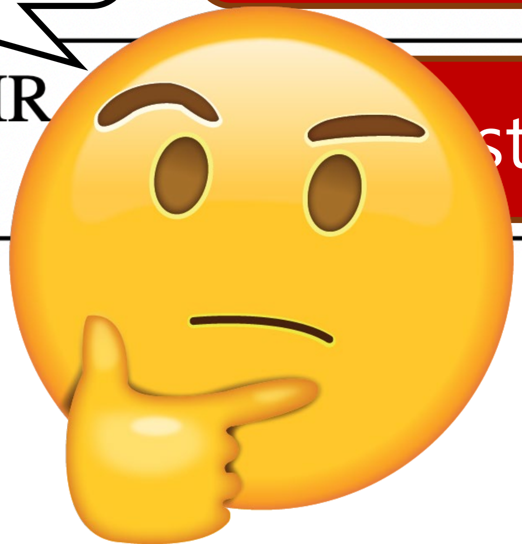
- Static analysis + Domain knowledge (Query)

	Input	Analysis objects	Syntactic	Data-flow	Symbolic
CodeQL [22] joern [76] Sys [7]					
angr [56] BAP [9]	Binary Binary	VEX IR BIL			

Decompiled code

Cannot handle binaries

stency b/w Analyst & Query



The Convergence of Source Code and Binary Vulnerability Discovery - A Case Study (AsiaCCS '22)

- Feed decompiled code from Hex-Rays to CodeQL and joern.
- Compared to tools with the original source code, tools with decompiled code got **less TP** and **more FP** because..

```c
int glob_x = 0x42;
struct Pos { int x; int y; };
int main() {
  struct Pos pos;
  pos.x = glob_x;
  pos.y = 0x43;
  print_pos(&pos);
}
```

X: 0x42
Y: 0x43

Original code

```c
int main() {
  int v1; // [rsp+0h] [rbp-20h]
  int v2; // [rsp+4h] [rbp-1Ch]

  v1 = dword_201
  v2 = 0x43;
  print_pos(&v1);
}
```

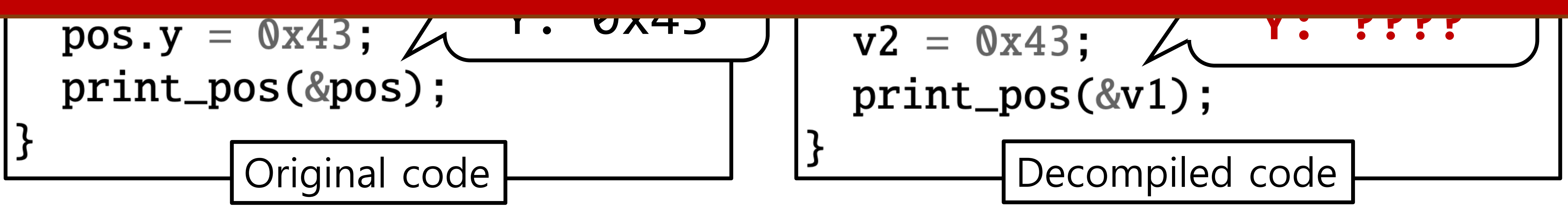
X: 0x42
Y: ????

Decompiled code

The Convergence of Source Code and Binary Vulnerability Discovery - A Case Study (AsiaCCS ‘22)

- Feed decompiled code from Hex-Rays to CodeQL and joern.
- Compared to tools with the original source code, tools with decompiled code got **less TP** and **more FP** because..

Previous source code analysis tools are binary-unaware!



QueryX: Symbolic Query on Decompiled Code

- Static analysis + Domain knowledge (Query)

