Blaze: A Framework for Interprocedural Binary Analysis

Matthew Revelle
Matt Parker
Kevin Orr

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

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(0x1aaa0, &var_710, 0x200); // {"password"}
cgiFormString(0x1acc4, &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,%s", "/tmp/video_list.xml", &var_510, &var_310);
    if (data_2c2e4 != 0)
    {
        sub_1194c("cmd[\%s]\n", &var_110);
    }
    system(&var_110);
}
Interprocedural Binary Analysis

Example

```c
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
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 **Binary Ninja** and **Ghidra**

---

Distribution Statement 'A' (Approved for Public Release, Distribution Unlimited)
Control-Flow Graphs
(CFGs)

```c
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
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

```
1 (declare-const x Int)
2 (declare-const y Int)
3 (assert (< x 10))
4 (assert (> y 0))
5 (assert (= x (* y 2)))
6 (check-sat)
7 (get-model)

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

Distribution Statement ‘A’ (Approved for Public Release, Distribution Unlimited)
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.

**Distribution Statement** 'A' (Approved for Public Release, Distribution Unlimited)
Constraint-Driven Transformations

Call Expansion

foo
0x1179
if (arg1@foo == 0x0)

arg1@foo != 0
0x118e
rax#2@0 = 0x0

arg1@foo == 0
0x11a6
rax_2#1@0 = 0x0

0x1193
call printf("a is not zero")

0x119f
rax_2#4@0 = call bar(arg1@0@0)

bar
0x1141
if (arg1@bar != 0x0)

arg1@bar == 0
0x1156
rax#3@0 = 0x0

0x1167
rax_1#2@0 = arg1@0@0 * arg1@0@0

arg1@bar != 0
0x115b
for:

arg1@bar == 0
0x1160
rax_1#4@0 = 0x0

0x1170
rax_1#5@0 = \phi{rax_1#2@0, rax_1#4@0}
return rax_1#5@0

Unsatisfiable
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

Nodes: 1535, Edges: 1688
Node/Edge Reduction

Nodes: 1535-1464 = 71, Edges: 1688-1614 = 74
Node/Edge Reduction
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
ICFG Interactions

Pruning

0x120d
if (arg1#0 == 0x0)

0x1238
rax_1#1@0 = 0x0

0x1222
rax#2@0 = 0x0

0x123d
call printf("a is zero")

0x1227
call printf("a is not zero")

0x1247
if (arg1#0 != 0x0)

0x1249
rax_2#6@0 = 0x0

0x1250
rax_2#5@0 = arg1#0@0 * arg1#0@0

0x1259
rax_2#7@0 = ϕ{rax_2#5@0, rax_2#6@0}
return rax_2#7@0

Distribution Statement ‘A’ (Approved for Public Release, Distribution Unlimited)