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)

```

int w2c_squanchy_calc_0(undefined8 param_1,uint param_2)

{
  uint uVar1;
  int iVar2;
  int iVar3;

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

```

About Us

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

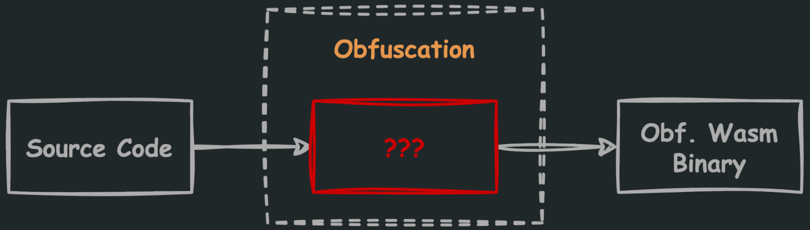
- **Peter Garba**
  - Principal Software Security Engineer at *Thales CDI*, *Singapore*
  - Product Owner
  - Author of the Thales internal obfuscation tools
  - Passionate reverse engineer at night.

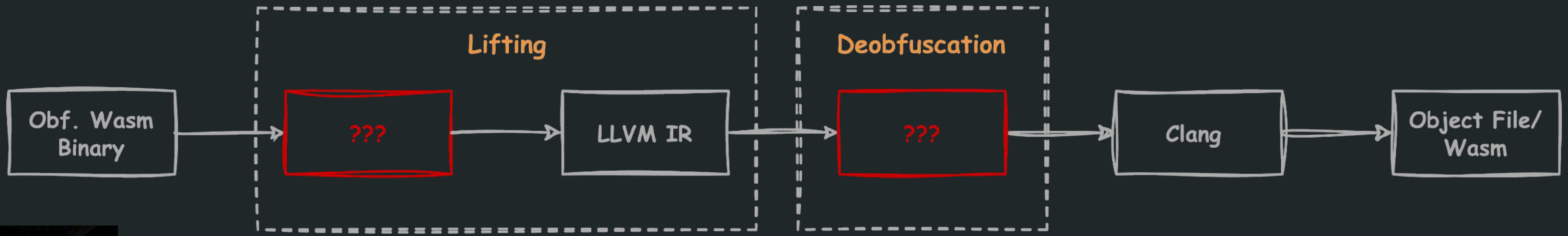
Motivation

THALES
Building a future we can all trust

Problem Statement

1. Is Wasm secure for us?
2. Obfuscate Wasm binaries
3. Lifting Wasm to LLVM IR
4. Deobfuscate Wasm binaries & recover original logic





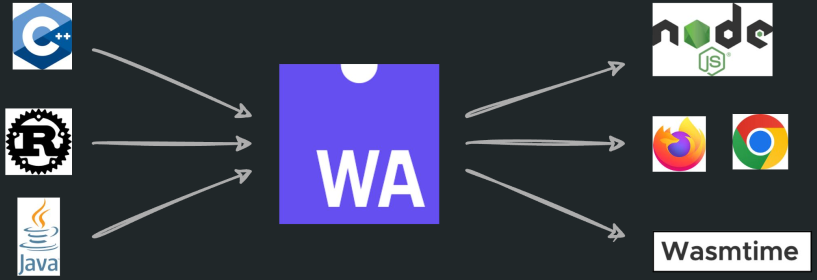
Achievements

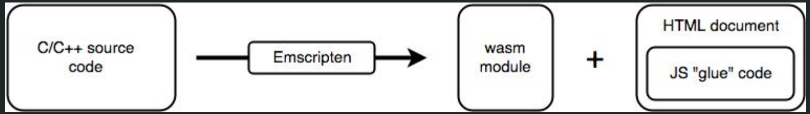
- Demonstrating use of existing tooling for Wasm
  - Obfuscation
  - Deobfuscation
- Lifting Wasm to LLVM IR - Squanchy
- Automated deobfuscation of Wasm

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
- Wide adoption
  - Games
  - Big web apps - Google Earth
  - Blockchain smart contracts
  - ...