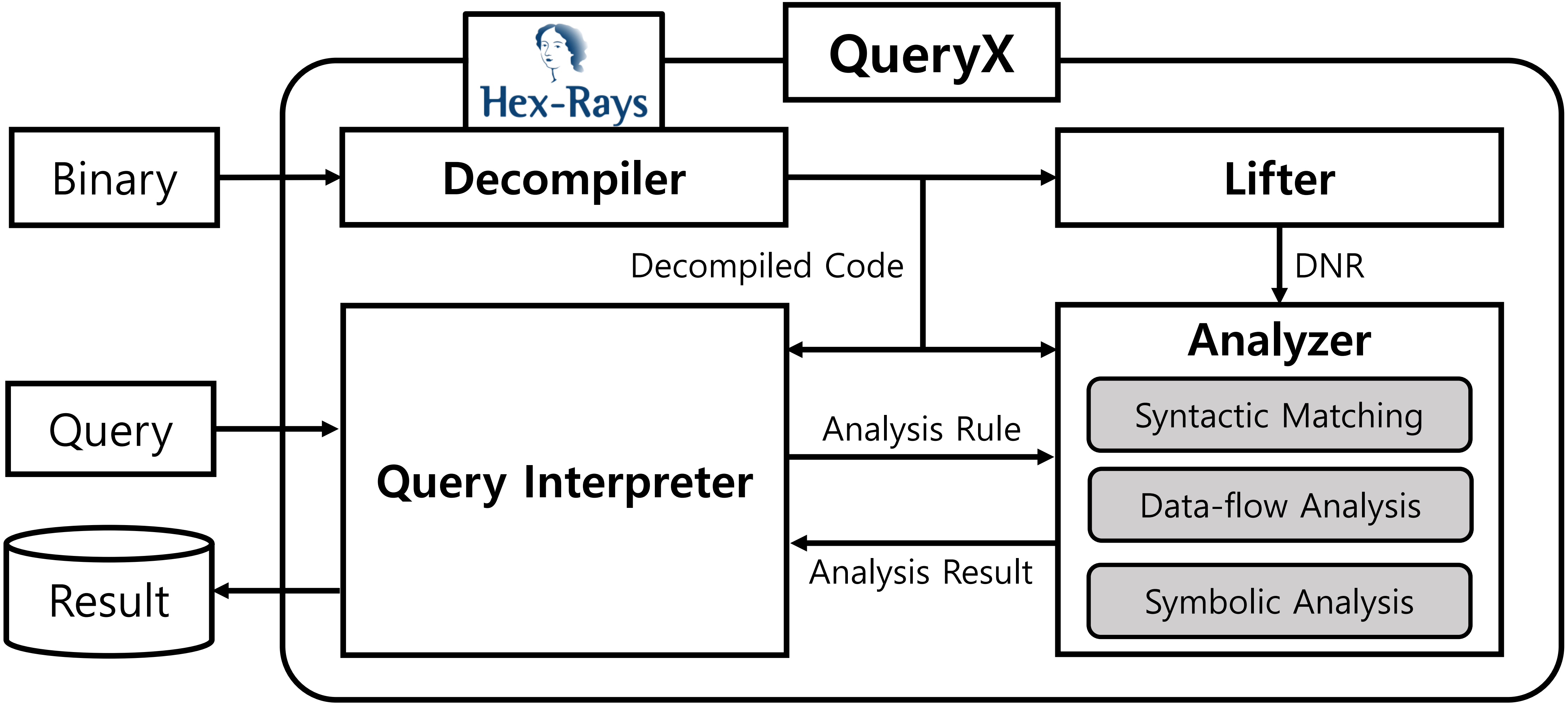
	Input	Analysis objects	Syntactic	Data-flow	Symbolic
CodeQL [22]	Source	Source	✓	✓	×
joern [76]	Source	Source	✓	✓	×
Sys [7]	Source	LLVM IR	✓	✓	✓
angr [56]	Binary	VEX IR	✓	✓	✓
BAP [9]	Binary	BIL	✓	✓	✓
QUERYX	Binary	Decompiled code	✓	✓	✓

Goals & Approaches of QueryX

1. Binary-aware analysis on decompiled code
   - Analysis based on our new IR, DNR
2. Analyst-friendly symbolic query
   - Symbolic query based on decompiled code and callbacks
   - JavaScript-like query
3. Scalable analysis with analyst-friendly symbolic query
   - Under-constrained symbolic execution
   - CFG reduction based on callbacks and their dependencies

   => Check our paper

QueryX Architecture



Goals & Approaches of QueryX

1. Binary-aware analysis on decompiled code
    - Analysis based on our new IR, DNR

2. Analyst-friendly symbolic query
    - Symbolic query based on decompiled code and callbacks
    - JavaScript-like query

3. Scalable analysis with analyst-friendly symbolic query
    - Under-constrained symbolic execution
    - CFG reduction based on callbacks and their dependencies

Decompiler-Neutral Representation (DNR)

```
int main() {
  int v1; // [rsp+0h] [rbp-20h]
  int v2; // [rsp+4h] [rb

  // ISSUE1: Binary-embedded data
  v1 = dword_2010A4;
  v2 = 0x43;

  //ISSUE2: Binary-dependent code
  print_pos(&v1);
}
```

