Standing on the Shoulders of Giants

De-obfuscating WebAssembly Using LLVM

Vikas Gupta & Peter Garba
Thales *Cybersecurity & Digital Identity* (CDI)

Decompile: w2c_squanchy_calc_0 - (hikari_bogus2_optimised_llvm_simba_gamba_souper.o)

```
1
2  int w2c_squanchy_calc_0(undefined8 param_1,uint param_2)
3
4  {
5    uint uVar1;
6    int iVar2;
7    int iVar3;
8
9    uVar1 = param_2 & 3;
10   iVar3 = (param_2 ^ 0xbaaad0bf) * (param_2 | 4);
11   if (uVar1 != 2) {
12     iVar3 = (param_2 & 5) * (param_2 + 0xbaaad0bf);
13   }
14   iVar2 = (param_2 | 0xbaaad0bf) * (param_2 ^ 2);
15   if (uVar1 != 0) {
16     iVar2 = (param_2 & 0xbaaad0bf) * (param_2 + 3);
17   }
18   if (uVar1 < 2) {
19     iVar3 = iVar2;
20   }
21   return iVar3;
22 }
23
```

About Us

- Vikas Gupta @su-vikas
  - Senior Security Researcher at *Thales CDI*, previously with *Google*.
  - Masters in information security, OSCP
  - Co-Author OWASP Mobile Security Testing Guide (MSTG)
  - Interests: Reverse engineering, mobile security

- Peter Garba @pgarba
  - Principal Software Security Engineer at *Thales CDI*
  - Product Owner
  - Author of the Thales internal obfuscation Tools
  - Passionate Reverse Engineer at Night

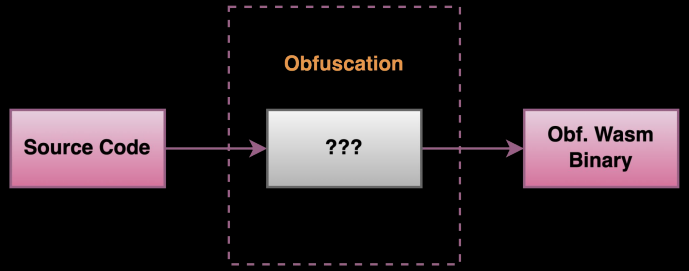
Motivation

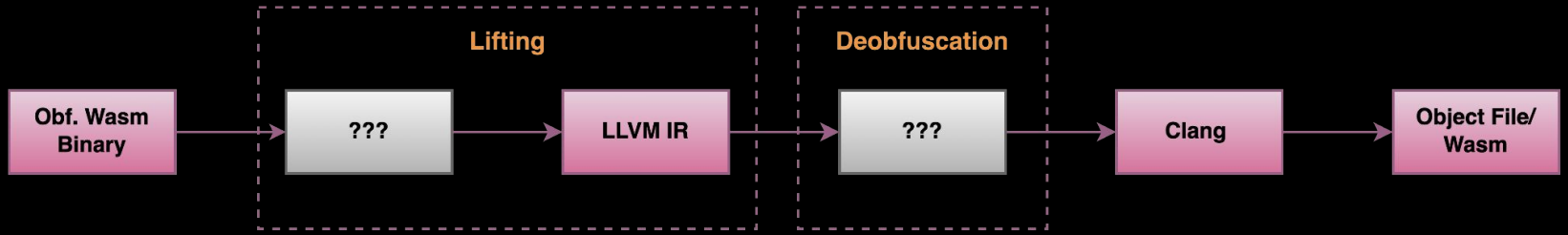

THALES
Building a future we can all trust


Java
node
JS
WA
android
intel
arm
iOS

Problem Statement

1. Is Wasm secure enough for us ?!
2. Obfuscate Wasm binaries
3. Lifting Wasm to LLVM-IR
4. Deobfuscate Wasm binaries and recover the original logic





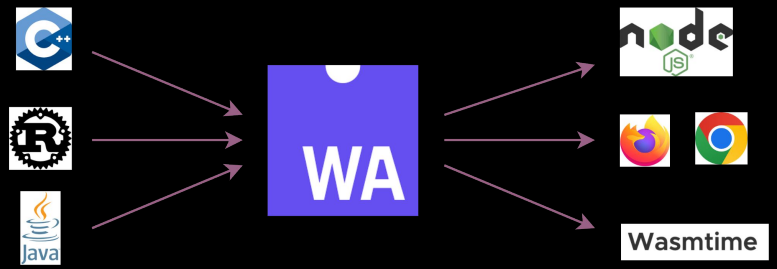
Achievements

- Demonstrating use of existing tooling for WebAssembly
  - Obfuscation
  - Deobfuscation
- Lifting Wasm to LLVM IR - Squanchy
- Automated deobfuscation of WebAssembly

WebAssembly Essentials

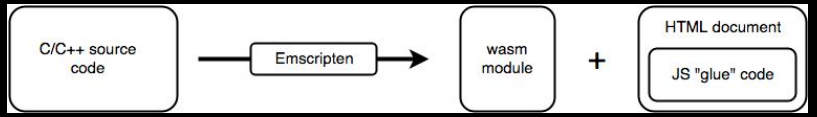
WebAssembly Essentials

- Announced in 2015, a high-performance, secure, and portable compilation target.



- Binaries that are compact and quick to parse.

- Runs in a stack based virtual machine (think JVM)
  - Communicates with host program using well defined exports and imports

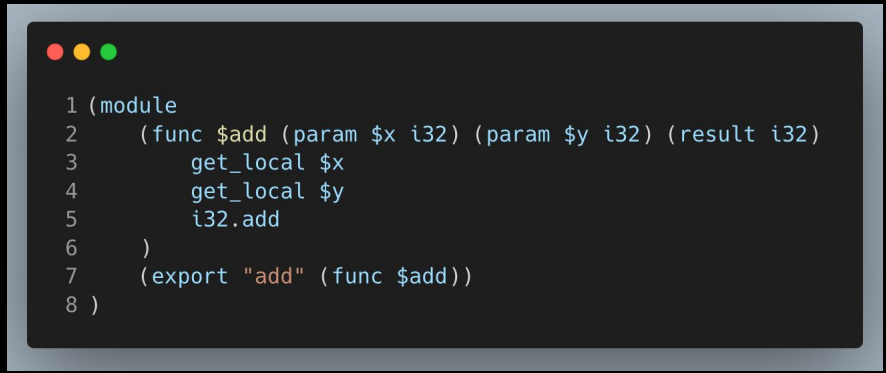


- Compile high level language to WebAssembly

WebAssembly Essentials

- Human readable representation is called *text format* of WebAssembly (*wat*)

- Wasm code is represented by *S-expressions*.



```
(module
    (func $add (param $x i32) (param $y i32) (result i32)
        get_local $x
        get_local $y
        i32.add
    )
    (export "add" (func $add))
)
```

WebAssembly Essentials

- Each Wasm program is a single file of code - Module.
- Module is organized in sections

SECTION	CODE	DESCRIPTION
Type	1	Contains a list of function signatures used by functions defined and called by the module. Each signature has an index, and can be used by multiple functions by specifying that index.
Import	2	Contains the names and types of objects to be imported.
Functions	3	The declarations (including the index of a signature specified in the Type Section) of the functions defined in this module.
Table	4	Contains details about function tables.
Memory	5	Contains details about memory.
Global	6	Global declarations.
Exports	7	Contains the names and types of objects and functions that will be exported.
Start	8	Specifies a function that will be called on Module start-up.
Elements	9	Table initialization information.
Code	10	The WebAssembly instructions that make up the body of each function.
Data	11	Memory initialization information.

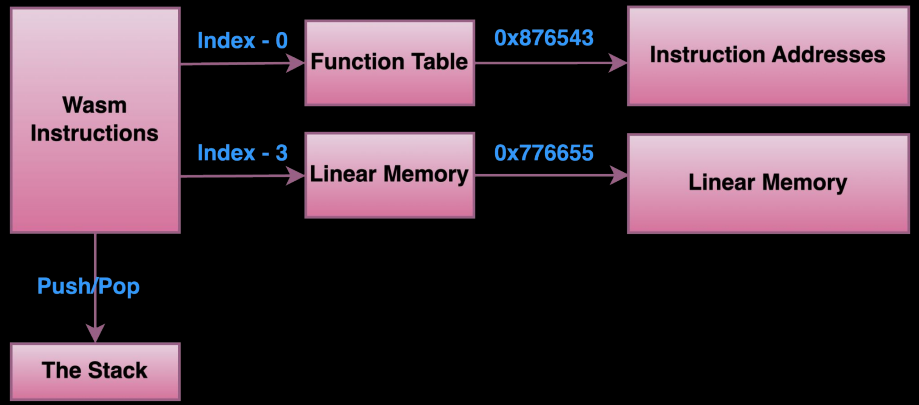
WebAssembly Essentials

- Primitives
  - Integer types
    - `i32`: 32-bit integer
    - `i64`: 64-bit integer
    - `f32`: 32-bit float
    - `f64`: 64-bit float
  - Vector types
    - `v128`: 128 bit vector

Memory Space	Read-write	List of linear memories, containing data elements, accessible by the load/store operators
Table Space	Read-only	Containing function pointers used for indirect invocation of functions
Global Space	Read-only or Read-Write	Containing read-write immutable global variables
Function Space	Read-only	Containing all imported and internal functions (and their bodies)

- Indexed Spaces
  - Items can be accessed by a 0-based integer index

- Code and data spaces are disjoint
  - compiled programs cannot corrupt their execution environment
  - Can not jump to arbitrary locations
  - Perform other undefined behaviour



WebAssembly Essentials: Control Flow

- Wasm VM is structured stack-based - control cannot flow to arbitrary locations.
- Unstructured and irreducible control flow using *goto* is impossible in Wasm.
- Flow moves to
  - Either to next instruction, if the current instruction is not branching
  - Or a “labeled” instruction

Labels
- BLOCK
- LOOP
- IF

Intra-function calls
- BR
- BR_IF
- BR_TABLE
- IF block IF/ELSE block

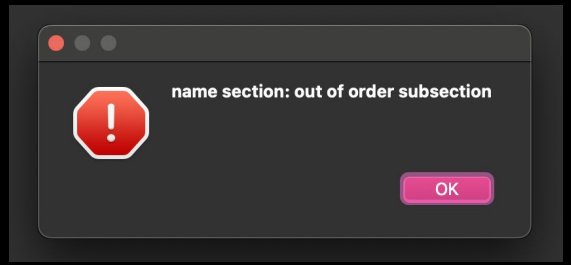
Inter-function calls
- CALL (static call invocation)
- CALL_INDIRECT (dynamic callsite invocation)

WebAssembly Tooling

WebAssembly Tooling

- WebAssembly Binary Toolkit
- Wasm-tools
- Ghidra
  - Using a Wasm plugin
  - Using to view Wasm files
- IDA Pro v9
  - It is hit-n-miss with Wasm
  - Using to view object files
- JEB Pro

TOOL	PURPOSE
wasm2wat	Translate binary Wasm to text format (.wat).
wasm-objdump	Print info about wasm binary. Similar to objdump
wasm-decompile	Decompile a wasm binary into readable C-like syntax.
wasm2c	Convert a WebAssembly binary file to a C source and header
wasm-interp	Decode and run a WebAssembly binary file using a stack-based interpreter
wasm-mutate	A WebAssembly test case mutator, producing a new, mutated wasm module.
Wasm-validate	Validate a file in the WebAssembly binary format



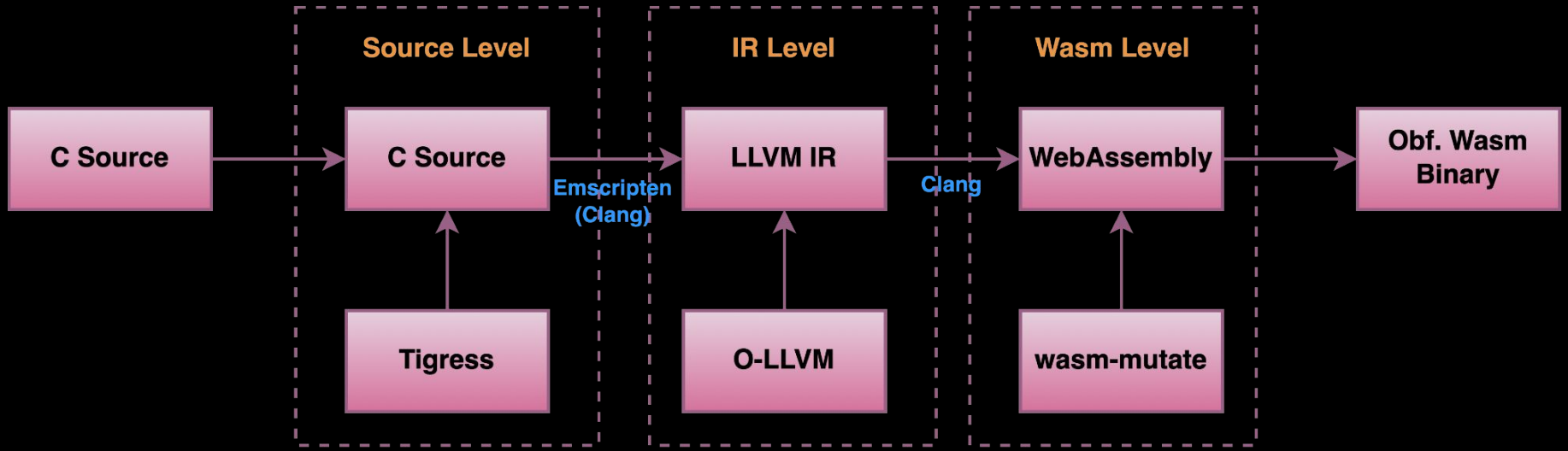
WebAssembly Obfuscation

Obfuscation

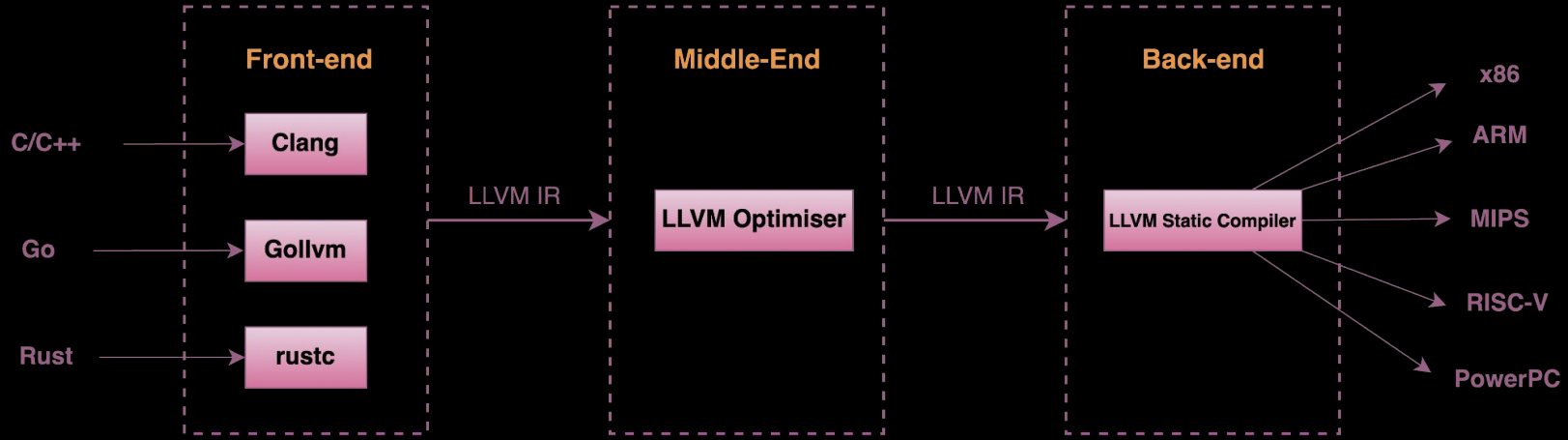
- Obfuscation: Process of making a program harder to understand while preserving the original program's behavior.

- To make code unreadable, for reasons ...
  - Malware author to avoid reversing
  - In apps, to prevent stealing/reversing of IP
  - Digital Rights Management (DRM)

WebAssembly Obfuscation Approaches



Obfuscation Using LLVM



- Open Source **Obfuscators**: O-LLVM, Hikari, Polaris
- Works on the middle-end
- Approach is source language agnostic

LLVM Based Obfuscators

O-LLVM

- Instruction Substitution
- Control Flow Flattening
- Basic block splitting
- Bogus control flow

Hikari

- Bogus control flow
- Control Flow Flattening
- Function call Obfuscate
- Function wrapper
- Basic block splitting
- String encryption
- Instruction Substitution
- Indirect Branching

Polaris

- Alias Access
- Flattening
- Indirect Branch
- Indirect Call
- String Encryption
- Bogus Control Flow
- Instruction Substitution
- Merge Function
- Linear MBA

Obfuscation: Steps

- Generate obfuscated Wasm binary

```
# 1. Generate obfuscated LLVM IR
obf_clang main.c -mllvm -sub -S -emit-llvm -o main_obf.ll
# 2. Generate Wasm using emscripten
emcc main_obf.ll -s Wasm=1 -o 1.html
```

- Newer versions of LLVM can process IR from older version, but not vice versa.