WebAssembly Essentials

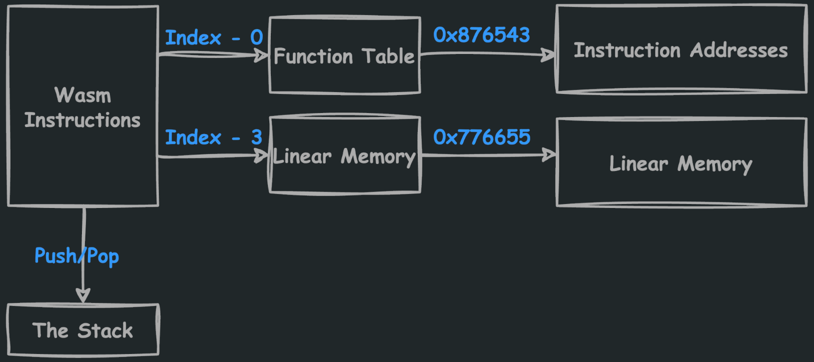
- Each Wasm program is a single file of code - Module.

- Module is organized in sections.
  - Sections - Export, imports, globals, functions etc.

- 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

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)

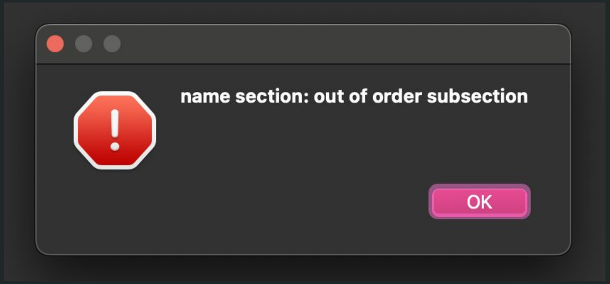


WebAssembly Tooling

WebAssembly Tooling

- [WebAssembly Binary Toolkit]
- [Wasm-tools]
- Ghidra
  - Using a [Wasm plugin]
  - Used to view decompiled Wasm
- 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