Resolve by adding Program Data to DNR

Missing global variable information (e.g, initial value, RWX permission)

Decompiler-Neutral Representation (DNR)

```
int main() {
  int v1; // [rsp+0h] [rbp-20h]
  int v2; // [rsp+4h]

  // ISSUE1: Binary-en
  v1 = dword_2010A4;
  v2 = 0x43;

  //ISSUE2: Binary-de
  print_pos(&v1);
}
```

Resolve by considering memory layout while lifting

```
real binary (w/ memory layout):
  X: 0x42
  Y: 0x43

w/o memory layout:
  X: 0x42
  Y: 0xff // undefined value
```

Decompiler-Neutral Representation (DNR)

```
int main() {
   int v1; // [rsp+0h] [rbp-20h]
   int v2; // [rsp+4h] [rbp-1Ch]

   // ISSUE1: Binary-embedded data
   v1 = dword_2010A4;
   v2 = 0x43;

   //ISSUE2: Binary-dependent code
   print_pos(&v1);
}
```

```
stack = Alloc(0x8)
// v1: @(stack + 0x0)
// v2: @(stack + 0x4)

Store(stack,
      Load(ProgAddr(0x2010a4), 4),
      4)
Store (stack + 0x4, 0x43, 4)

Call (print_pos, [|stack|])
```

Decompiler-Neutral Representation (DNR)

So, should we write queries based on DNR?

We can write queried based on **decompiled code** because **DNR contains which decompiled code is lifted to the corresponding DNR.**

Goals & Approaches of QueryX

1. Binary-aware analysis on decompiled code
   - Analysis based on our new IR, DNR
2. Analyst-friendly symbolic query
   - Symbolic query based on decompiled code and callbacks
   - JavaScript-like query
3. Scalable analysis with analyst-friendly symbolic query
   - Under-constrained symbolic execution
   - CFG reduction based on callbacks and their dependencies

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
char src[0x100];
...
buf = malloc(size);
if (cond)
  memcpy(buf, src, 0x100);
```

The copy size of memcpy can be greater than the size of dst.

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node, "malloc"))
    setCallback(node, collectAlloc, "alloc");

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}
```

Perform symexec from the entry of **func** based on **symRule**

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node, "malloc"))
    setCallback(node, collectAlloc, "alloc");

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}
```

Recursively traverse AST nodes in **func** and call **symRule**.

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node, "malloc"))
    setCallback(node, collectA

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}
```

Register **collectAlloc** as callbacks of **malloc** call nodes

Register **checkMemcpy** as callbacks of **memcpy** call nodes

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node, "malloc"))
    setCallback(node, collectA

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}
```

Save allocated addresses and their sizes to the current symstate.

Check whether the copy size can be greater than the size of dst.

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
function collectAlloc (node, state) {
  // node: AST node of malloc call
  // state: Current symbolic state
  var addr = state.getValue(node);
  var size = state.getValue(node.args[0]);
  if (state.allocs == undefined) state.allocs = [];
  // Store allocated address and its size
  state.allocs.push({addr: addr, size: size});
}
```

QueryX's Symbolic Query Example (1)

- A simple query for finding simple heap overflow

```
function checkMemcpy (node, state) {
  var dst = state.getValue(node.args[0]);
  var size = state.getValue(node.args[2]);
  for (var alloc of state.allocs) {
    if (dst.includes(alloc.addr)
      && state.isSAT(alloc.size < size)) {
      print("Overflow detected");
    }
  }
}
```

Find the size of dst

Check whether the copy size can be greater than the size of dst **under the current path constraints**.

QueryX's Symbolic Query Example (1)

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node, "malloc"))
    setCallback(node, collectAlloc, "alloc");

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}

function collectAlloc (node, state) {
  // node: AST node of malloc call
  // state: Current symbolic state
  var addr = state.getValue(node);
  var size = state.getValue(node.args[0]);
  if (state.allocs == undefined) state.allocs = [];
  // Store allocated address and its size
  state.allocs.push({addr: addr, size: size});
}

function checkMemcpy (node, state) {
  var dst = state.getValue(node.args[0]);
  var size = state.getValue(node.args[2]);
  for (var alloc of state.allocs) {
    if (dst.includes(alloc.addr)
      && state.isSAT(alloc.size < size)) {
      print("Overflow detected");
    }
  }
}
```

QueryX's Symbolic Query Example (2)

- A query for finding heap overflow due to integer overflow in Windows kernel

