Blaze: A Framework for Interprocedural Binary Analysis

Matthew Revelle
Matt Parker
Kevin Orr

Workshop on Binary Analysis Research (BAR) 2023
March 3, 2023

KUDU DYNAMICS

This material is based upon work supported by the
Defense Advanced Research Projects Agency (DARPA) and the Naval Information Warfare Center (NIWC) under Contract No. N6600122C4018.
Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA and NIWC.

Interprocedural Binary Analysis

Example

```
cgiFormString(0x1aaa4, &var_910, 0x200);  // {"user"}
cgiFormString(0x1aaac, &var_710, 0x200);  // {"passwd"}
cgiFormString(0x1accc, &var_510, 0x200);  // {"start"}
cgiFormString(0x1acd4, &var_310, 0x200);  // {"count"}
if (data_2c2e4 != 0)
{
    sub_1194c("name --> [%s][%s]\n", &var_910);
    sub_1194c("count -->[%s][%s]\n", &var_510);
}
int32_t r0_9;
if (sub_160a8(&var_910, &var_710) == 0)
{
    r0_9 = sub_119a4();
}
else
{
    sprintf(&var_110, "sqlsearch -t video -o %s -r %s,%…"  , "/tmp/video_list.xml", &var_510, &var_310);
    if (data_2c2e4 != 0)
    {
        sub_1194c("cmd[%s]\n", &var_110);
    }
    system(&var_110);
```

User input

Authentication check

Command injection

https://www.exploit-db.com/exploits/43435

Interprocedural Binary Analysis

Example

```
if (((uint32_t)*(int8_t*)arg2) != 0)
{
    __b64_pton(arg2, &var_1090, strlen(arg2));
    if (data_2c2e4 != 0)
    {
        sub_1194c("pwd [%s]\n", arg2);
        sub_1194c("pwd decode[%s]\n", &var_1090);
    }
}
int32_t r0_4 = strcmp(arg1, "mydlinkBRionyg");
int32_t r0_6;
void* r0_7;
if (r0_4 == 0)
{
    r0_6 = strcmp(&var_1090, "abc12345cba");
    if (r0_6 == 0)
    {
        r0_7 = 1;
    }
}
```