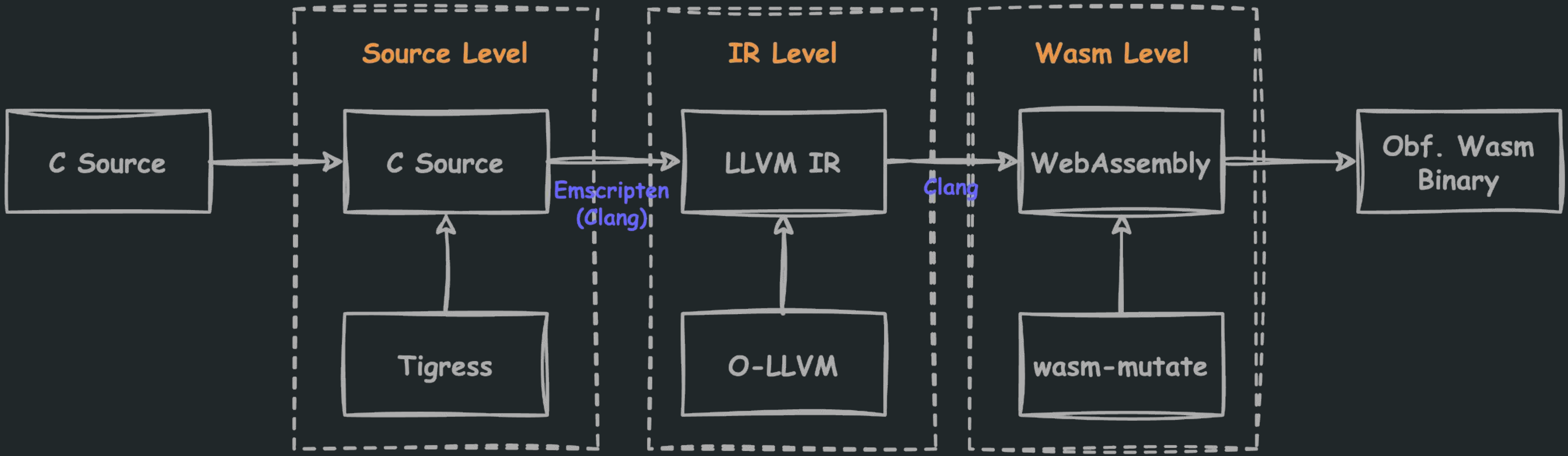
WebAssembly Obfuscation

Obfuscation

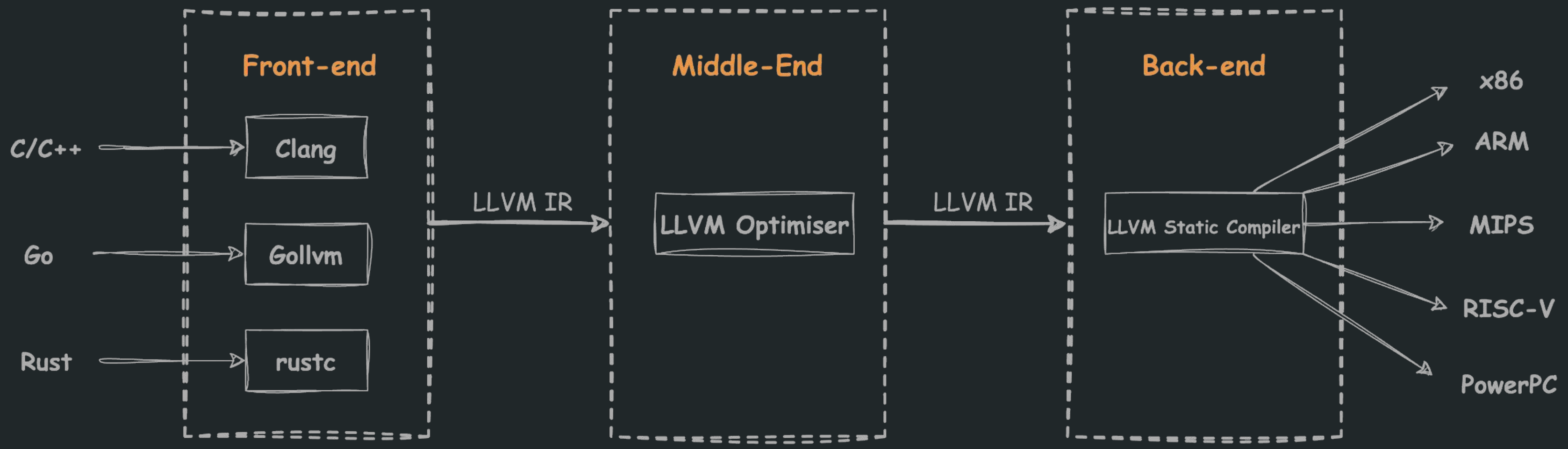
- Obfuscation: Process of making a program harder to understand while preserving the original program's behavior.

- 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: O-LLVM Instruction Substitution

```
if (mod == 0) result = (n | 0xbaaad0bf) * (2 ^ n)
```

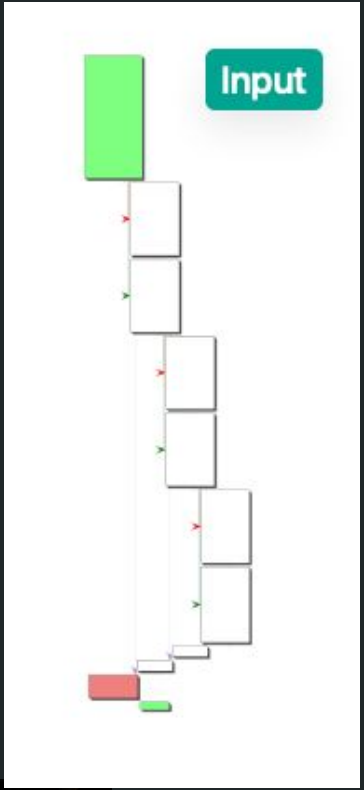
O-LLVM Instruction Substitution (Loop=1)

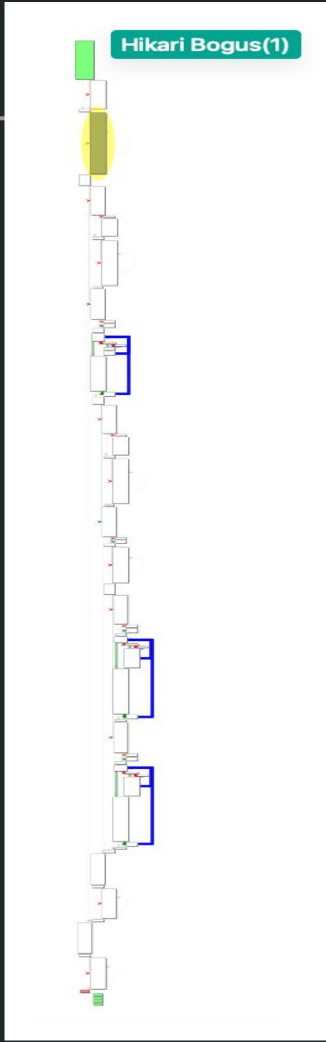
```
if (iVar1 == 0) {
  local_10 = (param1 & 0xbaaad0bf | param1 ^ 0xbaaad0bf) *
             (((param1 ^ 0xffffffff) & 0xbcec65b1 | param1 & 0x43139a4e) ^ 0xbcec65b3);}
```