Obfuscation: O-LLVM Instruction Substitution

Original Input

```c
#include <stdio.h>

int calc(unsigned int n) {

    unsigned int mod = n % 4;

    unsigned int result = 0;

    if (mod == 0) result = (n | 0xBAAAD0BF) * (2 ^ n);

    else if (mod == 1) result = (n & 0xBAAAD0BF) * (3 + n);

    else if (mod == 2) result = (n ^ 0xBAAAD0BF) * (4 | n);

    else result = (n + 0xBAAAD0BF) * (5 & n);

    return result;
}

int main(int argc, char **argv) {
    printf("Hello from WebAsm! %d\n", calc(argc + 23));
    return 0;
}
```

Decompile: unnamed_function_3 - (ollvm_sub_loop_1.wasm)

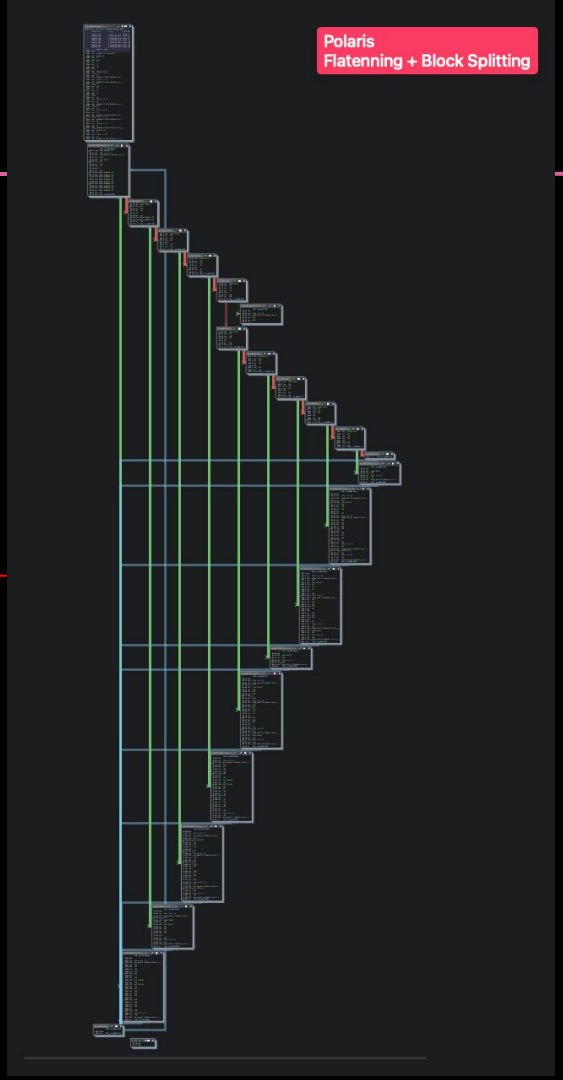
O-LLVM Inst. Sub

```c
int unnamed_function_3(uint param1)

{
  int iVar1;
  int local_10;

  iVar1 = (int)(param1 - 1) % 4;
  if (iVar1 == 0) {
    local_10 = (param1 & 0xbaaad0bf | param1 ^ 0xbaaad0bf) *
               (((param1 ^ 0xffffffff) & 0xbcec65b1 | param1 & 0x43139a4e) ^ 0xbcec65b3);
  }
  else if (iVar1 == 1) {
    local_10 = ((param1 ^ 0x45552f40) & param1) * (param1 + 3);
  }
  else if (iVar1 == 2) {
    local_10 = ((param1 ^ 0xffffffff) & 0xbaaad0bf | param1 & 0x45552f40) *
               (param1 & 4 | param1 ^ 4);
  }
  else {
    local_10 = (param1 + 0xbaaad0bf) * ((param1 ^ 0xffffffff | 0xfffffffa) ^ 0xffffffff);
  }
  return local_10;
}
```

Obfuscation: Flattening + Block Splitting

Original Input

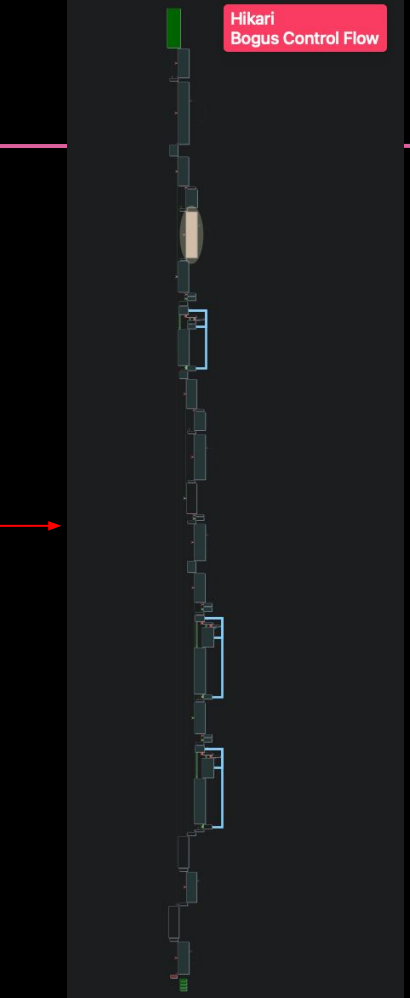
```c
#include <stdio.h>

int calc(unsigned int n) {

    unsigned int mod = n % 4;

    unsigned int result = 0;

    if (mod == 0) result = (n | 0xBAAAD0BF) * (2 ^ n);

    else if (mod == 1) result = (n & 0xBAAAD0BF) * (3 + n);

    else if (mod == 2) result = (n ^ 0xBAAAD0BF) * (4 | n);

    else result = (n + 0xBAAAD0BF) * (5 & n);

    return result;
}

int main(int argc, char **argv) {
    printf("Hello from WebAsm! %d\n", calc(argc + 23));
    return 0;
}
```



Obfuscation: Bogus Control Flow

Original Input

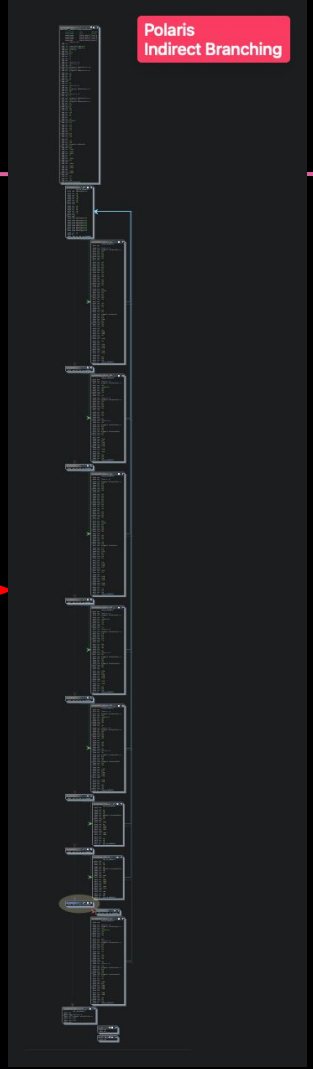
```c
#include <stdio.h>

int calc(unsigned int n) {

    unsigned int mod = n % 4;

    unsigned int result = 0;

    if (mod == 0) result = (n | 0xBAAAD0BF) * (2 ^ n);

    else if (mod == 1) result = (n & 0xBAAAD0BF) * (3 + n);

    else if (mod == 2) result = (n ^ 0xBAAAD0BF) * (4 | n);

    else result = (n + 0xBAAAD0BF) * (5 & n);

    return result;
}

int main(int argc, char **argv) {
    printf("Hello from WebAsm! %d\n", calc(argc + 23));
    return 0;
}
```

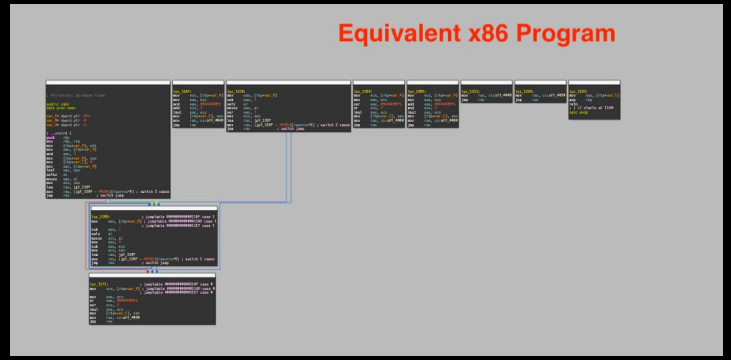


Obfuscation: Indirect Branching

Original Input

```c
#include <stdio.h>

int calc(unsigned int n) {

    unsigned int mod = n % 4;

    unsigned int result = 0;

    if (mod == 0) result = (n | 0xBAAAD0BF) * (2 ^ n);

    else if (mod == 1) result = (n & 0xBAAAD0BF) * (3 + n);

    else if (mod == 2) result = (n ^ 0xBAAAD0BF) * (4 | n);

    else result = (n + 0xBAAAD0BF) * (5 & n);

    return result;
}

int main(int argc, char **argv) {
    printf("Hello from WebAsm! %d\n", calc(argc + 23));
    return 0;
}
```





Obfuscation: Complexity Increases

- On applying obfuscation multiple times
  - Binary sizes can balloon, e.g to 12MB (*polaris with sub3_mba1_fla1*)
  - 2k+ LoC of decompiled code.
  - Tools start to break

Decompile: calc - (polaris_sub3_mba1_fla1.wasm)

```
Low-level Error: Flow exceeded maximum allowable instructions
```

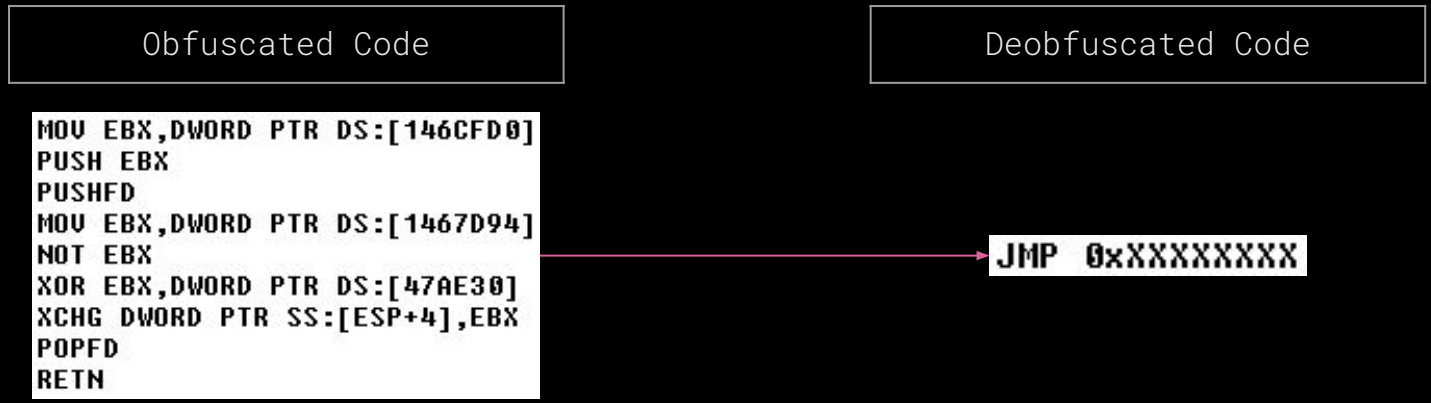
```
puVar1 = local_60;
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true )
{
while( true
)
{
while(
true
) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true ) {
while( true )
{
while( true
)
{
while (
local_20 == -0x7d3d55c7) {
  iVar10 = iRam00011d50 -
           ((uRam00011d54 ^ 0xffffffff) + 1);
  uVar2 = iVar10 + 0xceb83ae0;
  uVar2 = (uVar2 | 0x6852fabe) ^ 0xffffffff |
          (uVar2 ^ 0xffffffff | 0x6852fabe) +
          iVar10 + -0x3147c51f;
  uVar3 = ((uVar2 ^ 0xffffffff) & 0x7e4dd3c8 |
          uVar2 & 0x81b22c37) ^ 0x81b22c37 |
          0xdd78ec43;
  uVar2 = -uVar2 - 1;
```

RE//verse

WebAssembly Deobfuscation

Background: Classical (De)Obfuscation

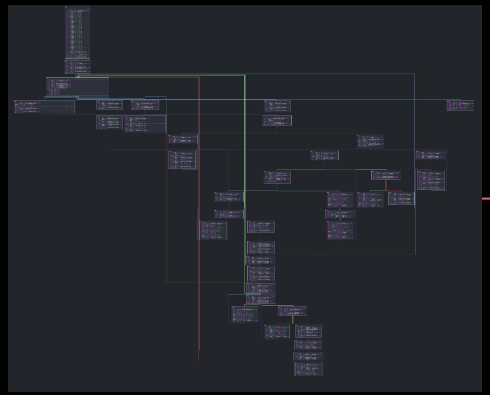
- *Classical Obfuscation*
  - Obfuscation patterns, constant unfolding, junk code insertion
- *Classical Deobfuscation*
  - Pattern matching



Background: Modern (De)Obfuscation

- *Modern obfuscation*
  - Source Code Level
  - Intermediate representation level
- *Modern deobfuscation*
  - Several Intermediate Languages at different abstract layers

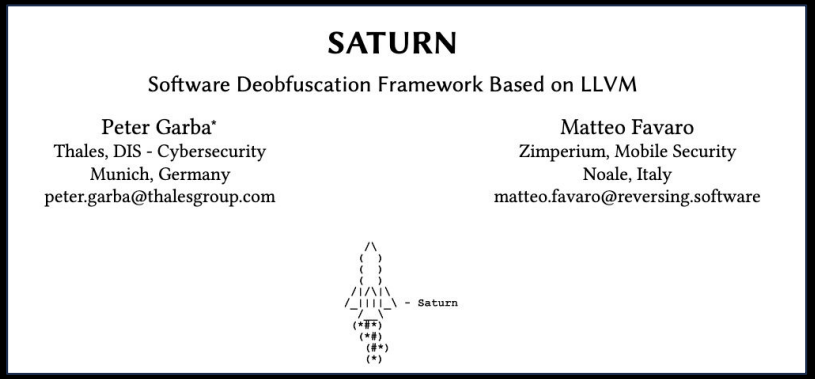
Obfuscated Code



Deobfuscated Code

```
int f(int a, int b) {
  return a + b + 15;
}
```

SATURN: Compiler Based Deobfuscation



SATURN

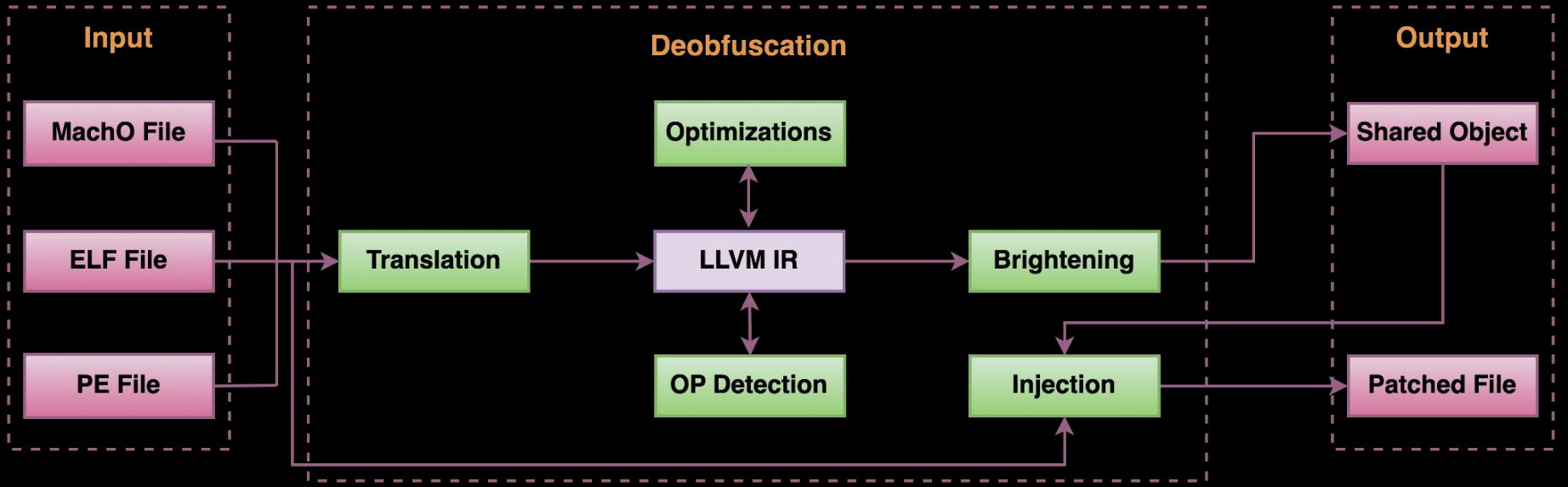
Software Deobfuscation Framework Based on LLVM

Peter Garba*
Thales, DIS - Cybersecurity
Munich, Germany
peter.garba@thalesgroup.com

Matteo Favaro
Zimperium, Mobile Security
Noale, Italy
matteo.favaro@reversing.software

- Generic approach for deobfuscation, based on LLVM compiler toolchain.
- Weakens certain obfuscations, and in best case completely removes them.