Decode Base64 encoded password

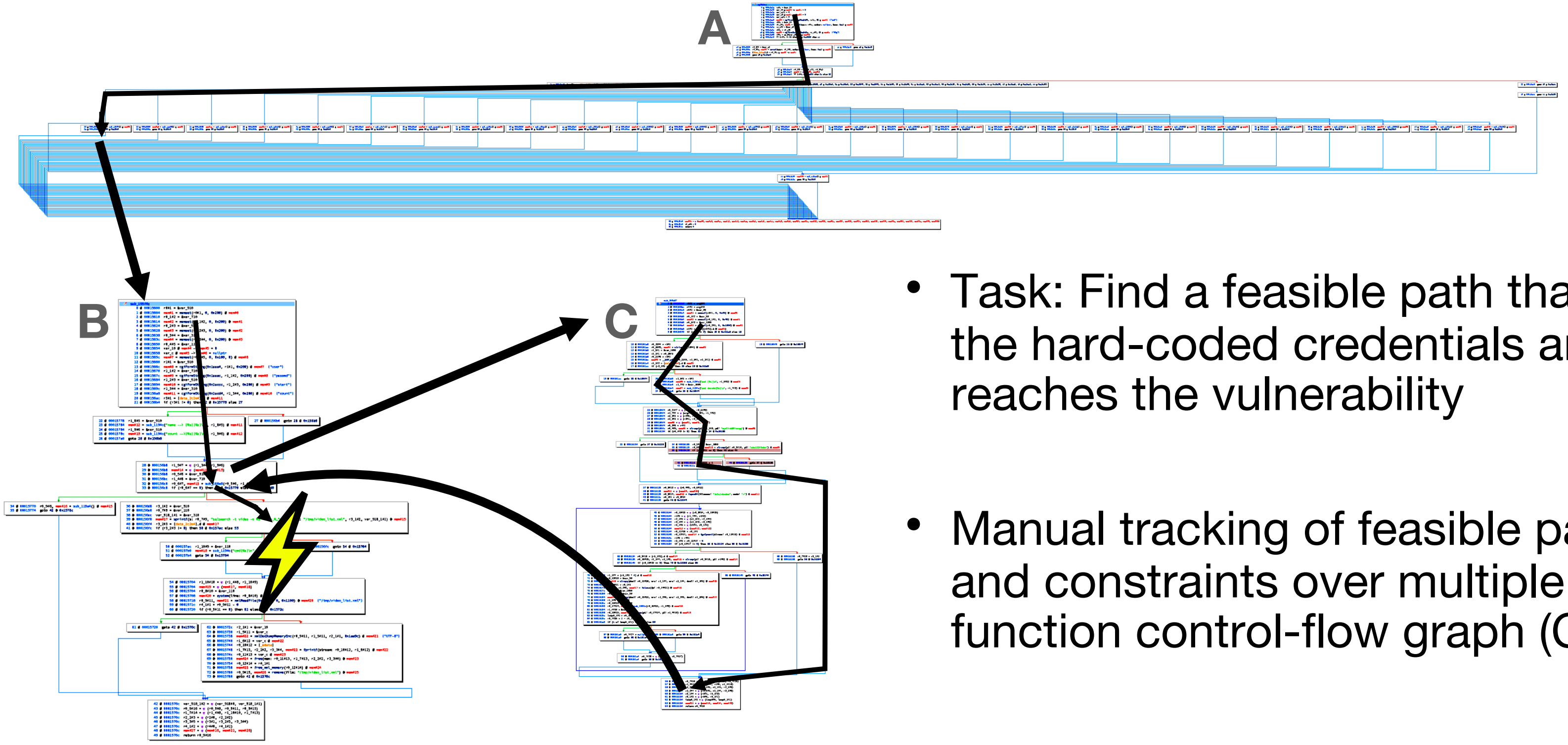
Hard-coded login credentials

`is_authenticated` variable

https://www.exploit-db.com/exploits/43435

Interprocedural Binary Analysis

Motivation

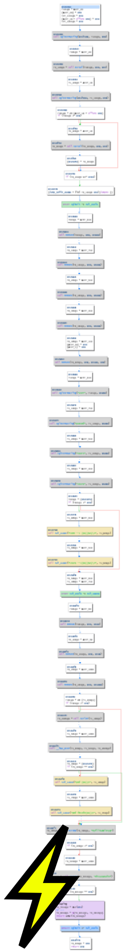


- Task: Find a feasible path that uses the hard-coded credentials and reaches the vulnerability
- Manual tracking of feasible paths and constraints over multiple function control-flow graph (CFGs)

Interprocedural Binary Analysis

Motivation

- Number of paths in a function can be very large, but often many are infeasible
- Automated removal of these paths can have a big impact
- Can use automated analyses to automatically simplify an interprocedural CFG as it is constructed



Interprocedural Binary Analysis

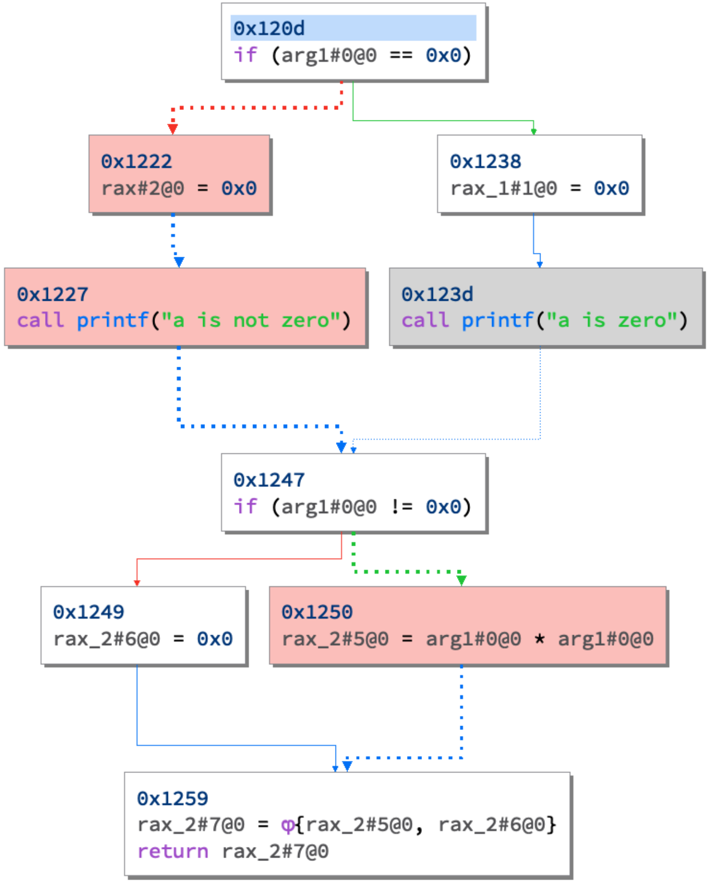
Problem Statement

Use automated analyses to interactively help reverse engineers manage the complexity of analyzing program binaries for vulnerabilities.

Blaze

Static Analysis Framework

- Built around ***interprocedural control-flow graphs (ICFGs)*** and a typed intermediate language ***(PIL)***
- Supports symbolic analysis through satisfiability modulo theories (SMT) solvers
- Open source, written in Haskell
- Support for many executable formats and architectures via BINARYNINJA and GHIDRA