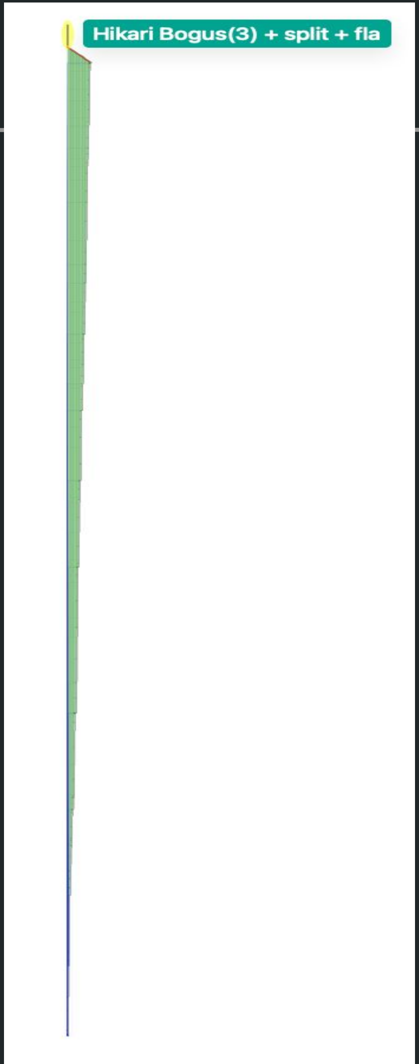
O-LLVM Instruction Substitution (Loop=3)

```
if (iVar1 == 0) {
  uVar10 = param1 & 0xc9e645ce | (param1 ^ 0xffffffff) & 0x3619ba31;
  uVar11 = ((uVar10 ^ 0x3619ba31) & 0x5c2ea1f5 | (uVar10 ^ 0xc9e645ce) & 0xa3d15e0a) ^ 0xf1571e9d
           | uVar10 ^ 0x3619ba31;
  uVar12 = ((param1 ^ 0xffffffff) & 0xad79bf68 | param1 & 0x52864097) ^ 0xffffffff |
           param1 ^ 0xffffffff;
  uVar4 = uVar11 | uVar12;
  uVar11 = (uVar12 ^ 0xffffffff) & uVar11 | (uVar11 ^ 0xffffffff) & uVar12;
  uVar12 = uVar11 ^ 0xffffffff;
  uVar11 = (uVar4 & 0xaea378c3 | (uVar4 ^ 0xffffffff) & 0x515c873c) ^
           (uVar12 & 0xaea378c3 | uVar11 & 0x515c873c) | (uVar4 | uVar12) ^ 0xffffffff;
  uVar12 = ((uVar11 ^ 0xffffffff) & 0xcb73214a | uVar11 & 0x348cdeb5) ^ 0xcb73214a | 0x604af11c;
  uVar11 = uVar11 ^ 0xffffffff | 0x9fb50ee3;
  uVar11 = (uVar12 & 0x5835bf98 | (uVar12 ^ 0xffffffff) & 0xa7ca4067) ^
           (uVar11 & 0x5835bf98 | (uVar11 ^ 0xffffffff) & 0xa7ca4067) |
           (uVar12 | uVar11) ^ 0xffffffff;
  uVar11 = (uVar11 ^ 0xffffffff) & 0x88666134 | uVar11 & 0x77999ecb;
  uVar12 = uVar10 ^ 0x3619ba31 | 0xbaaad0bf;
  uVar10 = (uVar10 ^ 0x3619ba31) & 0x565b27a0 | (uVar10 ^ 0xc9e645ce) & 0xa9a4d85f;
  uVar4 = uVar10 ^ 0xecf1f71f;
  uVar12 = (uVar12 & 0xc4fa1585 | (uVar12 ^ 0xffffffff) & 0x3b05ea7a) ^
           (uVar4 & 0xc4fa1585 | (uVar10 ^ 0x130e08e0) & 0x3b05ea7a) |
           (uVar12 | uVar4) ^ 0xffffffff;
  uVar10 = (((uVar12 ^ 0xffffffff) & 0xbe3c86ad | uVar12 & 0x41c37952) ^ 0xbe3c86ad | 0xe6842217)
           ^ 0xffffffff;
  uVar12 = (uVar12 ^ 0x197bdde8) & uVar12;
  uVar10 = uVar10 & uVar12 | uVar10 ^ uVar12;
  uVar10 = (uVar10 ^ 0xffffffff) & 0x9d11e123 | uVar10 & 0x62ee1edc;
  uVar10 = (uVar10 ^ 0x846a3ccb) & 0x61675078 | (uVar10 ^ 0x7b95c334) & 0x9e98af87;
  uVar10 = (uVar10 ^ 0x61675078) & 0xa63617bd | (uVar10 ^ 0x9e98af87) & 0x59c9e842;
  uVar12 = (uVar11 ^ uVar10 ^ 0xa63617bd) & uVar11;