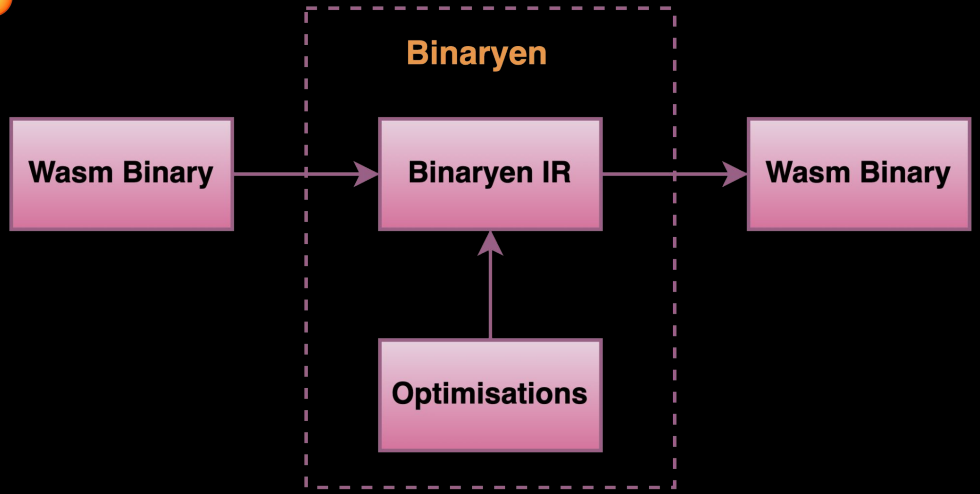
SATURN: Compiler Based Deobfuscation

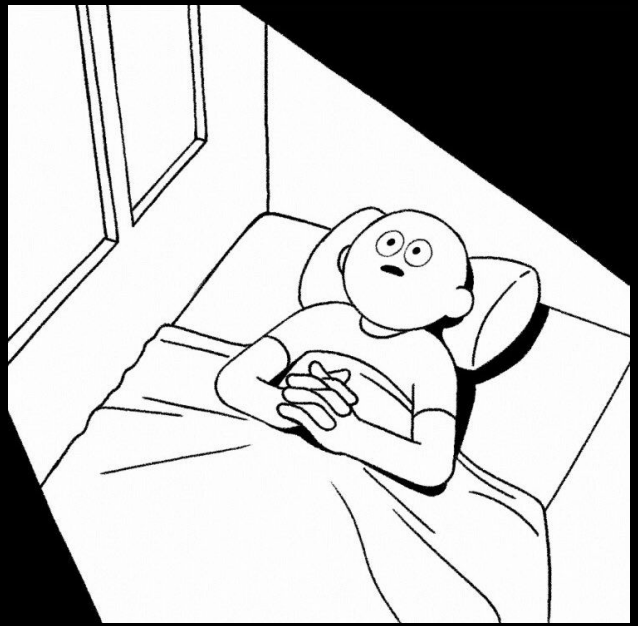


Binaryen

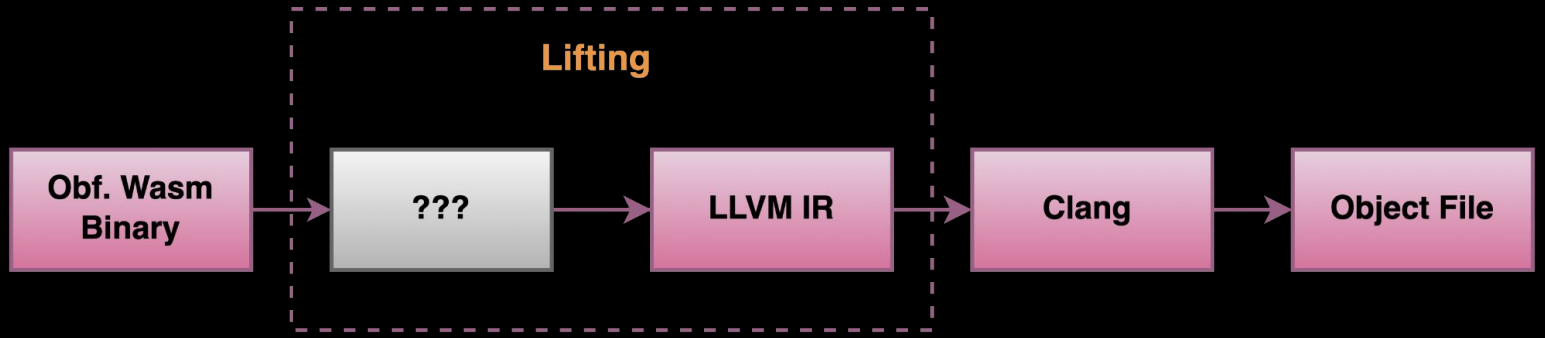
- Binaryen is a compiler and toolchain infrastructure library for Wasm.
- Binaryen's optimizer has many passes that can improve code size and speed.
- Input Wasm, Output Wasm
- **Didn’t work - no deobfuscation** 😞😡

Optimise using Binaryen
`wasm-opt -O4 input.wasm -o output.wasm`





Lifting to LLVM IR



Why LLVM?

- LLVM - a target-independent optimizer and code generator.
- LLVM has a language-independent intermediate representation (IR)
- **Advantages of using LLVM IR**
  - World Class Optimizations and Analysis Passes
  - Feature rich intermediate language
  - Accessible API
  - Normalization
  - Several backends available for recompilation
  - ***It's fast!***

Challenges of Lifting

- To leverage LLVM optimisation passes, requires lifting Wasm to LLVM IR.
- Challenges
  - Correctness
  - Captures side effects and expressiveness
  - Representation of the runtime environment
  - Stack machine to register machine transformation

Wasm Code Lifting to C: Lifting Principles

Wasm Code

```
        .text
        .file   "add.c"
        .functype       add (i32, i32) -> (i32)
        .section        .text.add,"",@
        .hidden add
        .globl  add
        .type   add,@function
add:                                    # @add
        .functype       add (i32, i32) -> (i32)
# %bb.0:
        local.get       1
        local.get       0
        i32.add
        i32.const       15
        i32.add

        end_function
```

Wasm Opcode Specification

Get Local

Mnemonic	Opcode	Immediates	Signature
`local.get`	0x20	`$id` : varuint32	`() : ($T[1])`

Integer Add

Mnemonic	Opcode	Signature
`i32.add`	0x6a	`(i32, i32) : (i32)`
`i64.add`	0x7c	`(i64, i64) : (i64)`

Mnemonic	Opcode	Immediates	Signature
`i32.const`	0x41	`$value` : varsint32	`() : (i32)`

Opcode Lifting Handler

```
case ExprType::LocalGet: {
  const Var& var = cast<LocalGetExpr>(&expr)->var;
  PushType(func_->GetLocalType(var));
  Write(StackVar(0), " = ", var, ";", Newline());
  break;
}
```

```
switch (expr.opcode) {
  case Opcode::I32Add:
  case Opcode::I64Add:
  case Opcode::F32Add:
  case Opcode::F64Add:
    WriteInfixBinaryExpr(expr.opcode, "+");
    break;
```

```
switch (const_.type()) {
  case Type::I32:
    Writef("%uu", static_cast<int32_t>(const_.u32()));
    break;
```

REVerse

Wasm Code Lifting to C: Good News - Already implemented!

Wasm Code

```
        .text
        .file   "add.c"
        .functype       add (i32, i32) -> (i32)
        .section        .text.add,"",@
        .hidden add
        .globl  add
        .type   add,@function
add:                                    # @add
        .functype       add (i32, i32) -> (i32)
# %bb.0:
        local.get       1
        local.get       0
        i32.add
        i32.const       15
        i32.add

        end_function
```

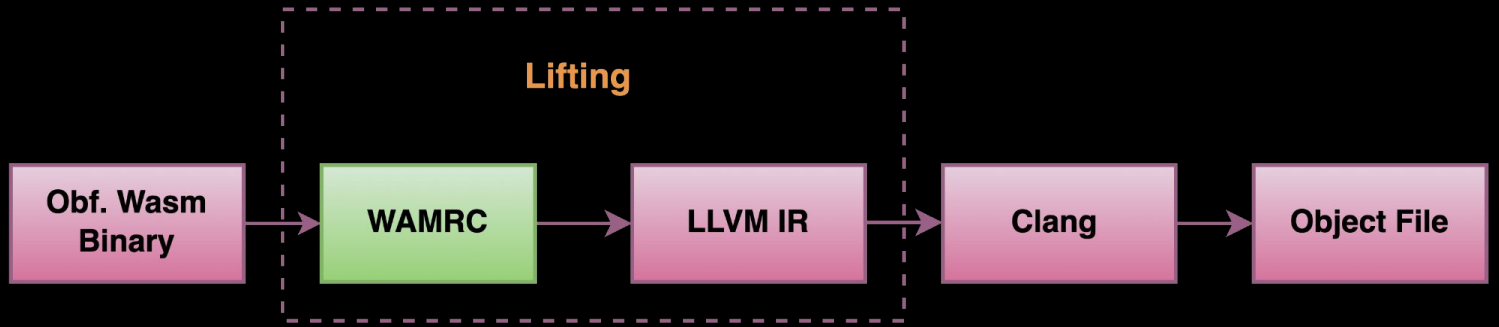
Lifted C Code

```
u32 add(u32 var_p0, u32 var_p1) {
  u32 var_i0, var_i1;
  var_i0 = var_p1;
  var_i1 = var_p0;
  var_i0 += var_i1;
  var_i1 = 15u;
  var_i0 += var_i1;
  return var_i0;
}
```

Wasm Code Lifting: Using WAMRC

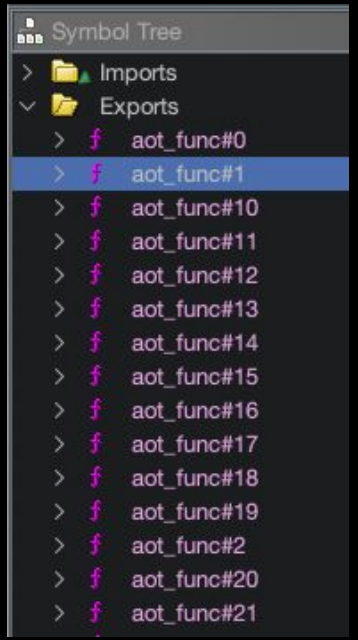
- WebAssembly Micro Runtime (WAMR)
  - Lightweight, standalone Wasm runtime
  - Small footprint, high performance
- WAMR Compiler (WAMRC)
  - The AOT compiler to compile Wasm file into AOT file

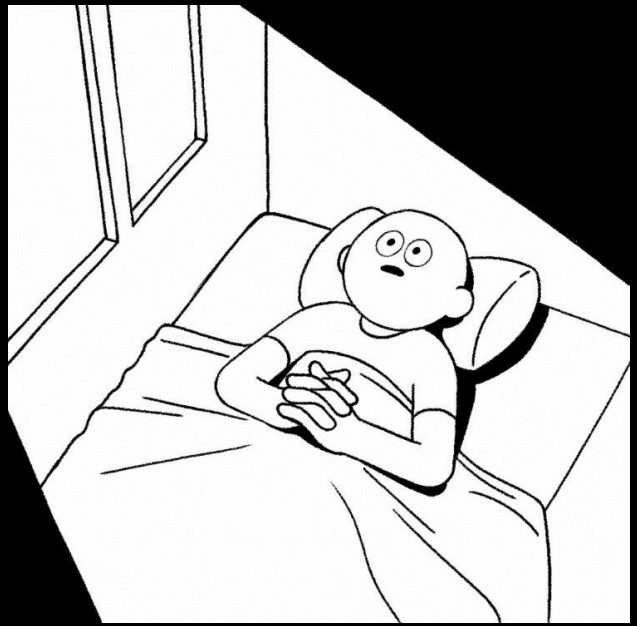
```
wamrc --format=llvmir-opt -o wamrc.ll obfuscated.wasm
```



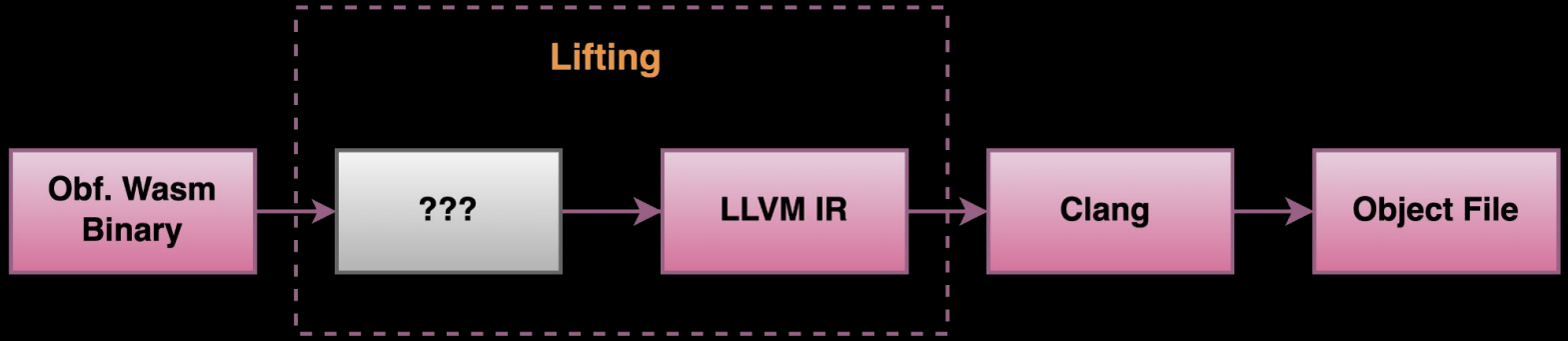
Wasm Code Lifting: Using WAMRC

- **Shortcomings**
  - Symbols information is lost
  - Generated LLVM IR does not contain various tables (global's table, function table)
  - LLVM **optimisations don't work**





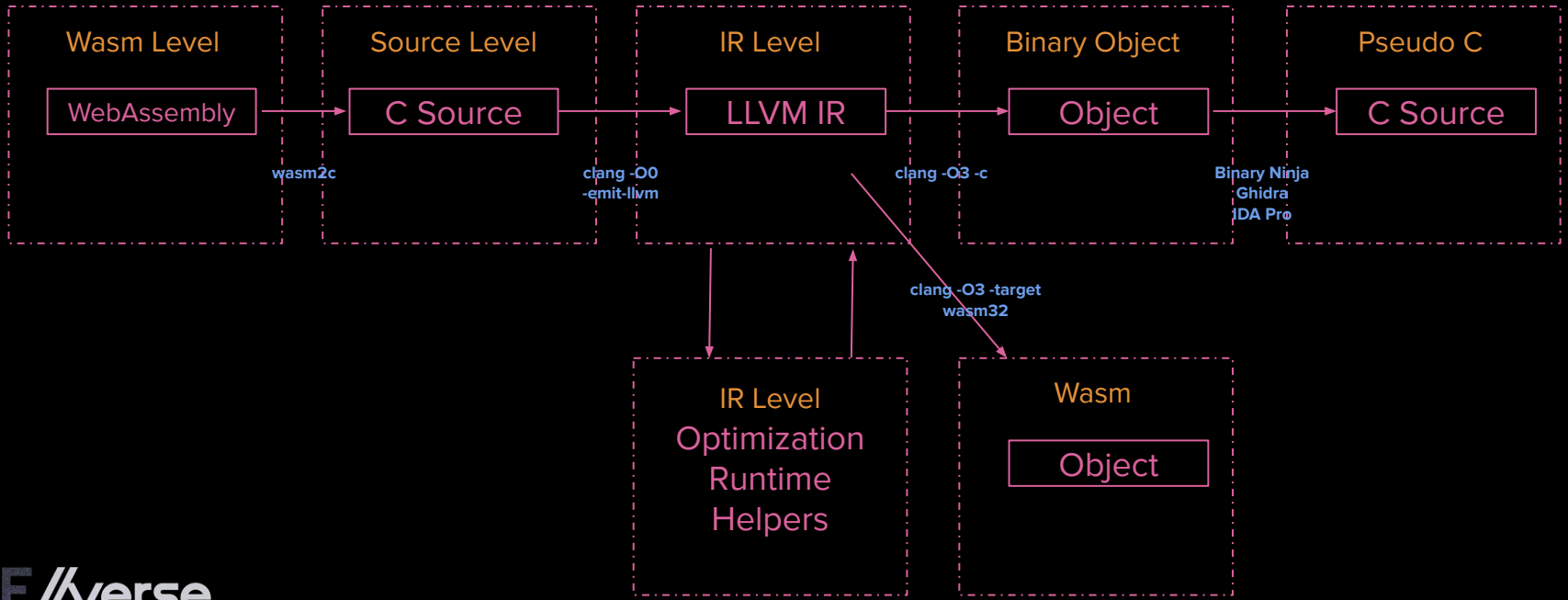
Wasm Lifter Problem Persists



Wasm Code Lifting: Code Lifters Comparison

Name	Lifting Language	Instance Parameter	Code Folding	Comments
Binaryen	Binaryen IR	Yes	No	Custom IR
WAMRC	LLVM IR	Yes	Partially	No tables (globals, functions...)
wasm2js	JS	No	Partially	Good candidate, but JS
wasm2c	C	Yes	No	Good candidate

Wasm Code Lifting: Lifting Idea!



Wasm Tool: wasm2c

- Great tool to lift Wasm to C
  - Well defined wasm runtime that helps during deobfuscation
    - Helpers to initialize wasm instance
    - Helpers to initialize the memory
    - Helpers to initialize and modify globals
  - Helpers for load/stores to memory
    - Load/Stores are access through helpers that can be overridden
  - Does not modify the original Control Flow Graph

Wasm Code Lifting: Motivating Example

```c
int add(int a, int b) {
  int arr[] = {1, 2, 3, 4, 5};
  int sum = 0;

  // Loop to calcuate a constant
  for (int i = 0; i < 5; i++) {
    sum += arr[i];
  }

  // MBA based Opaque Predicate
  if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
    sum += 1911;
  } else {
    sum += 2102;
  }

  return a + b + sum;
}
```

```c
int add(int a, int b) {
  return a + b + 1926;
}
```

Wasm Code Lifting: *wasm2c* (O3 Unobfuscated)

Wasm Level

WebAssembly

Pseudo C

C Source

```
        .text
        .file   "add.c"
        .functype       add (i32, i32) -> (i32)
        .section        .text.add,"",@
        .hidden add
        .globl  add
        .type   add,@function
add:                                    # @add
        .functype       add (i32, i32) -> (i32)
# %bb.0:
        local.get       1
        local.get       0
        i32.add
        i32.const       15
        i32.add

        end_function
```

wasm2c

```
u32 w2c_squanchy_add_0(w2c_squanchy* instance,
                       u32 var_p0, u32 var_p1) {
  u32 var_i0, var_i1;
  var_i0 = var_p1;
  var_i1 = var_p0;
  var_i0 += var_i1;
  var_i1 = 15u;
  var_i0 += var_i1;
  return var_i0;
}
```

clang -O3

```
define i32 @add(ptr %0, i32 %1, i32 %2) {
  %4 = add i32 %1, 15
  %5 = add i32 %4, %2
  ret i32 %5
}
```

clang -c + IDA Pro

```
int add(void *a1, int a2, int a3) {
  return a2 + a3 + 15;
}
```

REverse

Wasm Code Lifting: wasm2c

- *w2c_instance* is passed to all functions
  - Keeps track of execution state between functions
- *w2c_env_instance* can be freely used to keep track of important values
- Memory struct keeps the state of the current initialized memory
  - Will be initialized with table memory
- Function table is used for indirect function calls
- Globals will be dynamically generated

u32 w2c_add(w2c* instance, ...)