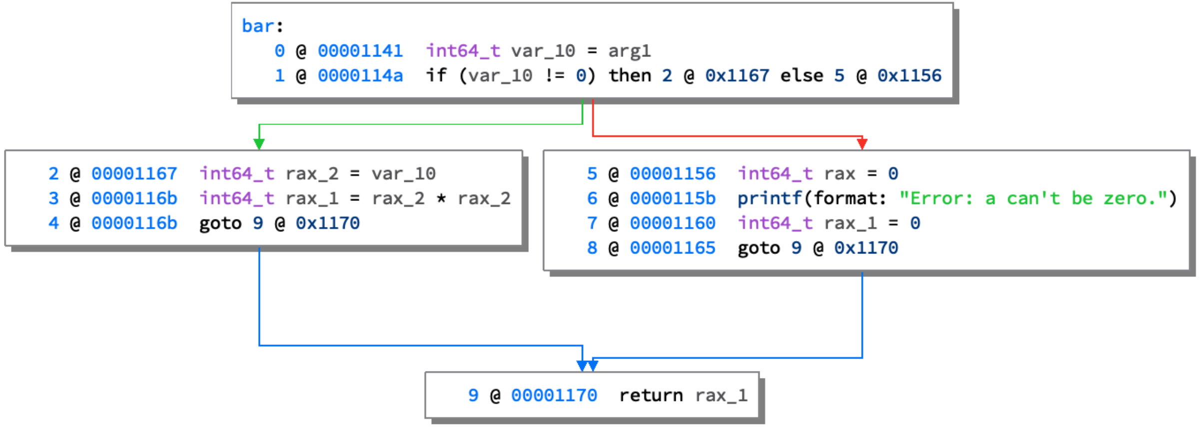


"Haskell logo." https://www.haskell.org/img/haskell-logo.svg
"Binary Ninja logo." https://www.cyberus-technology.de/assets/images/products/tycho/logo
"Ghidra logo." https://ghidra-sre.org/images/GHIDRA_1.png

Control-Flow Graphs

(CFGs)

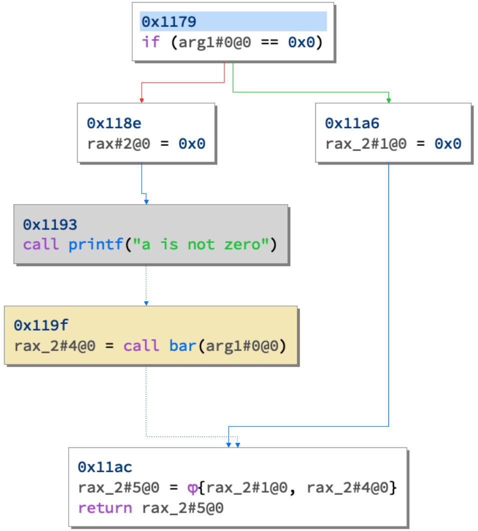
```
long bar(long a) {
  if (a == 0) {
    printf("Error: a can't be zero.");
    return 0;
  }
  else {
    return a * a;
  }
```



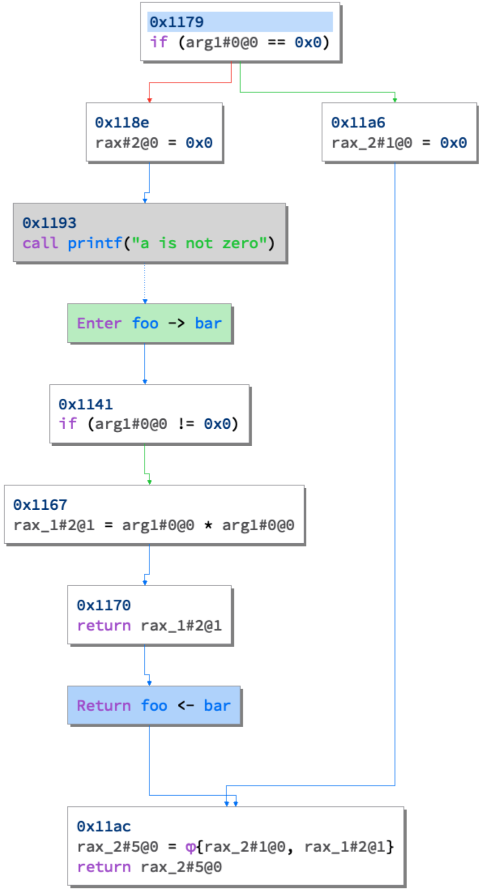
Interprocedural Control-Flow Graphs (ICFGs)

- Control-flow graphs (CFGs) that may span across function calls
- In ICFGs, function calls are expandable ***call nodes***
- ICFGs can be constructed programmatically or by user interaction

Before expansion



Call to bar expanded



Satisfiability Modulo Theories

(SMT)

- ***SMT solvers*** can check if a formula is satisfiable
- Support for integers, floats, bit vectors, arrays, and more through theories
- Describe program constraints as a mathematical formula
- Behind the scenes in Blaze, typed PIL statements are used to generate SMT formulas