  uVar10 = ((uVar11 ^ 0xffffffff) & 0xbcc6bdd4 | uVar11 & 0x4339422b) ^
           ((uVar10 ^ 0xa63617bd) & 0xbcc6bdd4 | (uVar10 ^ 0x59c9e842) & 0x4339422b);
  uVar11 = uVar12 ^ 0xffffffff;
  uVar4 = uVar10 ^ 0xffffffff;
  uVar5 = (param1 ^ 0xffffffff) & 0xaf91567b | param1 & 0x506ea984;
  uVar6 = uVar5 ^ 0xaf91567b;
  uVar5 = (uVar6 & 0xe30d84a7 | (uVar5 ^ 0x506ea984) & 0x1cf27b58) ^ 0xe30d84a5 |
          (uVar6 | 0xfffffffd) ^ 0xffffffff;
  uVar6 = (((param1 ^ 0xffffffff) & 0x7ce0ffb1 | param1 & 0x831f004e) ^ 0xea109553) & 0x96f06ae2;
  uVar7 = (param1 ^ 0xffffffff | 0x96f06ae2) ^ 0xffffffff;
  uVar8 = uVar6 ^ uVar7;
  uVar6 = (uVar8 ^ 0xffffffff) & 0x690f951d | uVar6 & uVar7 | uVar8 & 0x96f06ae2;
  uVar6 = uVar6 & 0xba8c5c70 | (uVar6 ^ 0xffffffff) & 0x4573a38f;
  uVar6 = (uVar6 ^ 0x4573a38f) & (uVar6 ^ 0xba8c5c72);
  uVar7 = (uVar5 ^ 0xcb0a5819) & uVar5;
  uVar8 = (uVar5 | 0x34f5a7e6) ^ 0xffffffff;
  uVar9 = (uVar6 ^ 0x34f5a7e6) & (uVar6 ^ 0xffffffff);
  uVar2 = (uVar6 ^ 0x34f5a7e6) & uVar6;
  uVar7 = uVar7 & uVar8 | uVar7 ^ uVar8;
  uVar8 = uVar9 ^ 0xffffffff;
  uVar3 = uVar2 ^ 0xffffffff;
  uVar8 = (uVar8 & 0xd3d541ce | uVar9 & 0x2c2abe31) ^ (uVar3 & 0xd3d541ce | uVar2 & 0x2c2abe31) |
          (uVar8 | uVar3) ^ 0xffffffff;
  uVar7 = ((uVar7 ^ 0xffffffff) & 0xefd84b11 | uVar7 & 0x1027b4ee) ^
          ((uVar8 ^ 0xffffffff) & 0xefd84b11 | uVar8 & 0x1027b4ee);
  uVar5 = ((uVar5 ^ 0xffffffff) & 0xffb73ee8 | uVar5 & 0x48c117) ^
          (uVar6 & 0xffb73ee8 | (uVar6 ^ 0xffffffff) & 0x48c117) |
          (uVar5 ^ 0xffffffff | uVar6) ^ 0xffffffff;
  uVar5 = (uVar5 ^ 0xffffffff) & 0xa95ee2be | uVar5 & 0x56a11d41;
  uVar6 = uVar5 ^ 0xa95ee2be;
  uVar8 = uVar7 ^ 0xffffffff;
  local_10 = ((uVar11 & 0x1e98326 | uVar12 & 0xfe167cd9) ^
              (uVar4 & 0x1e98326 | uVar10 & 0xfe167cd9) | (uVar11 | uVar4) ^ 0xffffffff) *
             ((uVar8 & 0xc8a77567 | uVar7 & 0x37588a98) ^
              (uVar6 & 0xc8a77567 | (uVar5 ^ 0x56a11d41) & 0x37588a98) |
             (uVar8 | uVar6) ^ 0xffffffff);
}
```