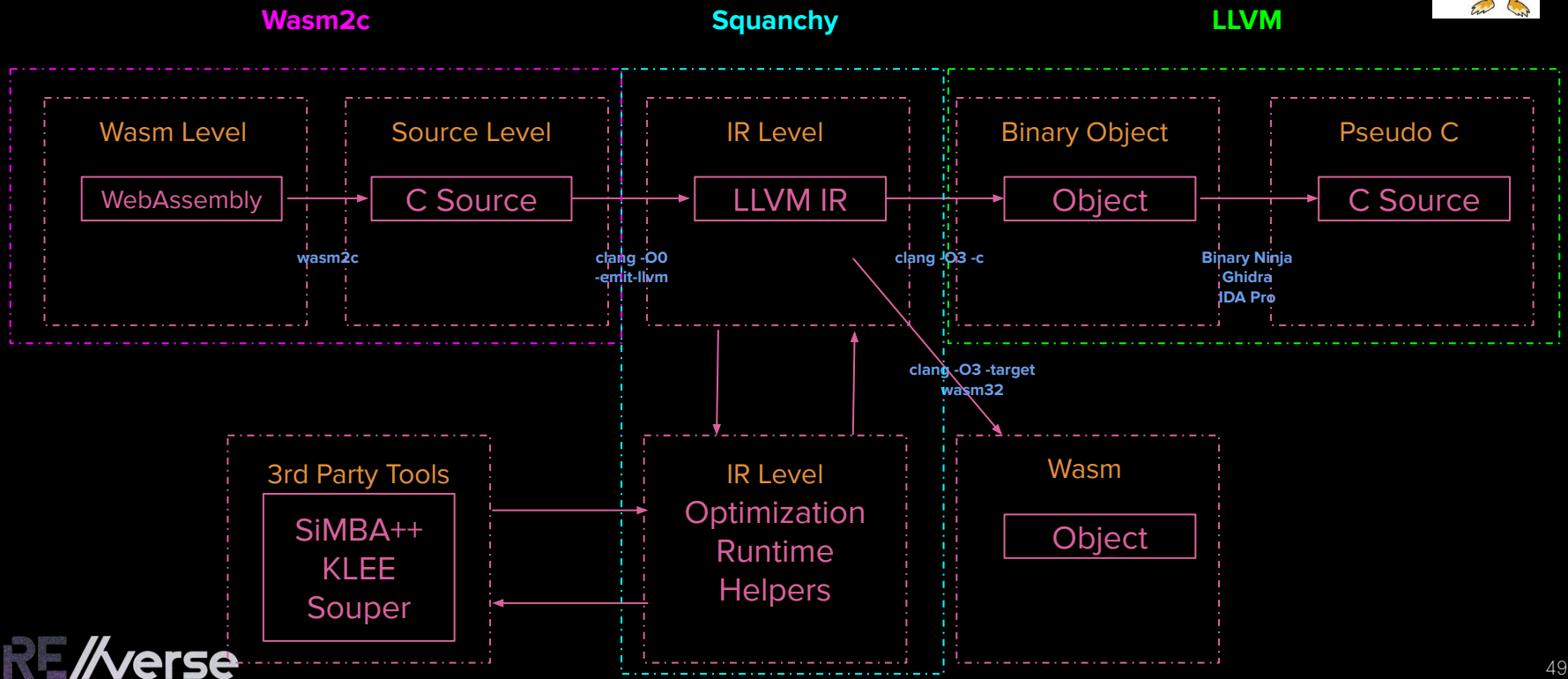
```
struct w2c_squanchy {
 struct w2c_env* w2c_env_instance;

 u32 *w2c_env_DYNAMICTOP_PTR;
 u32 *w2c_env_STACKTOP;
 u32 *w2c_env_STACK_MAX;

 // Memory
 struct wasm_rt_memory_t {
 uint8_t* data;
 uint64_t pages;
 uint64_t max_pages;
 uint64_t size;
 bool is64;
 } w2c_env_memory;

 u32 *w2c_env_memoryBase;

 // Function Ref Table
 wasm_rt_funcref_table_t *w2c_env_table;
 u32 *w2c_env_tableBase;

 // Globals
 u32 w2c_g5;
};
```

Wasm Code Lifting: wasm2c

- *w2c_instance* will be instantiated by helper functions
  - Initialized memory, globals and others

```
void wasm2c_instantiate(w2c* instance,
     struct w2c_env* w2c_env_instance) {
 assert(wasm_rt_is_initialized());
 init_instance_import(instance, w2c_env_instance);
 init_globals(instance);
 init_tables(instance);
 init_memories(instance);
 init_elem_instances(instance);
 init_data_instances(instance);
}
```

Wasm Code Lifting: Deobfuscation idea!



Wasm Code Lifting: Squanchy

- Tool to automate several deobfuscation steps
  - Models and injects the runtime
  - Injects runtime helpers into Module/Function
  - Inlines functions accordingly
  - Optimizes the function/module
    - Customized optimization pipeline to preserve Control Flow Graph
  - Removes *wasm2c* runtime
  - Extracts functions and dependencies into new clean module
    - https://github.com/pgarba/Squanchy

Wasm Code Lifting: Squanchy - Runtime Modeling

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Wasm function
W2C *, i32, i32

Lifted Wasm Code

Wasm2c Runtime Initializer

- Init Imports
- Init Globals
- Init Tables
- Init Elem/Data
- Init Memories
- ...

Wasm2c/Squanchy Helpers

- Allocate Memory
- Allocate FuncRef
- Allocate ExternRef
- Load/Stores
- TableBase
- ...

Wasm Code Lifting: Squanchy - Runtime Injection

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Wasm function
W2C *, i32, i32

Replace W2C Uses with local W2C

- Allocate W2C Instance
- Lifted Wasm Code

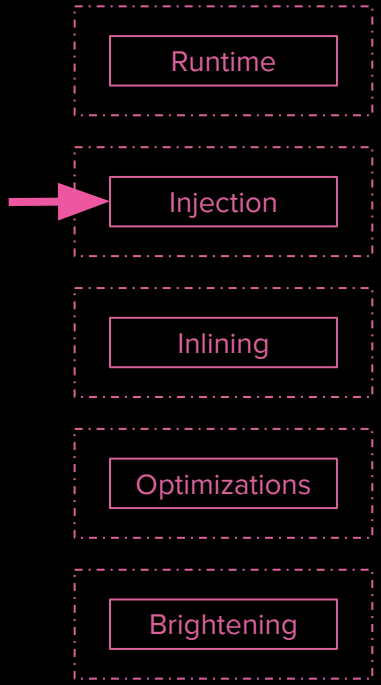
Wasm2c Runtime Initializer

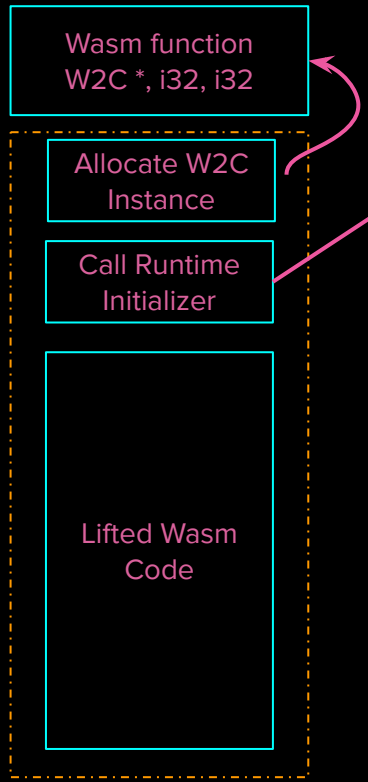
- Init Imports
- Init Globals
- Init Tables
- Init Elem/Data
- Init Memories
- ...

Wasm2c/Squanchy Helpers

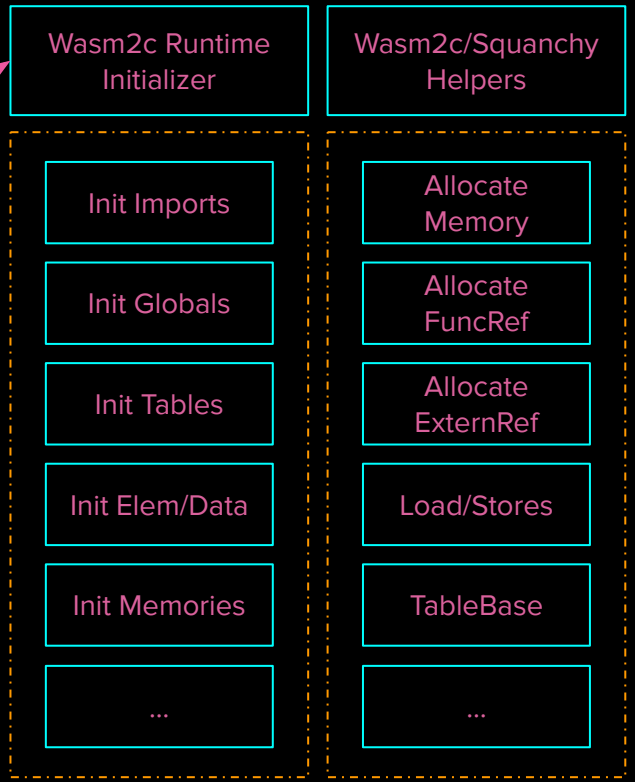
- Allocate Memory
- Allocate FuncRef
- Allocate ExternRef
- Load/Stores
- TableBase
- ...

Wasm Code Lifting: Squanchy - Runtime Injection





Create Call



Wasm Code Lifting: Squanchy - Runtime Injection

Runtime

Injection

Inlining

Optimizations

Brightening

Wasm function
W2C *, i32, i32

Allocate W2C
Instance

Call Runtime
Initializer

Lifted Wasm
Code

Mark functions
for inlining

Wasm2c Runtime
Initializer

Init Imports

Init Globals

Init Tables

Init Elem/Data

Init Memories

...

Wasm2c/Squanchy
Helpers

Allocate
Memory

Allocate
FuncRef

Allocate
ExternRef

Load/Stores

TableBase

...

Wasm Code Lifting: Squanchy - Runtime Injection

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Wasm function W2C *, i32, i32

- Allocate W2C Instance
- Init Imports
- Init Globals
- Init Tables
- Init Memories
- ...
- Lifted Wasm Code

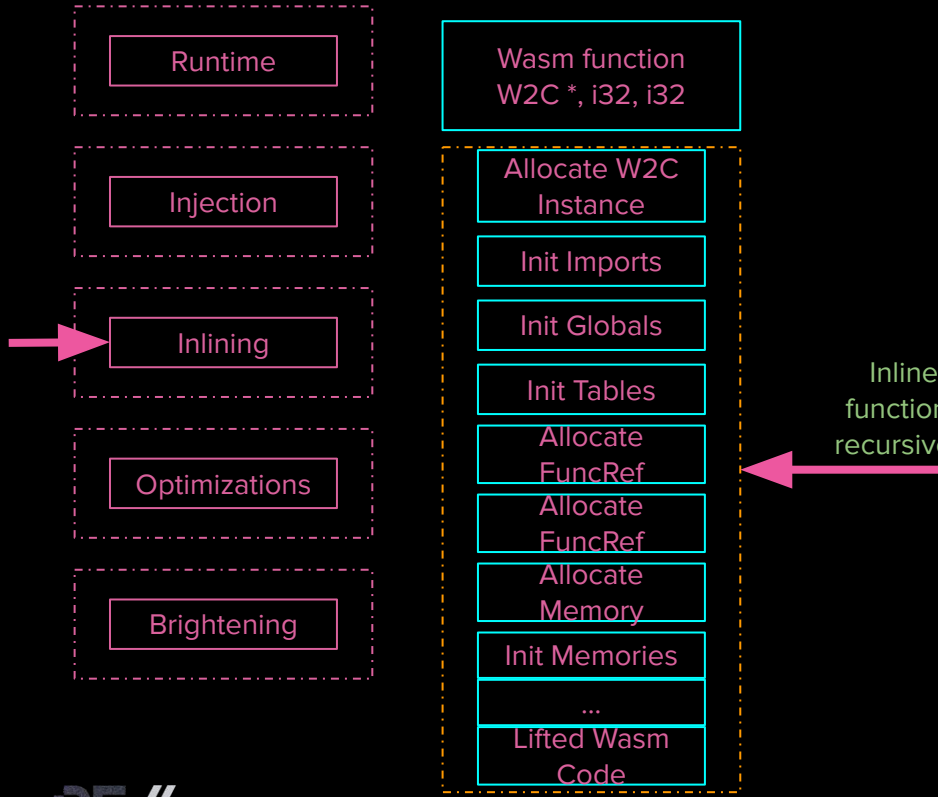
Inline functions recursively

Wasm2c Runtime Initializer

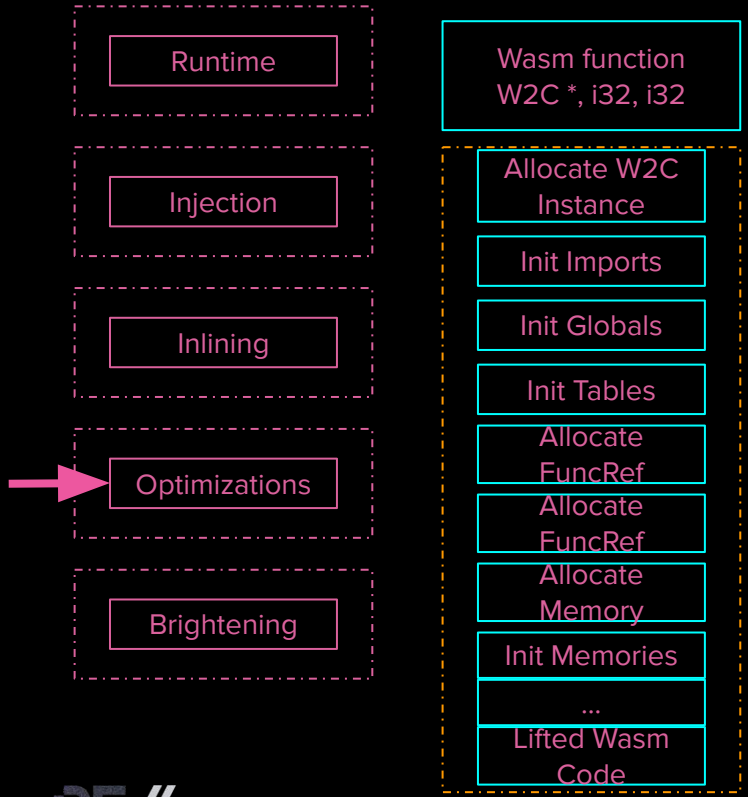
Wasm2c/Squanchy Helpers

- Allocate Memory
- Allocate FuncRef
- Allocate ExternRef
- Load/Stores
- TableBase
- ...