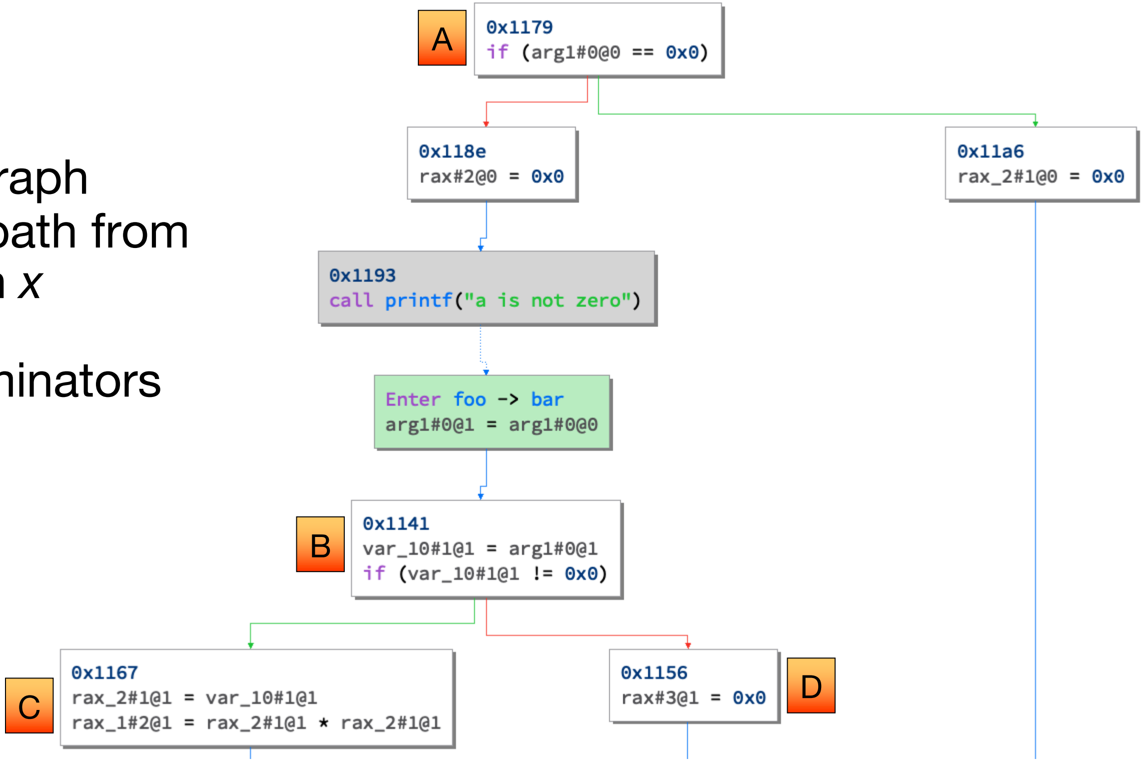
```
(declare-const x Int)
(declare-const y Int)
(assert (< x 10))
(assert (> y 0))
(assert (= x (* y 2)))
(check-sat)
(get-model)
```

```
sat
(
  (define-fun y () Int
    1)
  (define-fun x () Int
    2)
)
```

Dominators

Influence of a Node

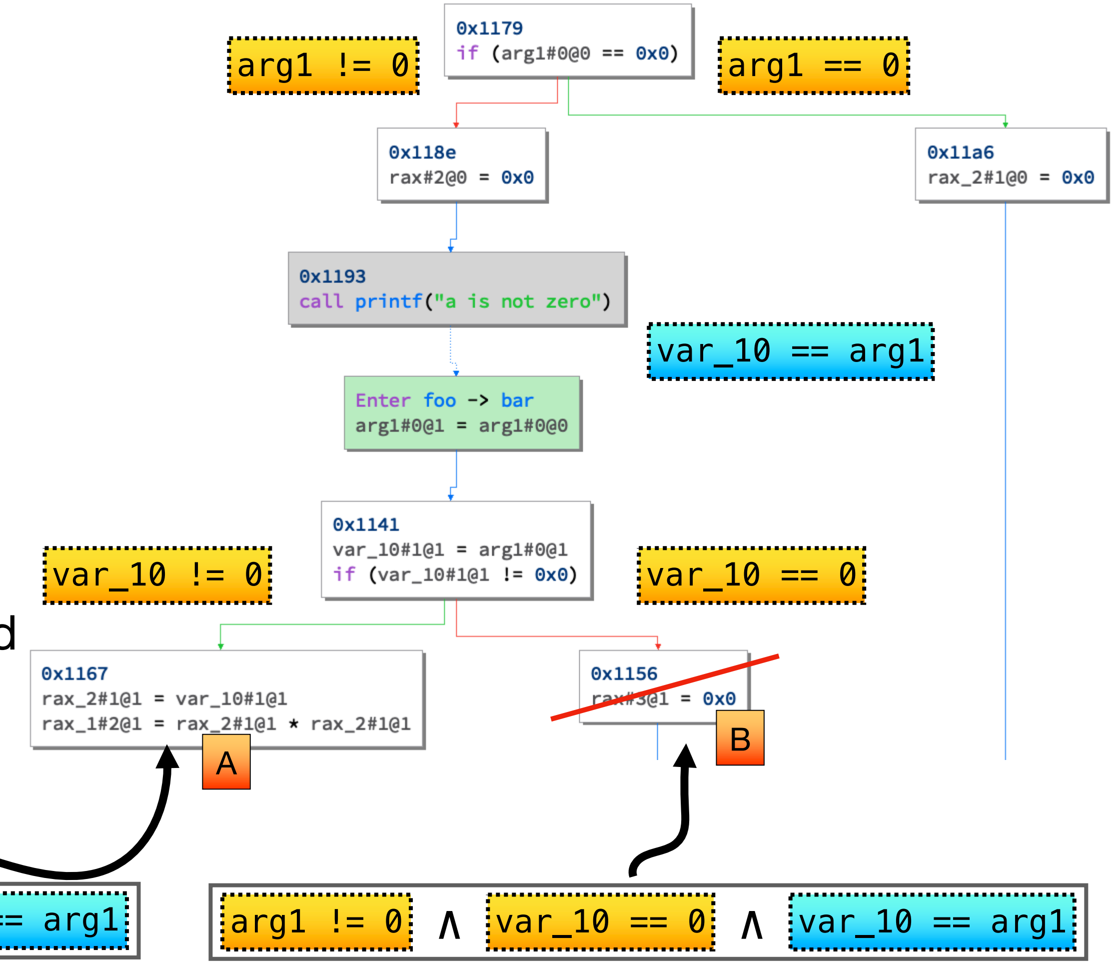
- A node *x* in a control-flow graph ***dominates*** node *y* if every path from the root to *y* passes through *x*
- A node may have many dominators