Obfuscation: Control Flow

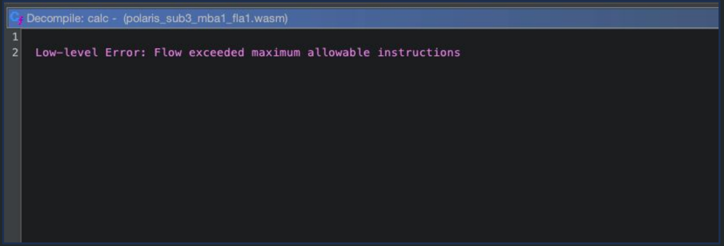


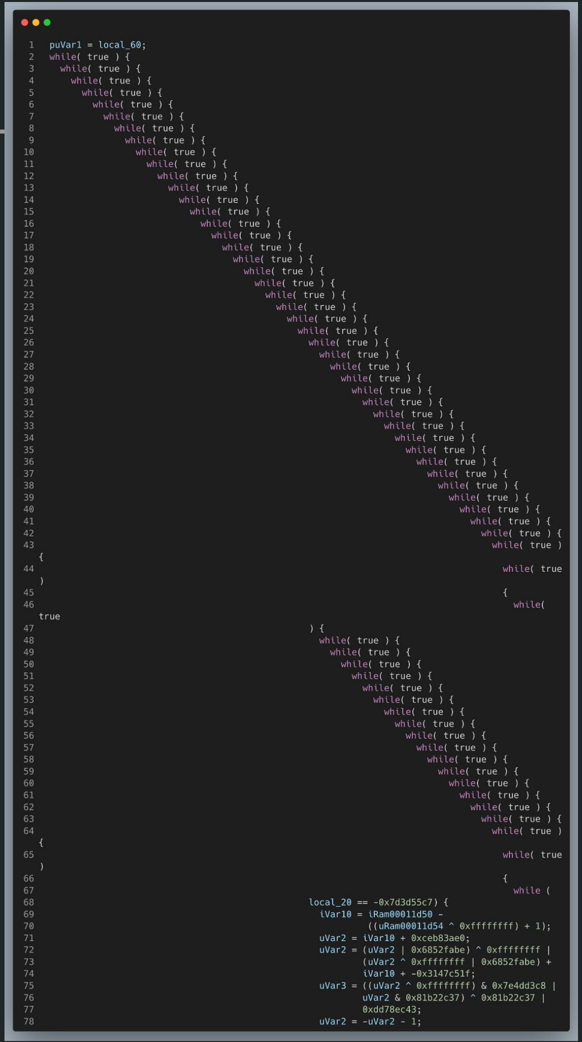




Obfuscation: Complexity Increases

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





black hat ASIA 2025

WebAssembly Deobfuscation

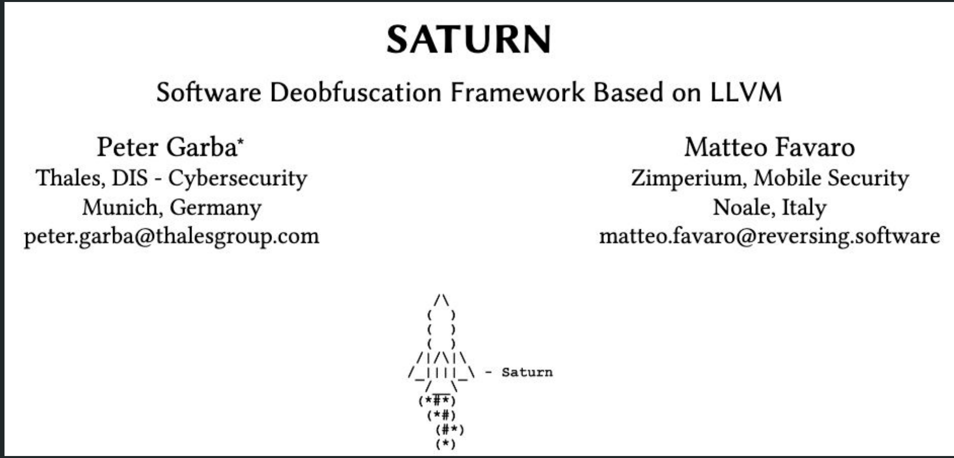
Deobfuscation

- Revert the transformations (sometimes impossible)
- Simplify the code to facilitate further analysis

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

- *Modern obfuscation*
  - Source Code Level
  - Intermediate representation level
- *Modern deobfuscation*
  - Several Intermediate Languages at different abstract layers
  - Based on generic optimization tools