Wasm Code Lifting: Squanchy - Runtime Injection



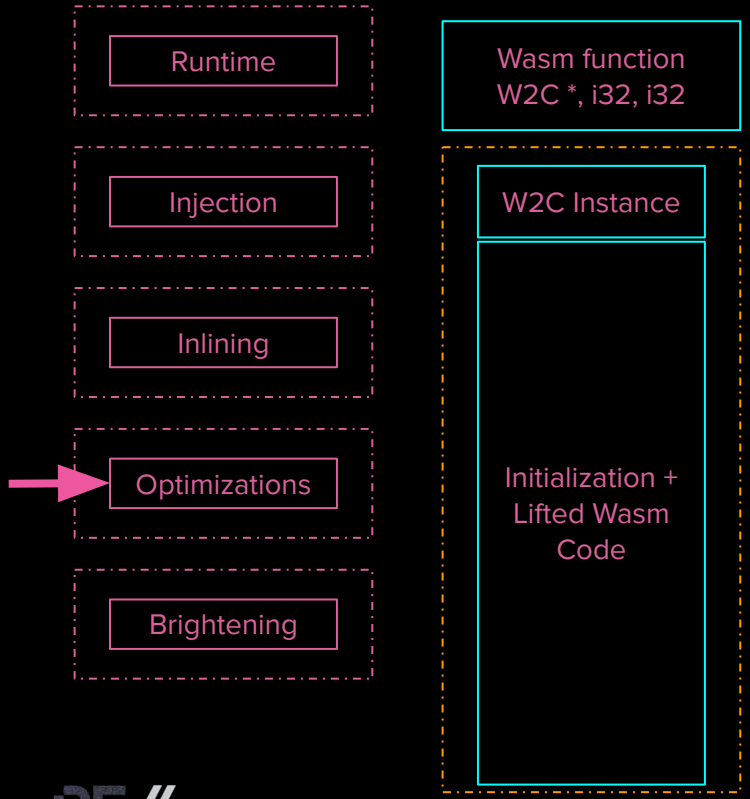
Wasm Code Lifting: Squanchy - Optimizations



Override LLVM Thresholds

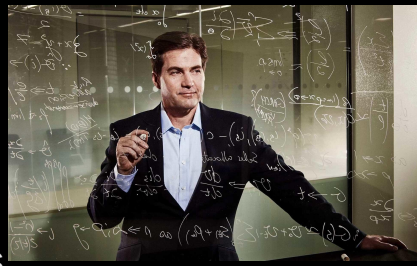
```
// DSE
-memdep-block-scan-limit=1000000
-dse-memoryssa-walklimit=1000000
-available-load-scan-limit=1000000
-dse-memoryssa-scanlimit=1000000

// Loop Unrolling
-unroll-threshold=1000000
-unroll-count=64
```

Wasm Code Lifting: **Squanchy - Optimizations**



Apply LLVM O3 pipeline and preserve Control Flow Graph

- Obfuscation pipelines are written by humans
  - Control Flow Protection
    - Control Flow Flattening
  - Code Protection
    - Instruction Substitutions
  - Harden Protections
    - Opaque Predicates
    - Mixed Boolean Arithmetics



RE//verse

Wasm Code Lifting: **Squanchy**

RE//verse

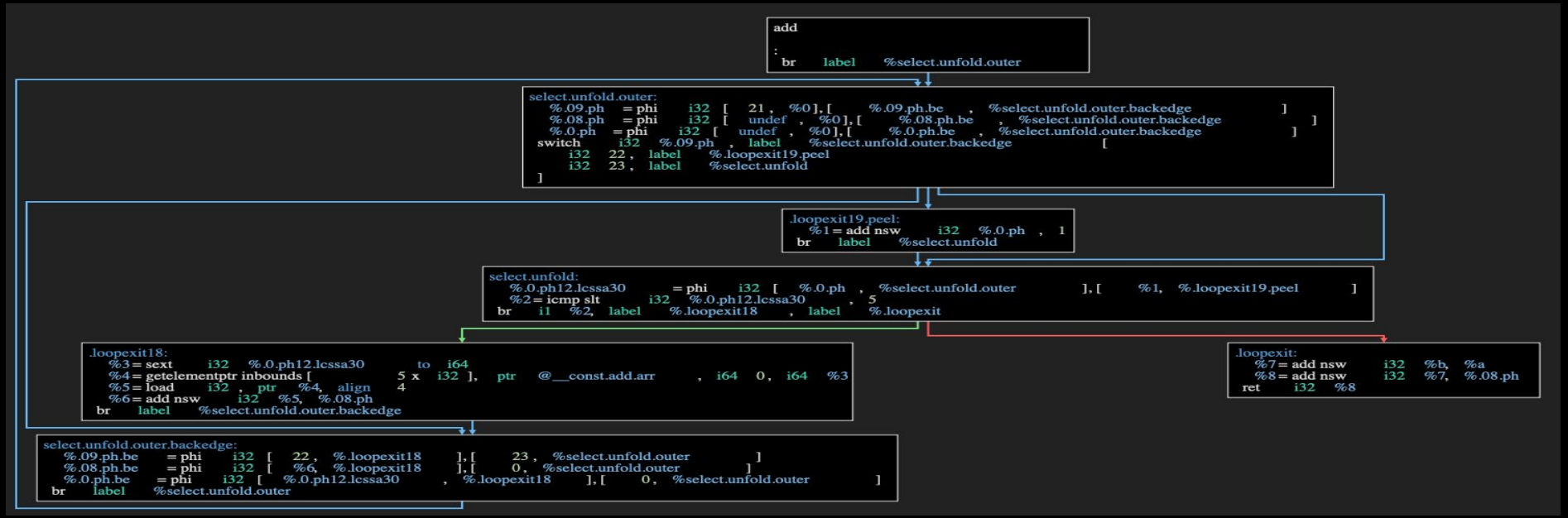
Wasm Code Lifting: **Squanchy - Optimizations**

Maintaining Original Control Flow Graph

REverse

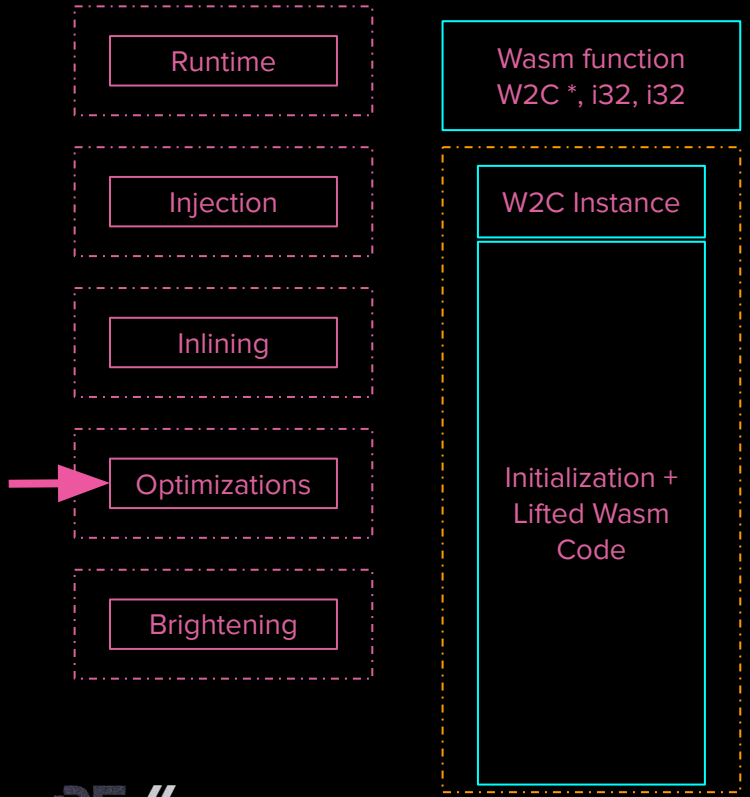
Wasm Code Lifting: **Squanchy - Optimizations**

Maintaining Original Control Flow Graph



RE//verse

Wasm Code Lifting: **Squanchy - Optimizations**



Weaken arithmetic expressions

- Apply tools like SiMBA++, Souper to find and weaken arithmetic expressions

```
// MBA based Opaque Predicate
if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
  sum += 1911;
} else {
  sum += 2102;
}
```

RE//verse

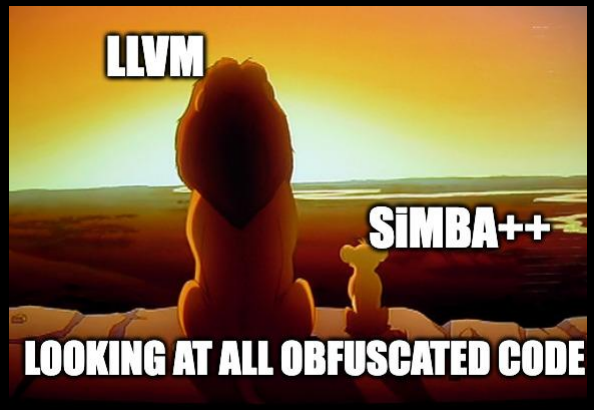
Beyond LLVM: Solving MBAs

- Mixed Boolean Arithmetic (MBA) expressions
  - Expressions mixing arithmetic operators (+,-,x) with boolean operators (¬, $\oplus$, ∧, ∨)
  - Difficult to analyze - no general rules for interaction b/w operators (no distributivity, no associativity etc.)
  - $(x \oplus y) + 2 \times (x \wedge y) = x + y$
  - With complex MBAs, SMT solvers may not able to solve them.

- Pattern based solving of MBAs can be overcome by chaining the MBAs.

Solving MBAs: Tooling

- Specialised tools for solving MBAs
  - SiMBA - For linear MBAs
  - GAMBA - Nonlinear MBA expression
  - SiMBA++ - For simplifying MBAs in LLVM IR
    - https://github.com/pgarba/SiMBA-
- SiMBA++
  - Detects candidate expressions in LLVM IR
  - Performs simplification using SiMBA or GAMBA
    - Supports calling external simplifiers
  - Replaces expressions with simplifications in LLVM IR



Solving MBAs: Tooling

```
[*] AST (Operators: 9):
3:   %12 = xor i32 %2, %1
2:   %13 = xor i32 %12, 15
1:   %14 = add i32 %13, 1911
3:   %15 = and i32 %2, %1
4:   %16 = or i32 %2, %1
3:   %17 = and i32 %16, 15
2:   %18 = or i32 %17, %15
1:   %19 = shl i32 %18, 1
0:   %20 = add i32 %14, %19
```

```
[!] Simplification: '1926+a+b with 2 operators!
```

```
[*] AST (Operators: 13):
7:   %168 = and i32 %167, -20481
6:   %170 = xor i32 %168, -559621617
5:   %178 = add i32 %177, %170
7:   %179 = sub i32 0, %178
6:   %180 = xor i32 %179, -1874919424
5:   %.neg29 = add i32 %180, 1
6:   %181 = shl i32 %178, 1
5:   %182 = add i32 %181, -2
4:   %183 = and i32 %182, -545128450
4:   %.neg28 = sub i32 %.neg29, %178
3:   %184 = add i32 %.neg28, %183
2:   %185 = lshr i32 %184, 16
1:   %196 = trunc nuw i32 %185 to i16
0:   %197 = xor i16 %196, -15696
```

```
[!] Simplification: '44400 with 0 operators!
```

Wasm Code Lifting: **Squanchy - Optimizations**

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Breaking Opaque Predicates

- Use SymLLVM to prove Basic Block edges

```
// MBA based Opaque Predicate
if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
  sum += 1911;
} else {
  sum += 2102;
}
```

Slice expression

```
[0]   %12 = xor i32 %1, -1
[1]   %13 = or i32 %12, %2
[2]   %14 = xor i32 %2, -1
[3]   %15 = and i32 %14, %1
[4]   %16 = xor i32 %2, %1
[5]   %17 = add i32 %16, 1
[6]   %.neg = add i32 %17, %15
[7]   %18 = add i32 %.neg, %13
[8]   %.not = icmp eq i32 %18, %16
[9]   %19 = select i1 %.not, i32 1926, i32 2117
```

SMT Formula

```
(declare-fun Result () (_ BitVec 32))
(declare-fun _a () (_ BitVec 32))
(declare-fun _b () (_ BitVec 32))
(assert
 (let ((?x12 (bvxor _b _a)))
(let ((?x8 (bvxor _a (_ bv4294967295 32))))
(let ((?x9 (bvor ?x8 _b)))
(let ((?x10 (bvxor _b (_ bv4294967295 32))))
(let ((?x11 (bvand ?x10 _a)))
(let ((?x14 (bvadd ?x12 (_ bv1 32))))
(let ((?x15 (bvadd ?x14 ?x11)))
(let ((?x16 (bvadd ?x15 ?x9)))
(let (($x17 (= ?x16 ?x12)))
(let ((?x20 (ite $x17 (_ bv1926 32) (_ bv2117 32))))
(= (bvsub ?x20 Result) (_ bv0 32))))))))))))
(check-sat)
```

RE//verse

Wasm Code Lifting: **Squanchy - Optimizations**

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Breaking Opaque Predicates

- Use SymLLVM to prove Basic Block edges

```
// MBA based Opaque Predicate
if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
  sum += 1911;
} else {
  sum += 2102;
}
```

Slice expression

```
[0]   %12 = xor i32 %1, -1
[1]   %13 = or i32 %12, %2
[2]   %14 = xor i32 %2, -1
[3]   %15 = and i32 %14, %1
[4]   %16 = xor i32 %2, %1
[5]   %17 = add i32 %16, 1
[6]   %.neg = add i32 %17, %15
[7]   %18 = add i32 %.neg, %13
[8]   %.not = icmp eq i32 %18, %16
[9]   %19 = select i1 %.not, i32 1926, i32 2117
```

SMT Formula

```
(declare-fun Result () (_ BitVec 32))
(declare-fun _a () (_ BitVec 32))
(declare-fun _b () (_ BitVec 32))
(assert
 (let ((?x12 (bvxor _b _a)))
(let ((?x8 (bvxor _a (_ bv4294967295 32))))
(let ((?x9 (bvor ?x8 _b)))
(let ((?x10 (bvxor _b (_ bv4294967295 32))))
(let ((?x11 (bvand ?x10 _a)))
(let ((?x14 (bvadd ?x12 (_ bv1 32))))
(let ((?x15 (bvadd ?x14 ?x11)))
(let ((?x16 (bvadd ?x15 ?x9)))
(let (($x17 (= ?x16 ?x12)))
(let ((?x20 (ite $x17 (_ bv1926 32) (_ bv2117 32))))
(= (bvsub ?x20 Result) (_ bv0 32))))))))))))
(check-sat)
```

```
((Z3Expr - V) == 0) -> SAT 1926
```

RE//verse

Wasm Code Lifting: **Squanchy - Optimizations**

- Runtime
- Injection
- Inlining
- Optimizations
- Brightening

Breaking Opaque Predicates

- Use SymLLVM to prove Basic Block edges

```
// MBA based Opaque Predicate
if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
  sum += 1911;
} else {
  sum += 2102;
}
```

Slice expression

```
[0]   %12 = xor i32 %1, -1
[1]   %13 = or i32 %12, %2
[2]   %14 = xor i32 %2, -1
[3]   %15 = and i32 %14, %1
[4]   %16 = xor i32 %2, %1
[5]   %17 = add i32 %16, 1
[6]   %.neg = add i32 %17, %15
[7]   %18 = add i32 %.neg, %13
[8]   %.not = icmp eq i32 %18, %16
[9]   %19 = select i1 %.not, i32 1926, i32 2117
```

SMT Formula

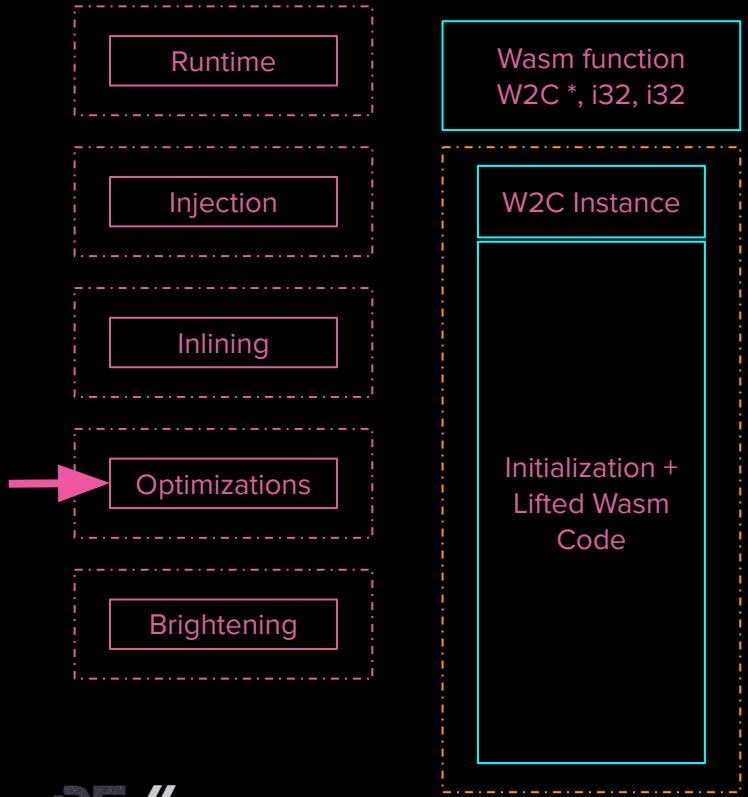
```
(declare-fun Result () (_ BitVec 32))
(declare-fun _a () (_ BitVec 32))
(declare-fun _b () (_ BitVec 32))
(assert
 (let ((?x12 (bvxor _b _a)))
(let ((?x8 (bvxor _a (_ bv4294967295 32))))
(let ((?x9 (bvor ?x8 _b)))
(let ((?x10 (bvxor _b (_ bv4294967295 32))))
(let ((?x11 (bvand ?x10 _a)))
(let ((?x14 (bvadd ?x12 (_ bv1 32))))
(let ((?x15 (bvadd ?x14 ?x11)))
(let ((?x16 (bvadd ?x15 ?x9)))
(let (($x17 (= ?x16 ?x12)))
(let ((?x20 (ite $x17 (_ bv1926 32) (_ bv2117 32))))
(= (bvsub ?x20 Result) (_ bv0 32))))))))))))))
(check-sat)
```

```
((Z3Expr - V) == 0) -> SAT 1926
```

```
((Z3Expr - V) == 0) != 1926 -> UNSAT
```

RE//verse

Wasm Code Lifting: **Squanchy - Optimizations**



Breaking Opaque Predicates

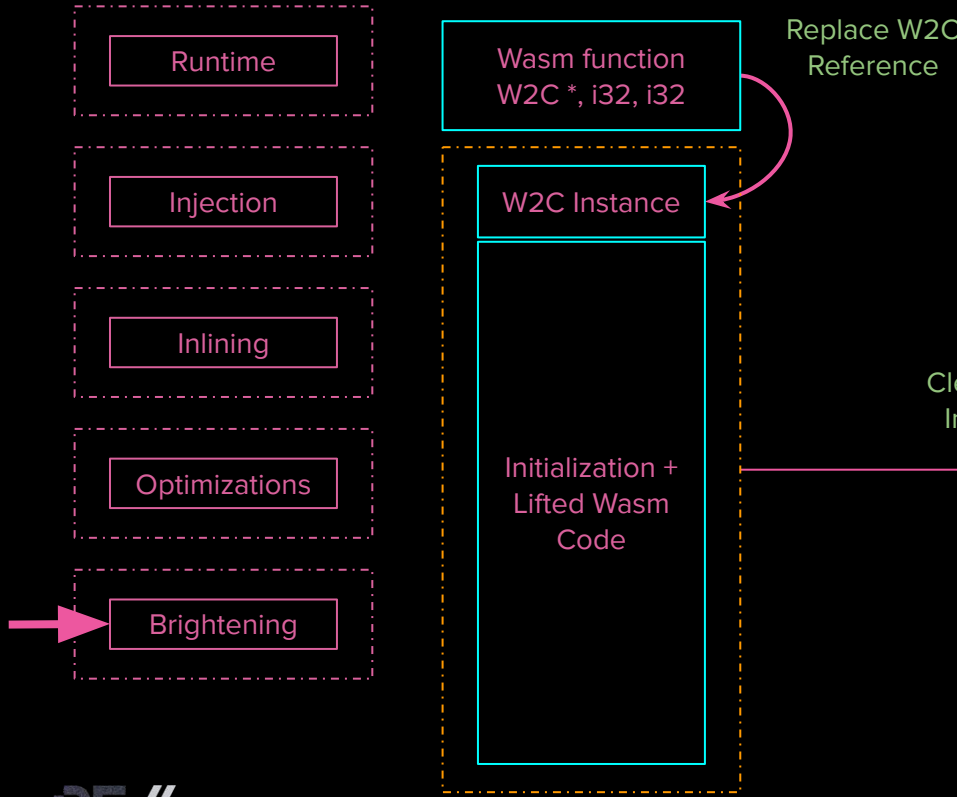
- Use SymLLVM to prove Basic Block edges

```
// MBA based Opaque Predicate
if (((~a|b)+(a&~b)-~(a^b)) - (a^b) == 0) {
  sum += 1911;
} else {
  sum += 2102;
}
```

```
sum += 1911;
```