Branch Contexts

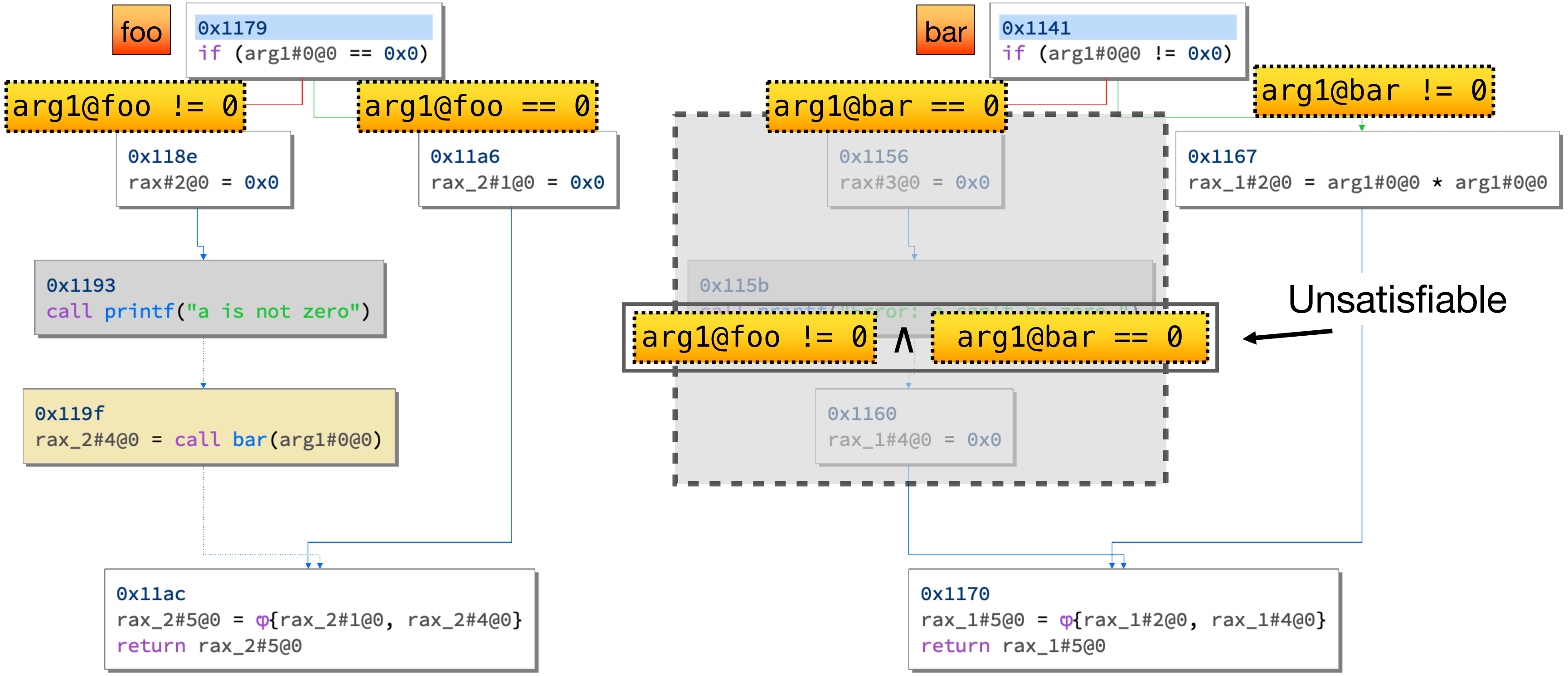
Dominating Constraints

- Nodes dominated by a conditional branch are in a ***branch context***
- Every branch context is associated with a constraint
- Branch contexts can be nested
- Use branch contexts to determine if a node is reachable