SATURN: Compiler Based Deobfuscation



SATURN

Software Deobfuscation Framework Based on LLVM

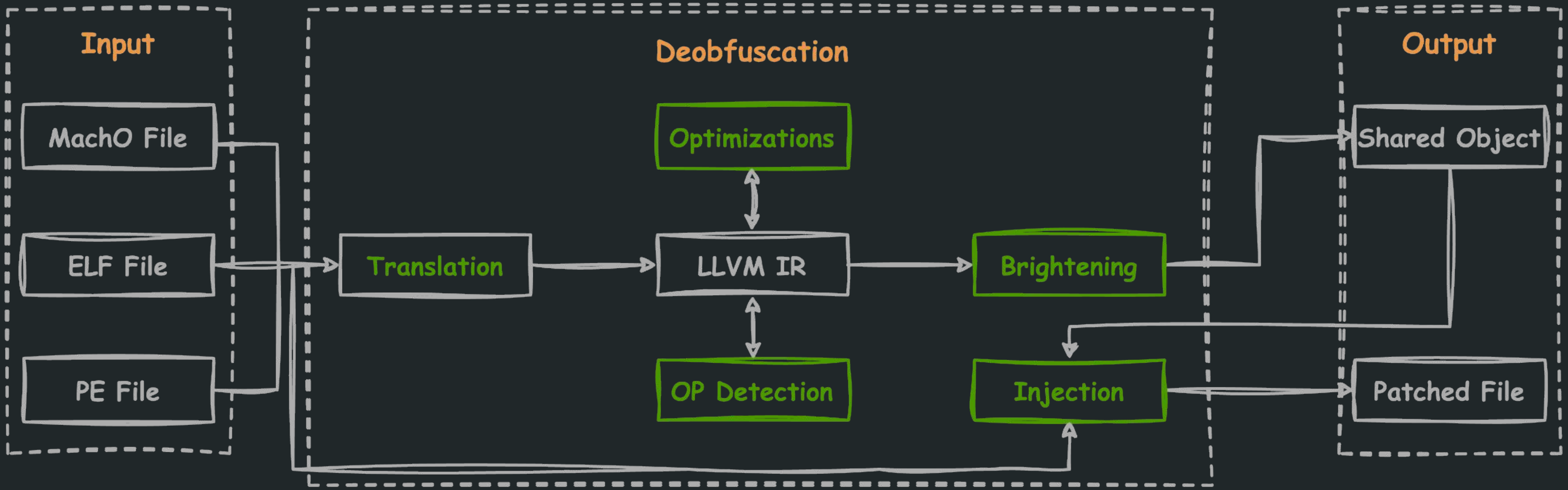
Peter Garba*
Thales, DIS - Cybersecurity
Munich, Germany
peter.garba@thalesgroup.com

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

```
   /\
  (  )
  (  )
  (  )
 /|/\|\
/_||||_\  - Saturn
  /__\
 (*#*)
  (*#)
   (#*)
   (*)
```

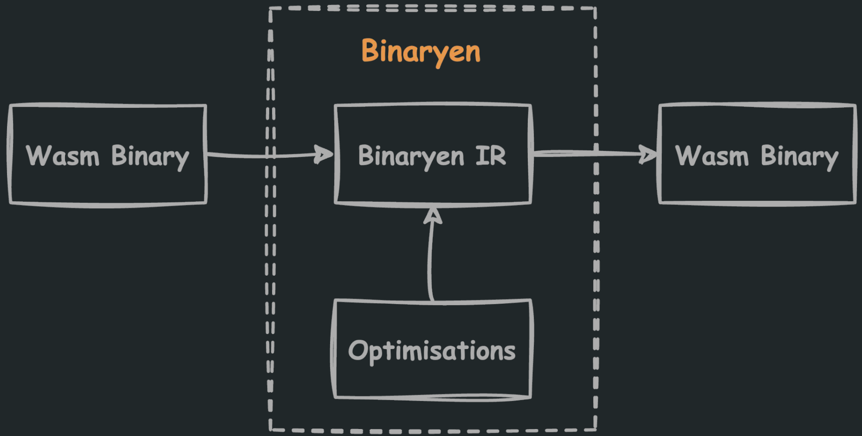
- Generic approach for deobfuscation based on LLVM compiler infrastructure.
- Weaken certain obfuscation, and in best case completely remove them.