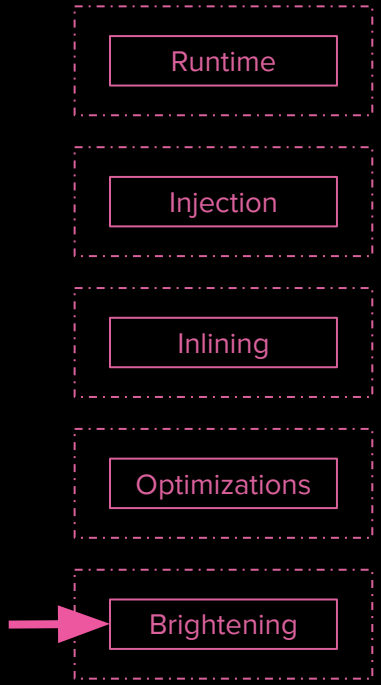
RE//verse

Wasm Code Lifting: **Squanchy - Brightening**



RE//verse

Wasm Code Lifting: **Squanchy**



- Brightening of the LLVM IR Module
  - Extract deobfuscated functions and data segments into new clean module without the *wasm2c* code
    - Based on *LLVM-Extract*
    - Module can be further deobfuscated with tools like *SiMBA++* or *Souper*
  - Apply LLVM Module optimization with O3
    - Some passes are only available as module passes
    - Gives the best results

Wasm Code Lifting: **Squanchy - Recompilation**

Runtime

Injection

Inlining

Optimizations

Brightening

Wasm function
W2C *, i32, i32

Deobfuscated
Wasm Code

LLVM IR

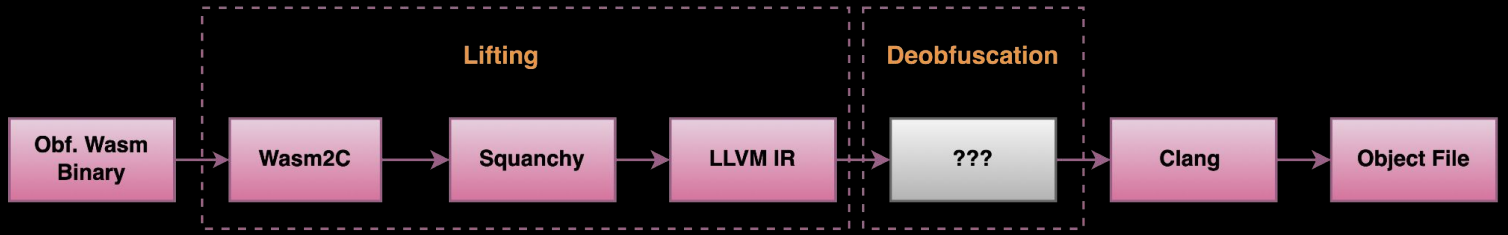
```
define i32 @add(ptr %0, i32 %1, i32 %2) {
  %3 = add i32 %1, 1926
  %4 = add i32 %3, %2
  ret i32 %4
}
```

ARM64

```
w2c_squanchy_add_0:
    add     w8, w1, #1926
    add     w0, w8, w2
    ret
```

REverse

Deobfuscation



Reminder: Original Input

Original Input

```c
#include <stdio.h>

int calc(unsigned int n) {

    unsigned int mod = n % 4;

    unsigned int result = 0;

    if (mod == 0) result = (n | 0xBAAAD0BF) * (2 ^ n);

    else if (mod == 1) result = (n & 0xBAAAD0BF) * (3 + n);

    else if (mod == 2) result = (n ^ 0xBAAAD0BF) * (4 | n);

    else result = (n + 0xBAAAD0BF) * (5 & n);

    return result;
}

int main(int argc, char **argv) {
    printf("Hello from WebAsm! %d\n", calc(argc + 23));
    return 0;
}
```

1
2
3
4

Deobfuscation: LLVM Optimisations

OLLVM
Instruction Subsitution

```
if (iVar1 == 0) {
  local_10 = (param1 & 0xbaaad0bf | param1 ^ 0xbaaad0bf) *
             (((param1 ^ 0xffffffff) & 0xbcec65b1 | param1 & 0x43139a4e) ^ 0xbcec65b3);   1
}
else if (iVar1 == 1) {
  local_10 = ((param1 ^ 0x45552f40) & param1) * (param1 + 3);                             2
}
else if (iVar1 == 2) {
  local_10 = ((param1 ^ 0xffffffff) & 0xbaaad0bf | param1 & 0x45552f40) *
             (param1 & 4 | param1 ^ 4);                                                   3
}
else {
  local_10 = (param1 + 0xbaaad0bf) * ((param1 ^ 0xffffffff | 0xfffffffa) ^ 0xffffffff);  4
}
```

```
if (iVar1 == 0) {
  local_24 = (param_2 | 0xbaaad0bf) * (param_2 ^ 2);                  1
}
else if (iVar1 == 1) {
  local_24 = (param_2 & 0xbaaad0bf) * (param_2 + 3);                  2
}
else if (iVar1 == 2) {
  local_24 = (param_2 ^ 0xbaaad0bf) * (param_2 | 4);                  3
}
else {
  local_24 = (param_2 + 0xbaaad0bf) * (param_2 & 5);                  4
}
```

Deobfuscation: LLVM Optimisations

O-LLVM
Instruction Subsitution (3 loops)

```
else if (iVar1 == 2) {
  uVar10 = (param1 ^ 0xffffffff) & param1;
  uVar11 = uVar10 ^ 0xffffffff;
  uVar10 = (param1 & 0x28b159a6 | (param1 ^ 0xffffffff) & 0xd74ea659) ^
           Unsigned Integer (compiler-specific size) | uVar10 & 0xd74ea659) | (param1 | uVar11) ^ 0xffffffff;
  uVar1  Length: 4                                   & 0xad02e611 | uVar10 & 0x52fd19ee;
  uVar10 = (uVar10 ^ 0x52fd19ee) & 0x2d4bd55a | (uVar10 ^ 0xad02e611) & 0xd2b42aa5;
  uVar10 = (uVar10 ^ 0x2d4bd55a) & uVar10;
  uVar11 = (param1 & 0x43df8f | (param1 ^ 0xffffffff) & 0xffbc2070) ^ 0xd2f7f52a |
           (param1 | 0x2d4bd55a) ^ 0xffffffff;
  uVar12 = uVar10 ^ 0xffffffff | uVar11;
  uVar10 = (uVar11 ^ 0xffffffff) & (uVar10 ^ 0xffffffff) | uVar10 & uVar11;
  uVar11 = uVar10 ^ 0xffffffff;
  uVar10 = (uVar12 & 0xac3e94d1 | (uVar12 ^ 0xffffffff) & 0x53c16b2e) ^
           (uVar11 & 0xac3e94d1 | uVar10 & 0x53c16b2e) | (uVar12 | uVar11) ^ 0xffffffff;
  uVar11 = uVar10 ^ 0x35ec8eeb;
  uVar10 = uVar10 ^ 0xffffffff | 0x35ec8eeb;
  uVar12 = uVar11 & 0x35ec8eeb ^ 0xffffffff;
  uVar10 = (uVar12 & 0x2d5bbaff | uVar11 & 0x10a40400) ^
           (uVar10 & 0x2d5bbaff | (uVar10 ^ 0xffffffff) & 0xd2a44500) |
           (uVar12 | uVar10) ^ 0xffffffff;
  uVar11 = (param1 ^ 0xffffffff) & 0x733e697e | param1 & 0x8cc19681;
  uVar11 = ((uVar11 ^ 0x8cc19681) & 0xfffffffb | (uVar11 ^ 0x733e697e) & 4) ^ 0xffffffff |
           0xfffffffb;
  uVar12 = (param1 ^ 0x55a04b31) & (param1 ^ 0xffffffff);
  uVar4 = (param1 ^ 0xffffffff | 0x55a04b31) ^ 0xffffffff;
  uVar12 = uVar12 & uVar4 | uVar12 ^ uVar4;
  uVar12 = (uVar12 ^ 0xffffffff) & 0xf7e551b4 | uVar12 & 0x81aae4b;
  uVar4 = (uVar12 ^ 0x5dbae57e) & 0xda9fbf90 | (uVar12 ^ 0xa2451a81) & 0x2560406f;
  uVar5 = (uVar11 ^ 0x8149b87a) & uVar11;
  uVar6 = (uVar11 ^ 0x8149b87a) & (uVar11 ^ 0xffffffff);
  uVar7 = (uVar4 ^ 0xa429f815) & (uVar4 ^ 0x2560406f);
  uVar12 = (uVar12 ^ 0xdcf35d04) & (uVar12 ^ 0xa2451a81);
  uVar8 = uVar5 ^ 0xffffffff;
  uVar9 = uVar6 ^ 0xffffffff;
  uVar5 = (uVar8 & 0xa7224c94 | uVar5 & 0x58ddb36b) ^ (uVar9 & 0xa7224c94 | uVar6 & 0x58ddb36b) |
          (uVar8 | uVar9) ^ 0xffffffff;
  uVar12 = uVar7 & uVar12 | uVar7 ^ uVar12;
  uVar12 = uVar12 & (uVar5 ^ 0xffffffff) | uVar5 & (uVar12 ^ 0xffffffff);
  uVar11 = ((uVar11 ^ 0xffffffff) & 0x3c7da929 | uVar11 & 0xc38256d6) ^
           ((uVar4 ^ 0xda9fbf90) & 0x3c7da929 | (uVar4 ^ 0x2560406f) & 0xc38256d6) |
           (uVar11 ^ 0xffffffff | uVar4 ^ 0xda9fbf90) ^ 0xffffffff;
  uVar11 = (uVar11 ^ 0xffffffff) & 0x51a3afbb | uVar11 & 0xae5c5044;
  uVar4 = uVar11 ^ 0x51a3afbb;
  uVar5 = uVar12 ^ 0xffffffff;
  local_10 = (((uVar10 ^ 0xffffffff) & 0xebfaad80 | uVar10 & 0x1405527f) ^ 0xb608d971) *
             ((uVar5 & 0xf368b83d | uVar12 & 0xc9747c2) ^
              (uVar4 & 0xf368b83d | (uVar11 ^ 0xae5c5044) & 0xc9747c2) |
             (uVar5 | uVar4) ^ 0xffffffff);
}
```

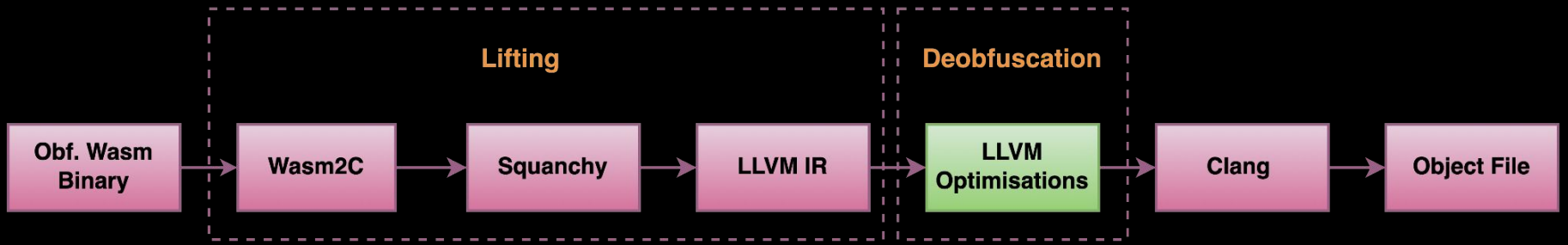
2

LLVM Optimised

```
if (iVar5 == 0) {
  uVar2 = (((param_2 | 0xbaaad0bf) & 0xc4fa1585 | param_2 & 0x1052a40) ^ param_2 ^ 0x80aa1085) &
          (param_2 | 0xbaaad0bf | param_2 ^ 0x45552f40);
  uVar1 = uVar2 & (param_2 ^ 0xbaaad0bf);
  uVar2 = uVar2 ^ param_2;
  uVar3 = ((param_2 & 0xfffffffd ^ 0xffffffff) & 0x34f5a7e6 | param_2 & 0xcb0a5819) ^
          (param_2 & 2 | 0xdb2decf5);
  uVar4 = ((param_2 & 0xfffffffd ^ 0xffffffff) & 0x48c117 | param_2 & 0xffb73ee8) ^
          (param_2 & 2 | 0x48c115) | param_2 ^ 0xfffffffd;
  local_24 = (uVar2 ^ uVar1 ^ 0xbaaad0bf | (uVar2 ^ 0xbaaad0bf) & uVar1) *
             (uVar3 ^ uVar4 ^ 0x1027b4ee | (uVar3 ^ 0x1027b4ee | uVar4) ^ 0xffffffff);
}
else if (iVar5 == 1) {
  local_24 = ((((param_2 | 0xbaaad0bf) & 0x3c966fda | param_2 & 0x41410000) ^ param_2 ^ 0x3882409a
              ) & (param_2 | 0xbaaad0bf | param_2 ^ 0x45552f40)) * (param_2 + 3);
}
else if (iVar5 == 2) {
  uVar2 = (param_2 ^ 0xfffffffb | param_2 ^ 4) & (param_2 ^ 0x7eb64781);
  local_24 = (uVar2 & (param_2 & 4 | 0x8149b87a) |
              (param_2 & 4 ^ 0x7eb64785) & (uVar2 ^ 0xffffffff)) * (param_2 ^ 0xbaaad0bf);
}
else {
  local_24 = (param_2 + 0xbaaad0bf) * (param_2 & 5);
}
```

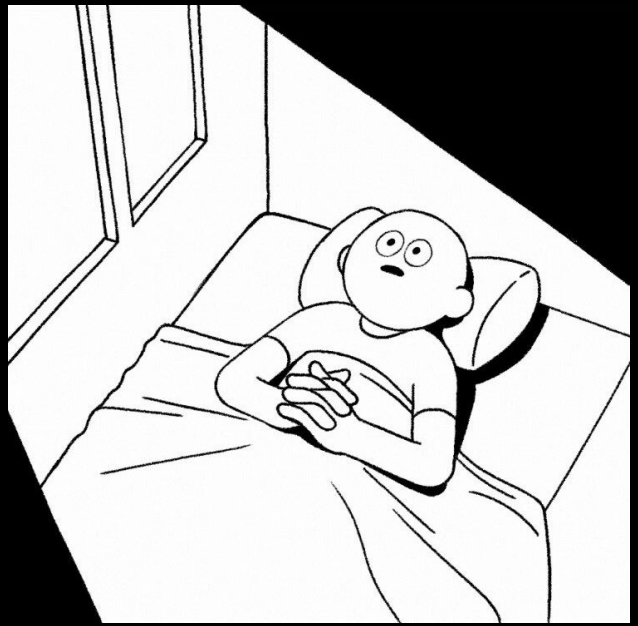
1
2
3
4
Original Expression Recovered

Pipeline


Lifting
Deobfuscation
Obf. Wasm Binary
Wasm2C
Squanchy
LLVM IR
LLVM Optimisations
Clang
Object File

Deobfuscation: LLVM Optimisation Shortcomings

- May only weaken some obfuscations
- Some techniques which LLVM cannot outright break
  - Control flow flattening *
  - Bogus control flow
  - Solving complex MBAs
    - Multiple iterations of substitution
  - . . .