Constraint-Driven Transformations

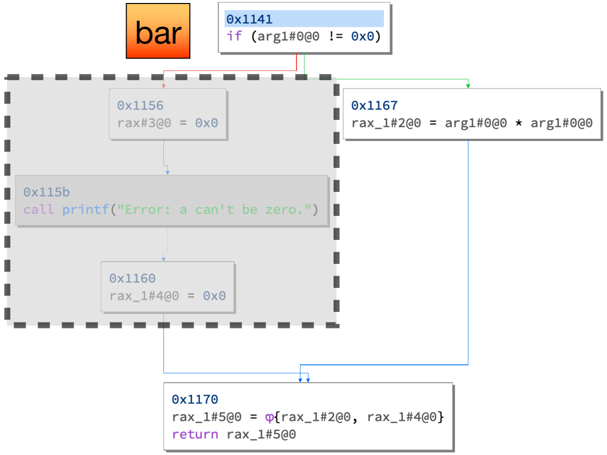
Call Expansion

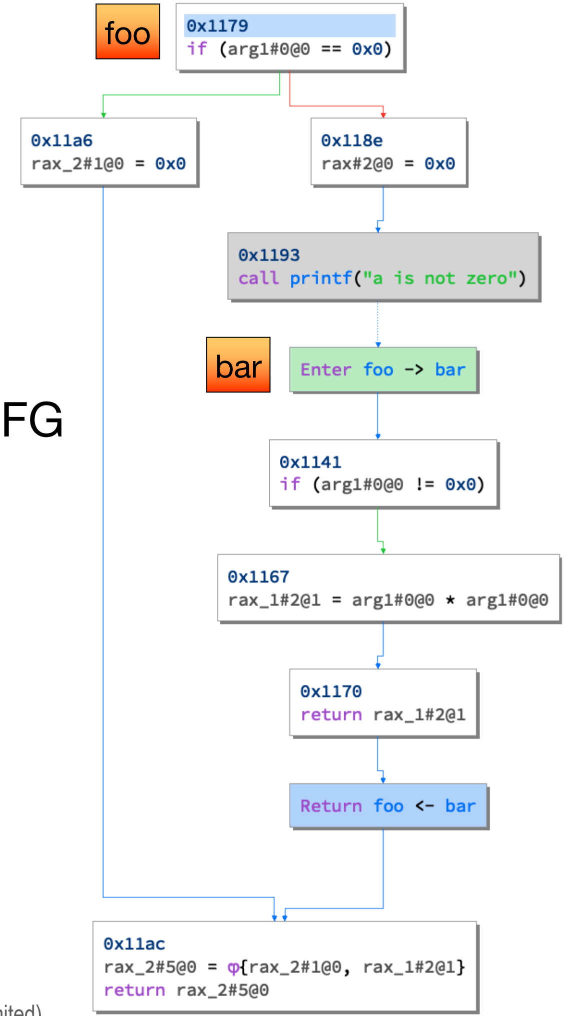


Constraint-Driven Transformations

Call Expansion

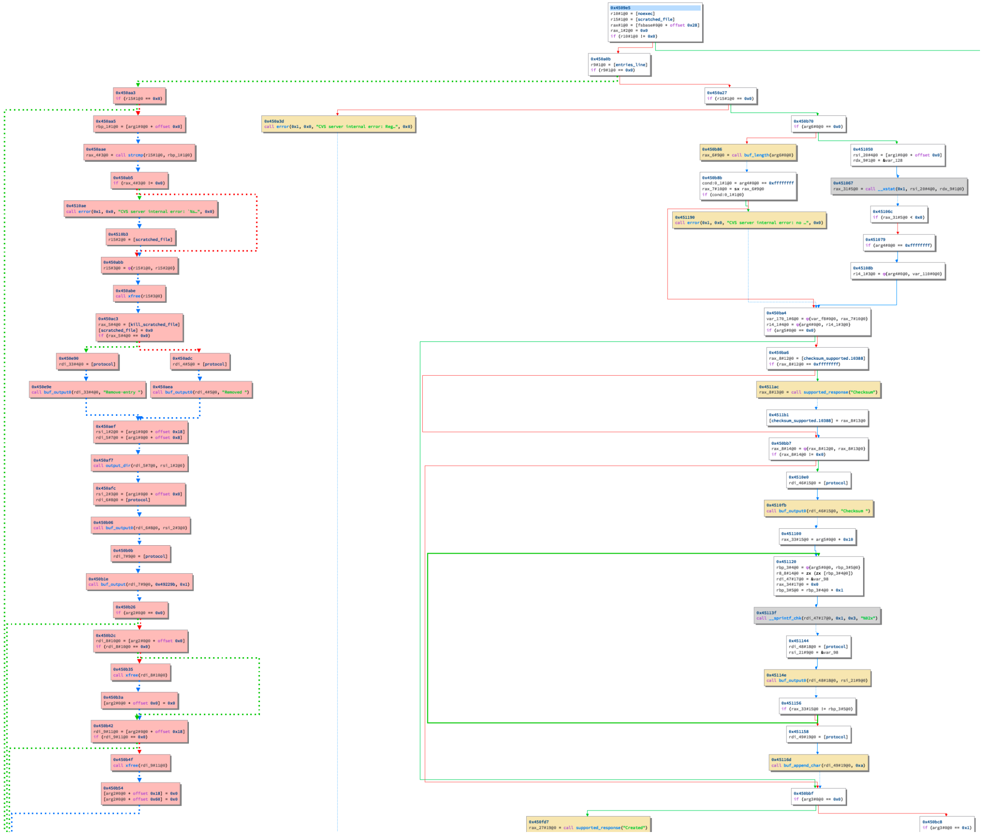
- The call to `bar` is expanded
- Infeasible path is automatically removed from the ICFG