```
__int64 __fastcall RtlpCreateServerAcl(...) {
  unsigned short AclSize = 8;
  ...
  if ((_WORD)AceCount) {
    for (i = 0; i < AceCount; i++) {
      AclSize += Aces[i]->AceSize;
      ...
  }
  buffer = E locatePoolWithTag(PagedPool, AclSize, 'cAeS');
  ...
  AclOffset =
  for (i = 0;                                  {
    ...
    memcpy (buffer + AclOffset, Aces[i], Aces[i]->AceSize);
    AclOffset = AclOffset + Aces[i]->AceSize;
  }
}
```

① 2byte type && not constant

② dst is from alloc result

③ can be greater than AclSize

CVE-2021-31979 (Heap overflow due to integer overflow)

```
function symRule (node) {
  if (isCall(node,"ExAllocatePoolWithTag")
     && node.args[1].type.size == 2
     && !isConstant(node.args[1]))
    setCallback(node, collectAlloc, "alloc");

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}

for (func in prog.functions)
  symExec(func, symRule);
```

① (marks the `node.args[1].type.size == 2 && !isConstant(node.args[1])` condition)

QueryX's Symbolic Query Example (2)

```
NTSTATUS __fastcall NtfsSetDispositionInfo(...) {
  ...
  length = volumeName->Length + dirName->Length
           + fileName->Length;
```

①

Idx	CVE	Program	Function	Bug Type	Bounty
1	CVE-2021-41370	ntfs.sys	NtfsSetShortNameInfo	Heap overflow	$20,000
2	CVE-2021-41378	ntfs.sys	NtfsSetDispositionInfo	Heap overflow	$20,000
3	CVE-2021-43229	ntfs.sys	TxfAllocateAndStoreNameForTxfLogging	Heap overflow	$20,000
4	CVE-2021-43230	ntfs.sys	TxfAllocateFullFilePathForChangeNotify	Heap overflow	$20,000
5	CVE-2021-43231	ntfs.sys	NtfsRenameToPrivateDir	Heap overflow	$20,000
6	CVE-2021-41367	ntfs.sys	TxfOpenFileProcessing	Heap overflow	$20,000
7	CVE-2022-23293	fastfat.sys	FatSetFullNameInFcb	Heap overflow	$20,000

```
  ...
}
```

CVE-2021-41378, one of heap overflow bugs QueryX found

angr vs QueryX in the Example

Binary IRs do not have such information
(e.g., type, constant value..)

```python
    self.blocks = {}
    self.cut_succs = []
    self.ends = ends
    self.nodes = nodes
    self.timeout = 600
    self.start_time = time.time()

  def filter(self, simgr, state, **kwargs):
    addr = state.addr
    if state in self.cut_succs:
      self.cut_succs.remove(state)
      return "end"
```

```python
    if is_include(dst, alloc[0]):
      if state.satisfiable(extra_constraints=[size > alloc[1]]):
        LOG.info ("gotcha!!")
        LOG.info (hex(alloc[2]) + ' alloc size '+  str(alloc[1]))
        LOG.info (hex(state.addr) + ' copy size ' + str(size))
        global DONE
        DONE = True
        return

def inspect_bf_instr (state):
  addr = state.solver.eval(state.inspect.instruction)
  if addr in hook_instrs:
```

```
for (func in prog.functions)
  symExec(func, symRule);

function symRule (node) {
  if (isCall(node,"ExAllocatePoolWithTag")
    && node.args[1].type.size == 2
    && !isConstant(node.args[1])) {
    setCallback(node, collectAlloc, "alloc");

  if (isCall(node, "memcpy"))
    setCallback(node, checkMemcpy, "copy");
}
```

①

angr: 267 LoC, QueryX: 33 LoC

```python
def collect_args (state, addr):
  state.queryx.args = [state.regs.rcx, state.regs.rdx, state.regs.r8]

def hook_alloc (state, addr):
  if len(state.queryx.args) == 0: return
  size = state.queryx.args[1]
  state.solver.add (state.regs.rax != 0)
  state.queryx.allocs.append((state.regs.rax, size, state.addr))
  state.queryx.args = []

def hook_memmove (state, addr):
  if len(state.queryx.args) == 0: return
  dst = state.queryx.args[0]
```

```python
  addr = func.get_call_target (cs_block_addr)
  if addr in func_table:
    callee = func_table[addr]
    for (name, tag, filt, hook) in hooks:
      if name in callee:
        cs_block = func.get_block(cs_block_addr)
        if filt (cs_block):
          call_addr = cs_block.instruction_addrs[-1]
          end_addr = cs_block_addr + cs_block.size
          hook_instrs[end_addr] = hook
          hook_instrs[call_addr] = collect_args
          ends.add(end_addr)
          if tag not in tags:
            tags[tag] = []
```