The Lion King Show: SiMBA, GAMBA and MBAs

LLVM Optimised

```
20 if (iVar5 == 0) {                                                                    1
21   uVar2 = (((param_2 | 0xbaaad0bf) & 0xc4fa1585 | param_2 & 0x1052a40) ^ param_2 ^ 0x80aa1085) &
22           (param_2 | 0xbaaad0bf | param_2 ^ 0x45552f40);
23   uVar1 = uVar2 & (param_2 ^ 0xbaaad0bf);
24   uVar2 = uVar2 ^ param_2;
25   uVar3 = ((param_2 & 0xffffffd ^ 0xffffffff) & 0x34f5a7e6 | param_2 & 0xcb0a5819) ^
26           (param_2 & 2 | 0xdb2decf5);
27   uVar4 = ((param_2 & 0xffffffd ^ 0xffffffff) & 0x48c117 | param_2 & 0xffb73ee8) ^
28           (param_2 & 2 | 0x48c115) | param_2 ^ 0xffffffd;
29   local_24 = (uVar2 ^ uVar1 ^ 0xbaaad0bf | (uVar2 ^ 0xbaaad0bf) & uVar1) *
30              (uVar3 ^ uVar4 ^ 0x1027b4ee | (uVar3 ^ 0x1027b4ee | uVar4) ^ 0xffffffff);
31 }
32 else if (iVar5 == 1) {                                                                 2
33   local_24 = ((((param_2 | 0xbaaad0bf) & 0x3c966fda | param_2 & 0x41410000) ^ param_2 ^ 0x3882409a
34                ) & (param_2 | 0xbaaad0bf | param_2 ^ 0x45552f40)) * (param_2 + 3);
35 }
36 else if (iVar5 == 2) {                                                                 3
37   uVar2 = (param_2 ^ 0xfffffffb | param_2 ^ 4) & (param_2 ^ 0x7eb64781);
38   local_24 = (uVar2 & (param_2 & 4 | 0x8149b87a) |
39               (param_2 & 4 ^ 0x7eb64785) & (uVar2 ^ 0xffffffff)) * (param_2 ^ 0xbaaad0bf);
40 }
41 else {
42   local_24 = (param_2 + 0xbaaad0bf) * (param_2 & 5);                                  4   Original Expression Recovered
43 }
```

LLVM Optimisation + SiMBA

```
19 if (uVar2 == 2) {                                                                    3
20   uVar2 = (param_2 ^ 0x97a3fc7) & (param_2 ^ 4);
21   *piVar1 = (((uVar2 ^ (param_2 | 4)) & (param_2 & 4 | 0x97a3fc3) | uVar2 & 0xf685c03c) ^
22             ((param_2 & 4 | 0x1d62fc09) ^ param_2 ^ 0xe29d03f2) & (param_2 | 4)) *
23             (param_2 ^ 0xbaaad0bf);
24   return;
25 }
26 if (uVar2 == 1) {                                                                      2
27   *piVar1 = (param_2 + 3) * (param_2 & 0xbaaad0bf);
28   return;
29 }
30 if (uVar2 == 0) {                                                                      1
31   uVar2 = (param_2 & 0xa81b62f7 | 0x17000408) ^ param_2 & 0x57e49d08;
32   *piVar1 = ((uVar2 ^ 0xadaad4b7) & (param_2 & 0xbaaad0bf ^ 0xffffffff) |
33             (uVar2 ^ 0x12000008) & param_2 & 0xbaaad0bf) *
34             (param_2 & 0xfffffffd ^ (param_2 & 2 | 0x4fa5c831) ^ 0x4fa5c833);
35   return;
36 }
37 *piVar1 = (param_2 & 5) * (param_2 + 0xbaaad0bf);                                      4
38 return;
```

The Lion King Show: SiMBA, GAMBA and MBAs

LLVM Optimisation + SiMBA

```
19 if (uVar2 == 2) {                                                                    3
20   uVar2 = (param_2 ^ 0x97a3fc7) & (param_2 ^ 4);
21   *piVar1 = ((((uVar2 ^ (param_2 | 4)) & (param_2 & 4 | 0x97a3fc3) | uVar2 & 0xf685c03c) ^
22             ((param_2 & 4 | 0x1d62fc09) ^ param_2 ^ 0xe29d03f2) & (param_2 | 4)) *
23             (param_2 ^ 0xbaaad0bf);
24   return;
25 }
26 if (uVar2 == 1) {                                                                      2
27   *piVar1 = (param_2 + 3) * (param_2 & 0xbaaad0bf);
28   return;
29 }
30 if (uVar2 == 0) {                                                                      1
31   uVar2 = (param_2 & 0xa81b62f7 | 0x17000408) ^ param_2 & 0x57e49d08;
32   *piVar1 = ((uVar2 ^ 0xadaad4b7) & (param_2 & 0xbaaad0bf ^ 0xffffffff) |
33             (uVar2 ^ 0x12000008) & param_2 & 0xbaaad0bf) *
34             (param_2 & 0xfffffffd ^ (param_2 & 2 | 0x4fa5c831) ^ 0x4fa5c833);
35   return;
36 }
37 *piVar1 = (param_2 & 5) * (param_2 + 0xbaaad0bf);                                      4
38 return;
```

LLVM Optimisation + SiMBA + GAMBA

```
19 if (uVar2 == 2) {
20   *piVar1 = (param_2 & 0x14bcdeb4 ^ 4 | param_2 & 0xeb43214f) * (param_2 ^ 0xbaaad0bf);   3
21   return;
22 }
23 if (uVar2 == 1) {
24   *piVar1 = (param_2 & 0xbaaad0bf) * (param_2 + 3);                                      2
25   return;
26 }
27 if (uVar2 == 0) {
28   *piVar1 = ((param_2 & 0x77b39989 | 0x88006252) ^ (param_2 ^ 0xed32f2d0) & (param_2 ^ 0x9a816b59)   1
29             ) * (param_2 | 0xbaaad0bf);
30   return;
31 }
32 *piVar1 = (param_2 & 5) * (param_2 + 0xbaaad0bf);                                       4
33 return;
```

REverse

Deobfuscation: LLVM Opt + SiMBA + GAMBA

Polaris
Instruction substitution (loop 3)

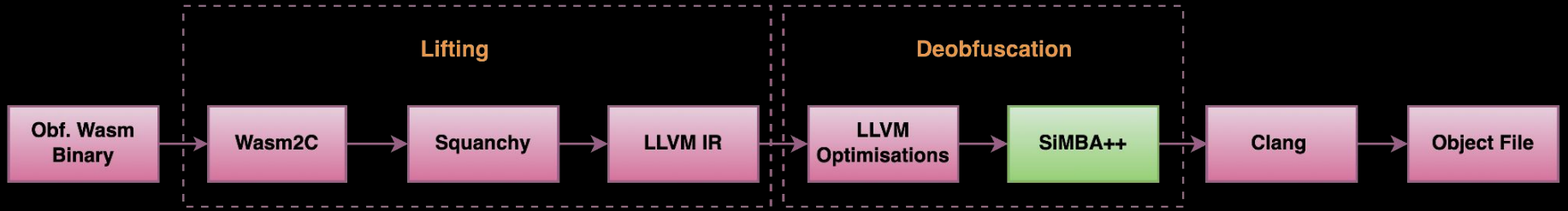
```
if (uVar1 == 0) {                                                                    1
  uVar1 = ((param1 ^ 0xffffffff) & 0x19495cff | param1 & 0xe6b6a300) ^ 0x5c1c73bf;
  uVar1 = (uVar1 ^ param1 ^ 0xffffffff) & uVar1;
  uVar5 = (param1 ^ 0x625558ec) & (param1 ^ 0xffffffff);
  uVar2 = (param1 ^ 0x625558ec) & param1;
  uVar5 = uVar5 & uVar2 | uVar5 ^ uVar2;
  uVar5 = (uVar5 ^ 0xffffffff) & 0xd8ff8853 | uVar5 & 0x270077ac;
  uVar2 = (param1 & 0x1ba026fa | (param1 ^ 0xffffffff) & 0xe45fd905) ^ 0x860a81e9;
  uVar3 = (param1 ^ 0xffffffff) & 0x625558ec | param1 ^ 0x9daaa713;
  uVar3 = (uVar3 ^ 0xffffffff) & 0x741226bb | uVar3 & 0x8bedd944;
  uVar2 = (uVar2 ^ param1) & uVar2 ^ 0xffffffff;
  uVar2 = (uVar2 ^ ((uVar3 ^ 0x741226bb) & 0x6590700b | (uVar3 ^ 0x8bedd944) & 0x9a6f8ff4) ^
           0x6590700b) & uVar2;
  uVar2 = (uVar2 ^ 0xffffffff) & 0x1eba5d23 | uVar2 & 0xe145a2dc;
  uVar1 = (uVar2 ^ 0x1eba5d23 | 0x625558ee) ^ 0xffffffff;
  uVar1 = (((uVar2 ^ 0x1eba5d23) & 0x60b6df70 | (uVar2 ^ 0xe145a2dc) & 0x9f49208f) ^ 0x2e3879e) &
          0x625558ee;
  local_c = (((uVar1 ^ 0xffffffff) & 0x7c83e458 | uVar1 & 0x837c1ba7) ^ 0x7c83e458 |
             ((uVar5 ^ 0xffffffff) & 0x721da317 | uVar5 & 0x8de25ce8) ^ 0x721da317) *
             (uVar1 & uVar1 | uVar1 ^ uVar1);
}
else if (uVar1 == 1) {                                                               2
  uVar1 = (((param1 ^ 0xffffffff) & 0x5fb8011c | param1 & 0xa047fee3) ^ 0x5fb8011c | 0x8de25ce8) &
          ((param1 ^ 0x721da317) & param1 ^ 0xffffffff);
  uVar1 = uVar1 & 0x8de25ce8 | (uVar1 ^ 0xffffffff) & 0x721da317;
  uVar1 = ((uVar1 ^ 0xffffffff) & 0x53584bcb | uVar1 & 0xaca7b434) ^ 0x53584bcb | 0x45552f40;
  uVar5 = uVar1 ^ 0xffffffff;
  local_c = ((uVar5 ^ uVar1) & uVar5) * -(-3 - param1);
}
else if (uVar1 == 2) {                                                               3
  uVar1 = (param1 | 0xfaf8dc98) & ((param1 ^ 0x5072367) & param1 ^ 0xffffffff);
  uVar1 = ((uVar1 & 0x68b867d3 | (uVar1 ^ 0xffffffff) & 0x9747982c) ^ 0x9240bb4b) & 0xbaaad0bf;
  uVar5 = ((param1 ^ 0xffffffff) & 0x368db37e | param1 & 0xc9724c81) ^ 0x8c2763c1;
  uVar5 = (uVar5 ^ param1 ^ 0xffffffff) & uVar5;
  uVar2 = (param1 ^ 0xffffffff) & 0xfffffffb | param1 & 4;
  uVar2 = (uVar2 ^ 0xfffffffb) & uVar2;
  uVar3 = (param1 | 0xb4a770ab) & (param1 ^ 0xffffffff | 0x4b588f54);
  uVar3 = uVar3 & 0x1c4a08ec | (uVar3 ^ 0xffffffff) & 0xe3b5f713;
  uVar4 = (uVar2 ^ 0xffffffff | uVar3 ^ 0xa8ed7843) ^ 0xffffffff;
  uVar1 = (uVar2 ^ 0xffffffff) & (uVar3 ^ 0x571287bc) | uVar2 & (uVar3 ^ 0xa8ed7843);
  local_c = ((uVar1 ^ 0xffffffff | uVar5 ^ 0xffffffff) &
             ((uVar1 ^ 0xffffffff) & uVar5 | uVar1 & (uVar5 ^ 0xffffffff)) ^ 0xffffffff) ^
            0xffffffff) * (uVar4 & uVar1 | uVar4 ^ uVar1);
}
else {                                                                               4
  uVar1 = ((param1 ^ 0xffffffff) & 0x769a091f ^ 0x769a091f) &
          ((param1 ^ 0xffffffff) & param1 ^ 0xffffffff);
  uVar5 = (uVar1 ^ 0x8bedd944) & uVar1;
  uVar1 = (uVar1 | 0x741226bb) ^ 0xffffffff;
  uVar1 = uVar5 & uVar1 | uVar5 ^ uVar1;
  local_c = -(-0x374a3fe6 - (-0x374a3fe6 - (0x45552f41 - param1))) *
            ((((uVar1 ^ 0xffffffff) & 0x52a311c3 | uVar1 & 0xad5cee3c) ^ 0x26b1377d) & 5);
}
return local_c;
```

LLVM Optimisation + SiMBA + GAMBA

```
uVar1 = param_2 & 3;
iVar3 = (param_2 ^ 0xbaaad0bf) * (param_2 | 4);                                      3
if (uVar1 != 2) {
  iVar3 = (param_2 + 0xbaaad0bf) * (param_2 & 5);                                    4
}
iVar2 = (param_2 + 2) * (param_2 | 0xbaaad0bf);                                      1
if (uVar1 != 0) {
  iVar2 = (param_2 & 0xbaaad0bd) * (param_2 + 3);                                    2
}
if (uVar1 < 2) {
  iVar3 = iVar2;
}
return iVar3;
```

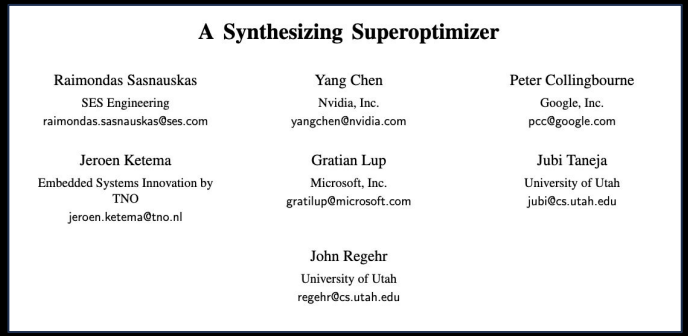
Completely Recovered

Pipeline


Lifting
Deobfuscation
Obf. Wasm Binary
Wasm2C
Squanchy
LLVM IR
LLVM Optimisations
SiMBA++
Clang
Object File

SOUPER: Supercharging Deobfuscation

- Souper - a synthesis-based superoptimizer for a domain specific intermediate representation (IR) that resembles a purely functional, control-flow-free subset of LLVM IR
- Souper can run as an LLVM optimization pass, ensuring that its code improvements can be exploited by other passes, such as constant propagation and dead code elimination.
- Synthesise optimisations
  - Counterexample guided inductive synthesis (CEGIS)
    - Multiple RHS generated, cheapest among them is chosen.
  - Dataflow
- [Souper on Github]



A Synthesizing Superoptimizer

Raimondas Sasnauskas
SES Engineering
raimondas.sasnauskas@ses.com

Yang Chen
Nvidia, Inc.
yangchen@nvidia.com

Peter Collingbourne
Google, Inc.
pcc@google.com

Jeroen Ketema
Embedded Systems Innovation by TNO
jeroen.ketema@tno.nl

Gratian Lup
Microsoft, Inc.
gratilup@microsoft.com

Jubi Taneja
University of Utah
jubi@cs.utah.edu

John Regehr
University of Utah
regehr@cs.utah.edu

Super SOUPER

- Converting a sequence of LLVM-IR instructions into an SMT formula and use several SMT solvers to discover additional peephole optimizations.
- Can resolve opaque predicates
- Caches SMT queries into an external Redis database.

```
# Souper - synthesis using CEGIS algorithm
```

```
opt -load-pass-plugin $SOUPER --souper-use-cegis -S -O3
--souper-synthesis-debug-level=1 obf.ll -o deobf_souper.ll
```

```
# Souper - synthesis using dataflow pruning algorithm
```

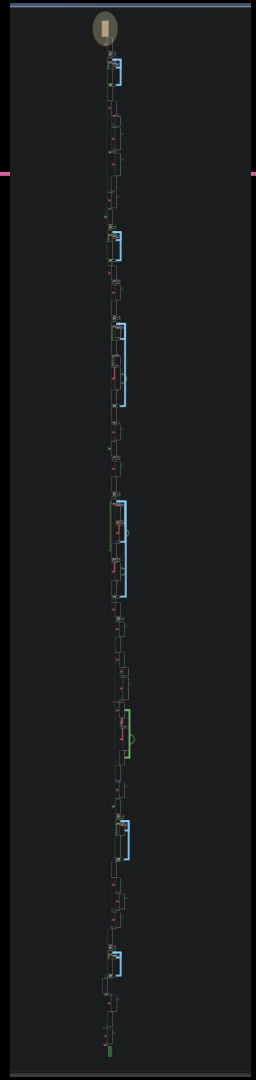
```
opt -load-pass-plugin $SOUPER --souper-synthesis-debug-level=1 -S -O3
--souper-use-cegis --souper-dataflow-pruning-heavy obf.ll -o deobf_souper.ll
```

Deobfuscation: SOUPER

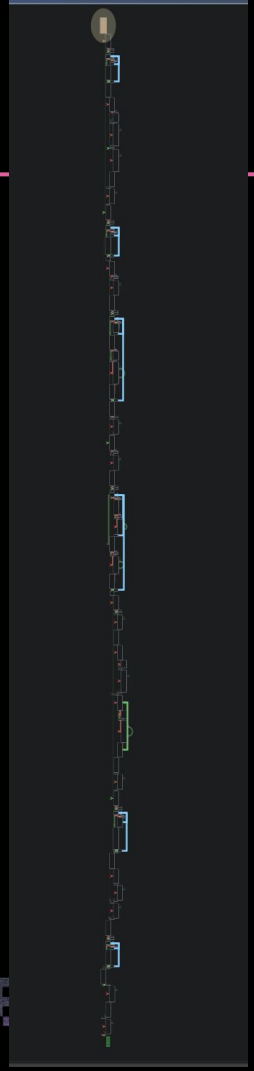
Hikari
Bogus Control Flow (loop 2)

Opaque Predicate

```
14  if (uVar2 == 0) {
15    if ((((uRam00010ae8 | uRam00010aec) ^ 0x3c2e5570) & 0x97ff2bd7) + 0x64293ba9 < 0xe0465c21) {
16      bVar1 = true;
17    }
18    else {
19      bVar1 = false;
20    }
21    while( true ) {
22      while (bVar1) {
23        bVar1 = false;
24      }
25      if (0x5fd76e94 < ((iRam00010af0 + iRam00010af4 ^ 0x41005e8aU) + 0x63d028ff) * 0x65edfa51)
26      break;
27      bVar1 = true;
28    }
29    if ((iRam00010af8 * iRam00010afc + 0xd9ef92c7U | 0xba1e4315) + 0xce83d3a0 < 0x8bb488b1) {
30      do {
31      } while (((uRam00010c30 ^ uRam00010c34) + 0x6f44ee27 & 0x7ca03c77) * 0x5ed0b58a == -0x29120a04
32              );
33    }
34    do {
35      local_c = (param1 | 0xbaaad0bf) * (param1 ^ 2);
36    } while (((uRam00010b00 / uRam00010b04 | 0xbabf5164) * -0x32ee4c95 & 0x67c7c119) < 0x20a96022);
37    do {
38    } while ((uRam00010b10 / uRam00010b14 + 0xc8de4516) / 0x936b17aa == 0xa9f0a4ac);
39  }
```