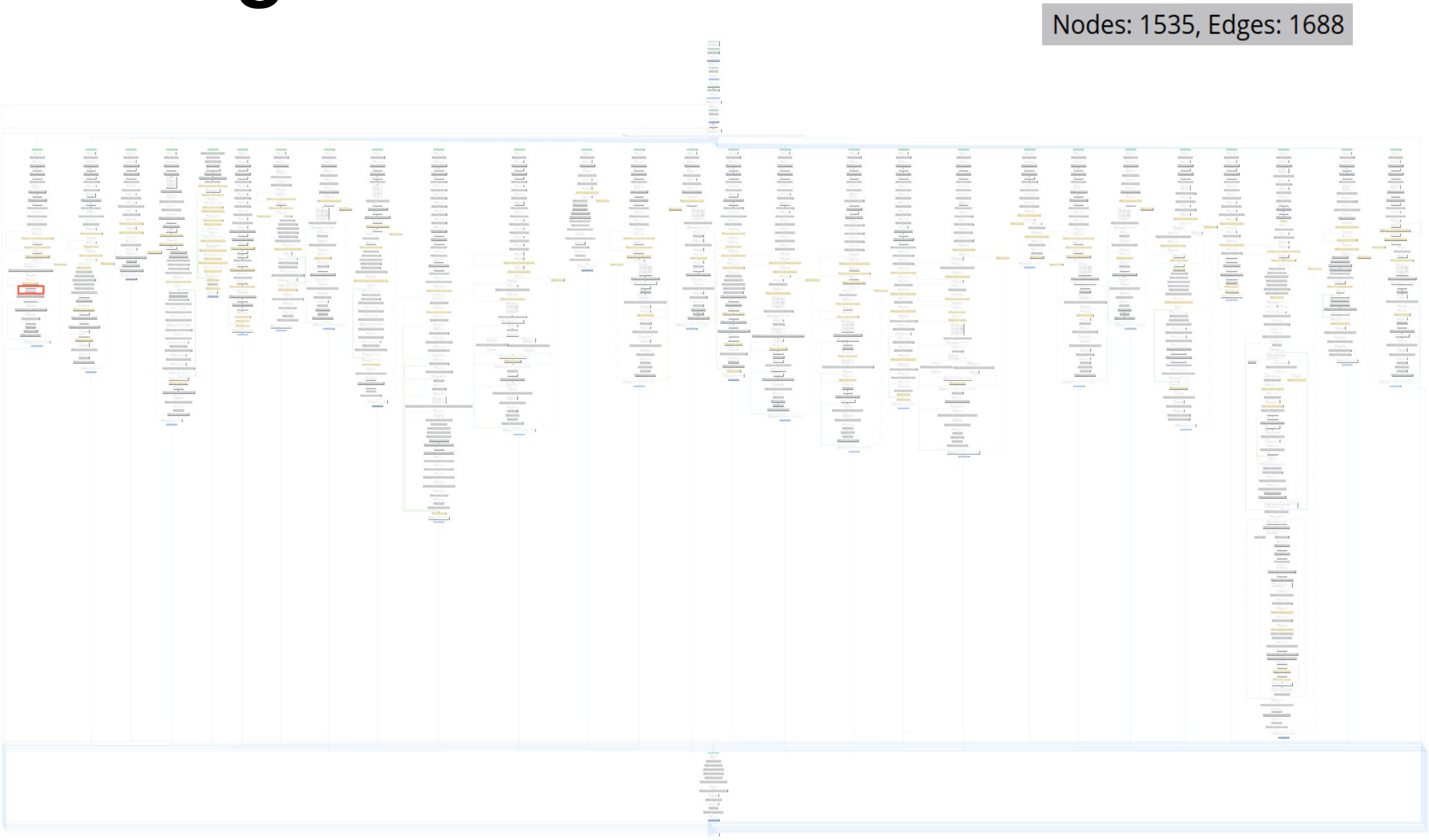


Constraint-Driven Transformations

CVS Example

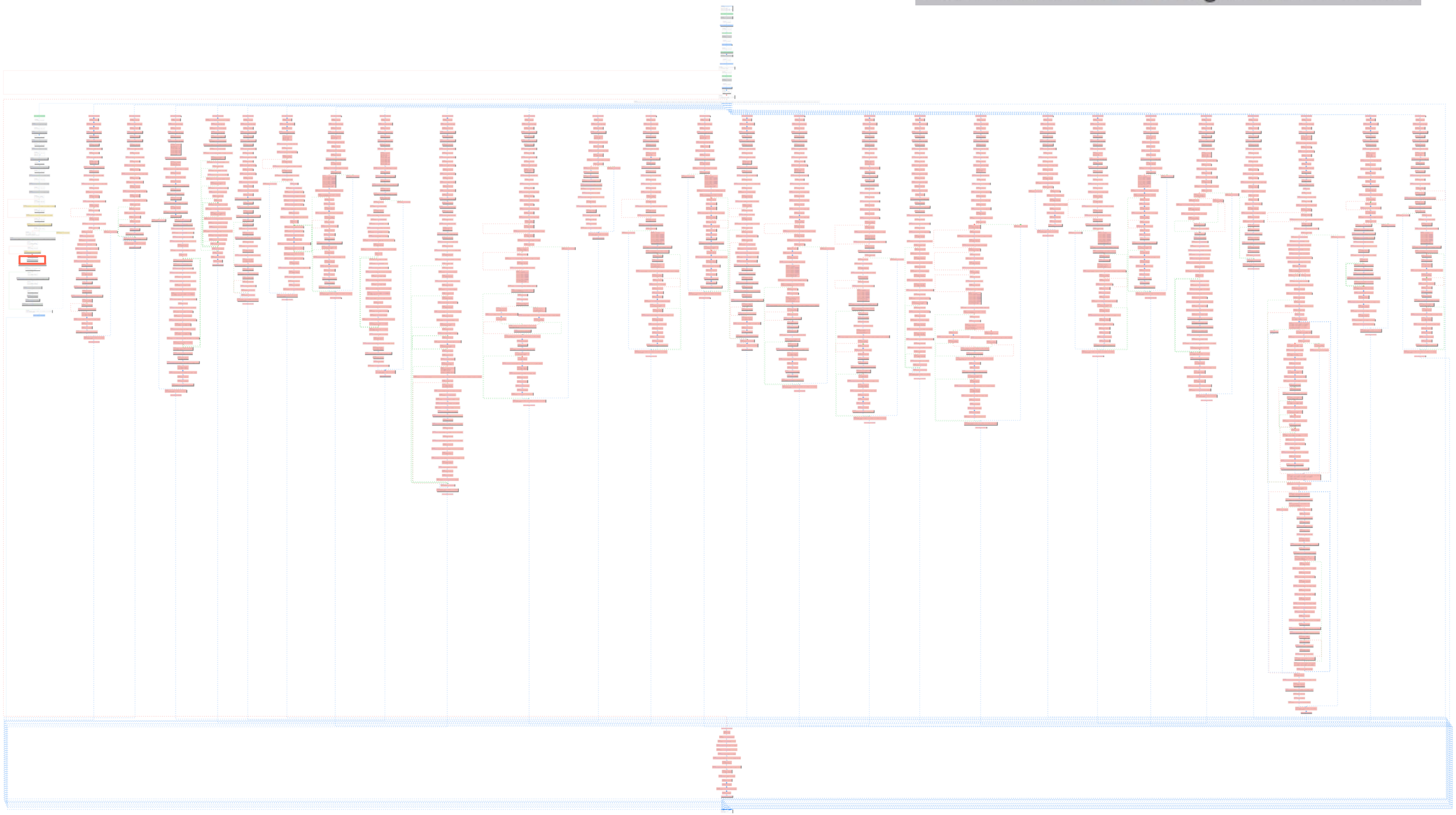


Node/Edge Reduction

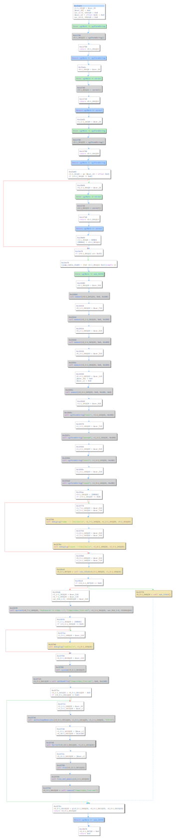


Node/Edge Reduction

Nodes: 1535-1464 = 71, Edges: 1688-1614 = 74



Node/Edge Reduction



Nodes: 71, Edges: 74

Blaze

Implementation

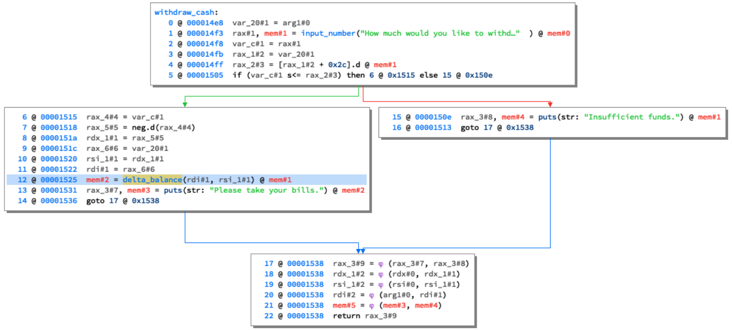
Implementation available at: https://github.com/kudu-dynamics/blaze