```
  var dst = state.getValue(node.args[0]);
  var size = state.getValue(node.args[2]);
  for (var alloc of state.allocs) {
    if (dst.includes(alloc.addr)
      && state.isSAT(alloc.size < size)) {
      print("Overflow detected");
    }
  }
}
```

Evaluation (1-day)

- Comparison on the Dataset of Mantovani *et al.* ("The convergence of source code and binary vulnerability discovery–a case study", AsiaCCS '22)

Less FP

More TP

		QUERYX					joern	CodeQL
Bug		**Total**	**Detected**			**tected**	**Detected**	**Detected**
CVE-2017-1000249	5s	1	✓	3m	1	✓	✓	✓
CVE-2013-6462	30s	1	✓	6s	2	✓	✓	✓
BUG-2012	-	-	×	-	-	×	×	×
CVE-2017-6298	4s	1	✓	24s	1	✓	✓	✓
CVE-2018-11360	13h	13	✓	> 24h	303	✓	×	×
CVE-2017-17760	2.5m	19	✓	1.8h	30	✓	✓	×
CVE-2019-19334	40s	1	✓	1.1h	1	✓	✓	✓
CVE-2019-1010315	20m	4	✓	13m	4	×	✓	✓
BUG-2010	3s	1	✓	12m	2	✓	×	×
BUG-2018	2.6h	2	✓	> 24h	18	✓	✓	×

Evaluation (1-day)

- Comparison on the Dataset of Mantovani *et al.* ("The convergence of … v … discovery–a case stu…

Lack of high-level info (e.g., type)

Binary-unaware

	QueryX			angr			joern	CodeQL
Bug	**Time**	**Total**	**Detected**	**Time**	**Total**	**Detected**	**Detected**	**Detected**
CVE-2017-1000249	5s	1	✓	3m	1	✓	✓	✓
CVE-2013-6462	30s	1	✓	6s	2	✓	✓	✓
BUG-2012	-	-	×	-	-	×	×	×
CVE-2017-6298	4s	1	✓	24s	1	✓	✓	✓
CVE-2018-11360	13h	13	✓	> 24h	303	✓	×	×
CVE-2017-17760	2.5m	19	✓	1.8h	30	✓	✓	×
CVE-2019-19334	40s	1	✓	1.1h	1	✓	✓	✓
CVE-2019-1010315	20m	4	✓	13m	4	×	✓	✓
BUG-2010	3s	1	✓	12m	2	✓	×	×
BUG-2018	2.6h	2	✓	> 24h	18	✓	✓	×

Evaluation (0-day)

- QueryX found 15 previous unknown vulnerabilities including **10 CVEs** and earned **$180,000** from MS bug bounty program.

Idx	CVE	Program	Function	Bug Type	Bounty
1	CVE-2021-41370	ntfs.sys	NtfsSetShortNameInfo	Heap overflow	$20,000
2	CVE-2021-41378	ntfs.sys	NtfsSetDispositionInfo	Heap overflow	$20,000
3	CVE-2021-43229	ntfs.sys	TxfAllocateAndStoreNameForTxfLogging	Heap overflow	$20,000
4	CVE-2021-43230	ntfs.sys	TxfAllocateFullFilePathForChangeNotify	Heap overflow	$20,000
5	CVE-2021-43231	ntfs.sys	NtfsRenameToPrivateDir	Heap overflow	$20,000
6	CVE-2021-41367	ntfs.sys	TxfOpenFileProcessing	Heap overflow	$20,000
7	CVE-2022-23293	fastfat.sys	FatSetFullNameInFcb	Heap overflow	$20,000
8	CVE-2022-30162	win32kfull.sys	NtUserSetClassLongPtr	Kernel Address Disclosure	-
9	CVE-2020-17041	PrintConfig.dll	CopyFileFromPrinterData	Path Traversal	$20,000
10	CVE-2020-17042	PrintConfig.dll	UniDrvUI::PConcatFilename	Path Traversal	$20,000
11 – 15	-	Automotive	REDACTED	Out-of-bound Access	-

More in the paper

- Scalable analysis with analyst-friendly symbolic query
- 4 kinds queries for finding 0-days
- More detail comparison against other tools
- Other evaluations and details

Thank you