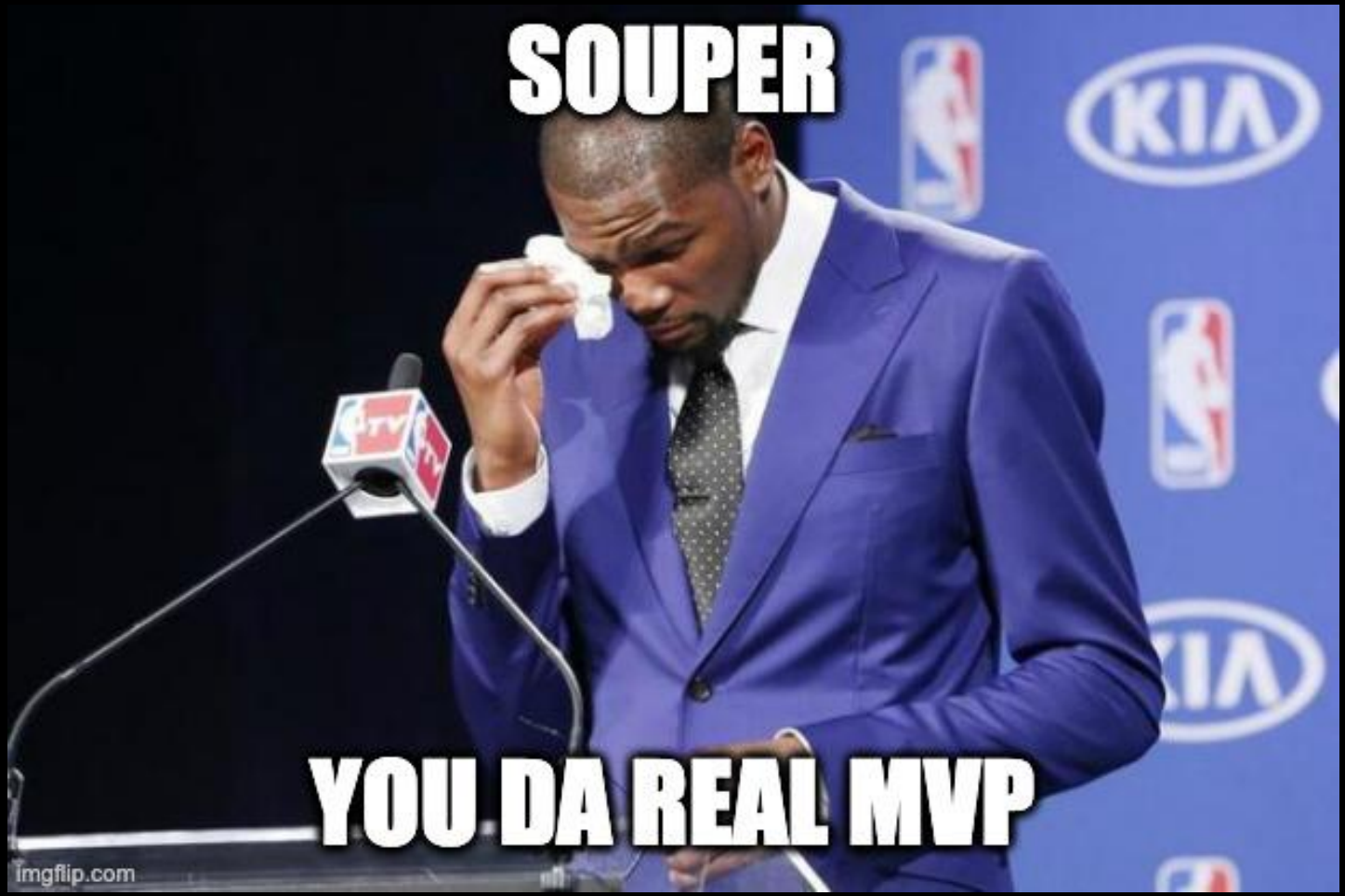


Deobfuscation: SOUPER

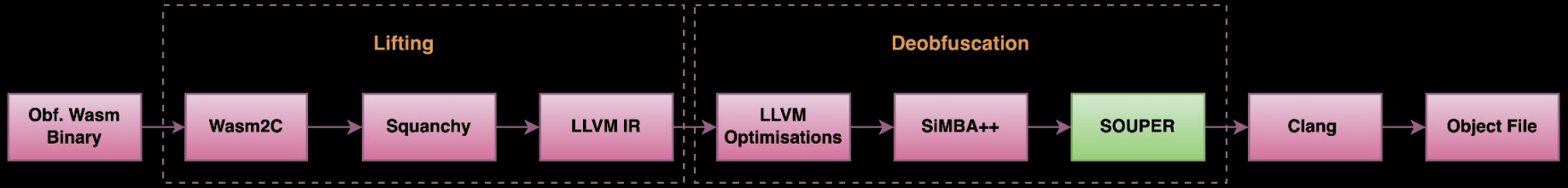


SOUPER

```
 9  uVar1 = param_2 & 3;
10  iVar3 = (param_2 ^ 0xbaaad0bf) * (param_2 | 4);          4
11  if (uVar1 != 2) {
12    iVar3 = (param_2 & 5) * (param_2 + 0xbaaad0bf);        3
13  }
14  iVar2 = (param_2 | 0xbaaad0bf) * (param_2 ^ 2);          1
15  if (uVar1 != 0) {
16    iVar2 = (param_2 & 0xbaaad0bf) * (param_2 + 3);        2
17  }
18  if (uVar1 < 2) {
19    iVar3 = iVar2;
20  }
21  return iVar3;
22}
```


SOUPER
KIA
KIA
YOU DA REAL MVP
imgflip.com

Pipeline


Lifting
Deobfuscation
Obf. Wasm Binary
Wasm2C
Squanchy
LLVM IR
LLVM Optimisations
SiMBA++
SOUPER
Clang
Object File

Deobfuscation: SOUPER vs Hikari (Sub=1, bogus=1, split=1)

Long complex obfuscated code

```
 8  uVar1 = param_2 & 3;
 9  if (uVar1 < 2) {
10    iVar2 = (param_2 | 0xbaaad0bf) * (param_2 + 2);   1
11    if (uVar1 != 0) {
12      iVar2 = (param_2 + 3) * (param_2 & 0xbaaad0bf);  2
13    }
14    return iVar2;
15  }
16  if (uVar1 == 3) {
17    return (param_2 & 5) * (param_2 + 0xbaaad0bf);   4
18  }
19  return (param_2 & 0xbaaad0b8 ^ 0xbaaad0b9) * (param_2 & 0xbaaad0bf) +  3
20         (param_2 & 0x45552f44 ^ 4) * (param_2 | 0xbaaad0bf);
21}
```

Real World Application

WebAssembly Malwares

- Steady increase in usage of Wasm for cryptomining in browsers
  - Compared to JS, Wasm is fast in performing hashing operations
  - Monero is the most used cryptocurrency for cryptomining.

- Carbera-Arteaga et. al. demonstrate use of *wasm-mutate* to evade detection (https://arxiv.org/pdf/2309.07638)

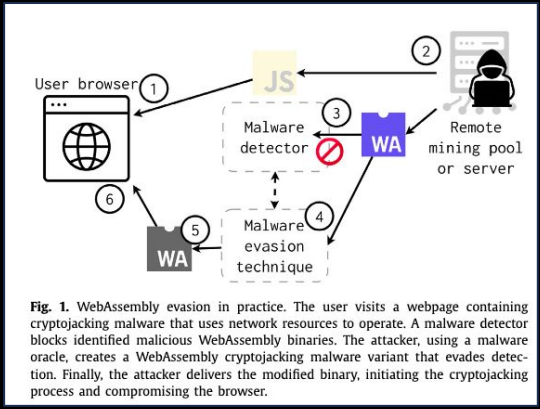


Fig. 1. WebAssembly evasion in practice. The user visits a webpage containing cryptojacking malware that uses network resources to operate. A malware detector blocks identified malicious WebAssembly binaries. The attacker, using a malware oracle, creates a WebAssembly cryptojacking malware variant that evades detection. Finally, the attacker delivers the modified binary, initiating the cryptojacking process and compromising the browser.

WebAssembly diversification for malware evasion

Javier Cabrera-Arteaga*, Martin Monperrus, Tim Toady, Benoit Baudry

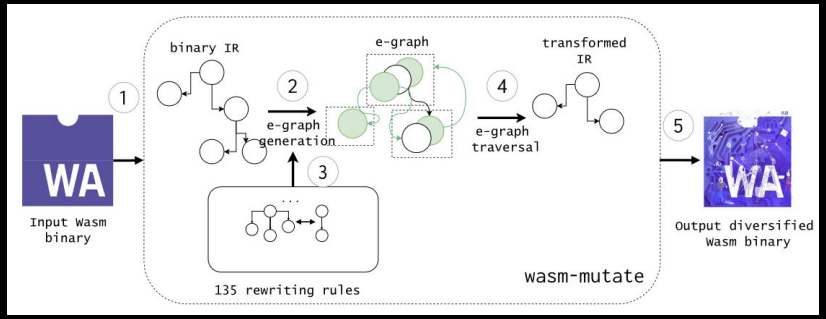
KTH Royal Institute of Technology, Stockholm, Sweden

Table 2
Baseline evasion algorithm for VirusTotal. The table contains as columns: the hash of the program, the number of initial VirusTotal detectors, the maximum number of evaded antivirus vendors and the mean number of iterations needed to generate a variant that fully evades detection. The rows of the table are sorted by the number of initial detectors, from left to right and top to bottom.

Hash	#D	Max. #evaded	Mean #trans.
47d29959	31	26 (83.8%)	N/A
9d30e7f0	30	24 (80.0%)	N/A
8ebf4e44	26	21 (80.7%)	N/A
dc11d82d	20	20 (100.0%)	355
0d996462	19	19 (100.0%)	401
a32a6f4b	18	18 (100.0%)	635
fbdd1efa	18	18 (100.0%)	310
d2141ff2	9	9 (100.0%)	461
aafff587	6	6 (100.0%)	484
046dc081	6	6 (100.0%)	404
643116ff	6	6 (100.0%)	144
15b86a25	4	4 (100.0%)	253
006b2fb6	4	4 (100.0%)	282
942be4f7	4	4 (100.0%)	200
7c36f462	4	4 (100.0%)	236
fb15929f	4	4 (100.0%)	297
24aae13a	4	4 (100.0%)	252
000415b2	3	3 (100.0%)	302
4cbdbbb1	3	3 (100.0%)	295
65debcbe	2	2 (100.0%)	131
59955b4c	2	2 (100.0%)	130
89a3645c	2	2 (100.0%)	431
a74a7cb8	2	2 (100.0%)	124
119c53eb	2	2 (100.0%)	104
089dd312	2	2 (100.0%)	153
c1be4071	2	2 (100.0%)	130
dceaf65b	2	2 (100.0%)	140
6b8c7899	2	2 (100.0%)	143
a27b45ef	2	2 (100.0%)	145
68ca7c0e	2	2 (100.0%)	137
f0b24409	2	2 (100.0%)	127
5bc53343	2	2 (100.0%)	118
e09c32c5	1	1 (100.0%)	120

Wasm-mutate

- *wasm-mutate* automatically transforms a WebAssembly binary program into a variant binary program that preserves the original functionality.
- 3 kind of transformations
  - Peephole
    - ~135 rewrite rules.
  - Module structure transformation
    - Add new type, new function, new export etc.
  - Control flow graph
    - Loop unrolling, swap conditional branches.
- Robust
  - Written in Rust
- *wasm-mutate* output needs to verified with *wasm-validate*
  - Some transformation break the WASM file
  - *WebAssembly Diversification for Malware Evasion*
    - https://arxiv.org/pdf/2212.08427



DEMO

Use Case: wasm-mutate

- wasm-mutate
  - 3000 (real) iterations are applied
  - *100%* of mutations are removed
  - Code is normalized and matches 100% the original code!
    - Our approach fully recovers the function ... (and optimizes it!)

Use Case: Deobfuscating Malwares

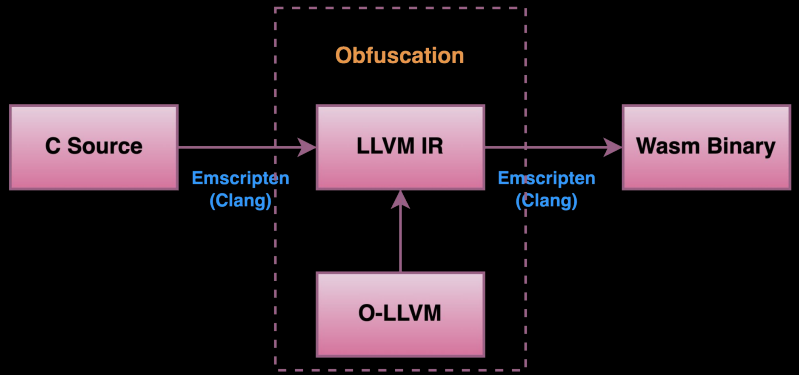
- CryptoNight, CryptoNight Obfuscated
  - Deobfuscated functions by Squanchy match non-obfuscated functions
  - https://www.crowdstrike.com/en-us/blog/ecriminals-increasingly-use-webassembly-to-hide-malware/
  - https://arxiv.org/abs/2403.15197

Future Work

- Re-compile to equivalent (deobfuscated) Wasm binary
- Improve deobfuscation time for squanchy
- Iterative Control Flow Graph recovery

Conclusion

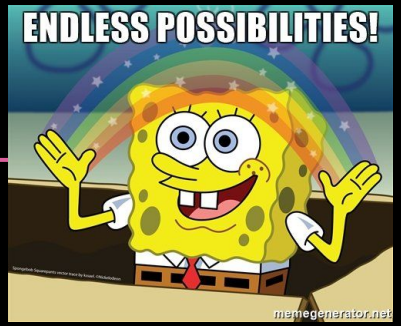
- Tooling
  - Existing tooling can be reused
  - Obfuscation - Polaris, O-LLVM, ~~Wasmixer~~
  - Deobfuscation - LLVM, SiMBA++, SOUPER
  - Symbolic Execution - KLEE, ~~Manticore~~, ~~SeeWasm~~



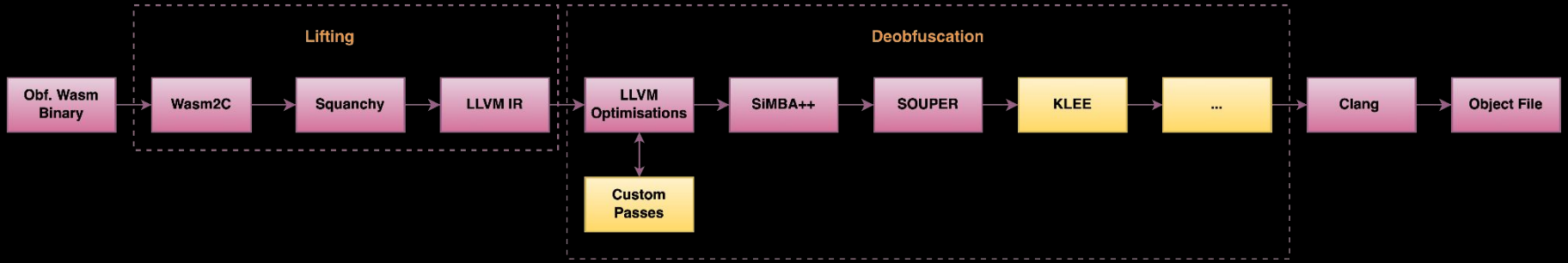
- Wasm obfuscation

- Squanchy: Lifting Wasm to LLVM IR
  - There are shortcomings with WAMRC and other tools.
  - Wasm2c + Squanchy works great.

Conclusion



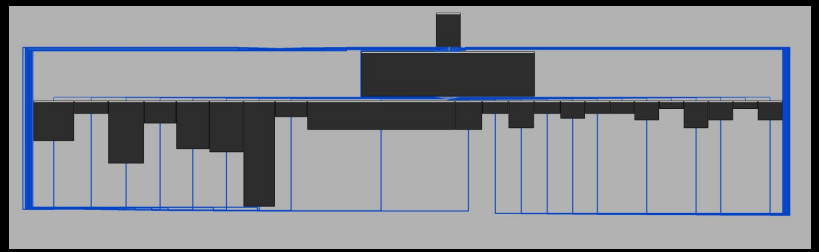
- Deobfuscation using LLVM Optimizations
  - LLVM + SiMBA + GAMBA + SOUPER + ...

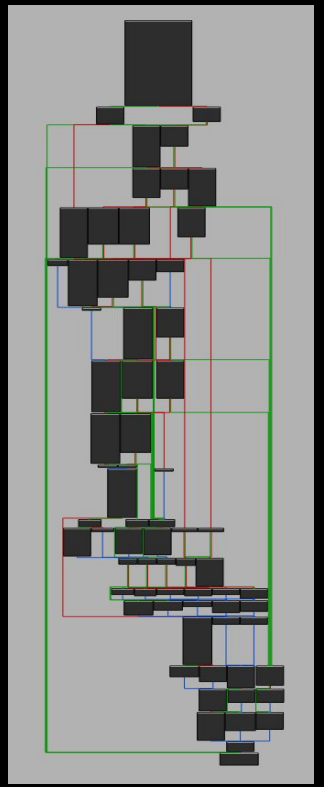


Conclusion

- Malware Normalisation
  - Wasm-mutate output can be simplified
  - Cryptonight malware simplified

- Trade offer





Thank You!!

- Slides - https://github.com/su-vikas/Presentations
- *Squanchy* - https://github.com/pgarba/Squanchy

RE//verse