Backup Slides

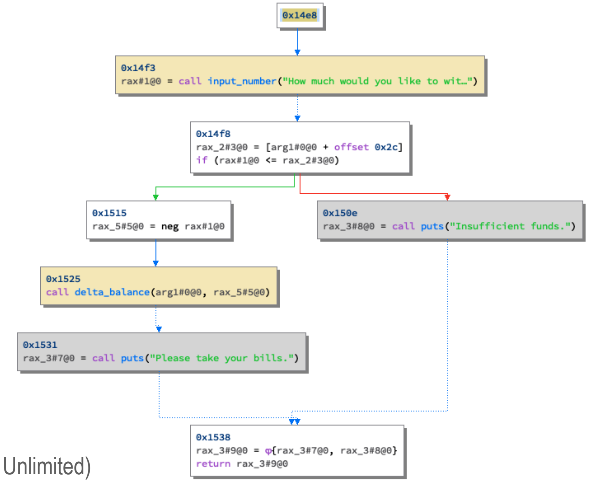
Path Intermediate Language (PIL)

- ICFG basic blocks contain PIL statements
- PIL provides a common target representation for importing
- All analysis algorithms operate on PIL
- PIL has a type system and unification-based checker capable of type inference
- SMT formulas can be generated by PIL statements

BN Medium Level IL (SSA Form)



Blaze PIL



ICFG Interactions

Pruning