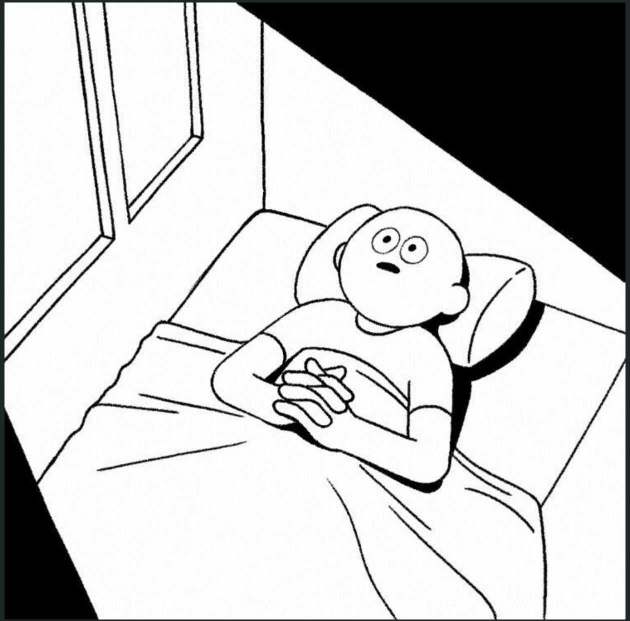
SATURN: Compiler Based Deobfuscation



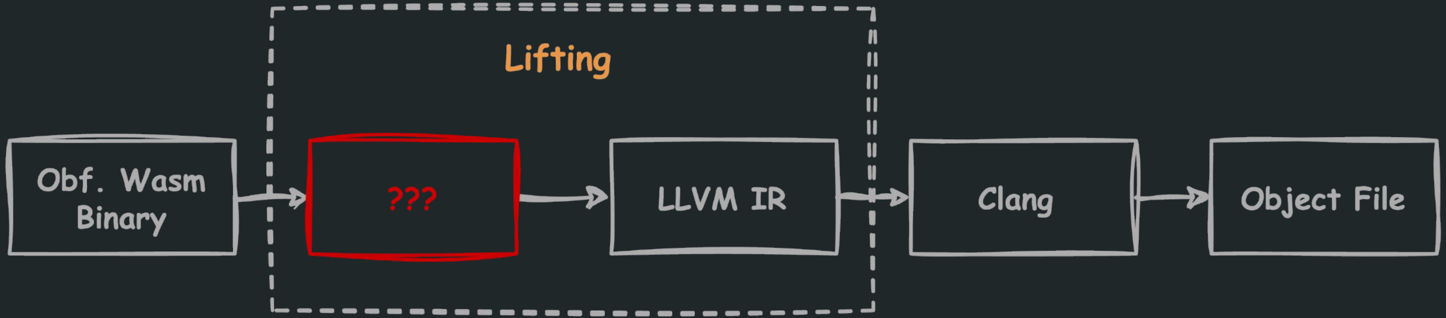
Binaryen

- 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 😞😡



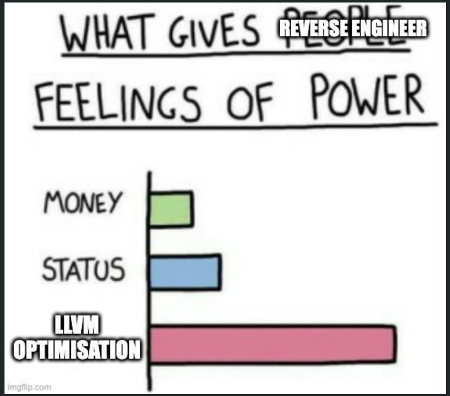


Lifting to LLVM IR



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	Wasm Opcode Specification	Lifted C 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
```

Get Local

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

Integer Add

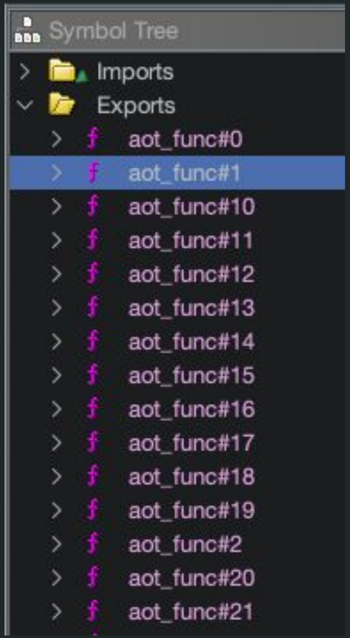
Mnemonic	Opcode	Signature
i32.add	0x6a	(i32, i32) : (i32)
i64.add	0x7c	(i64, i64) : (i64)

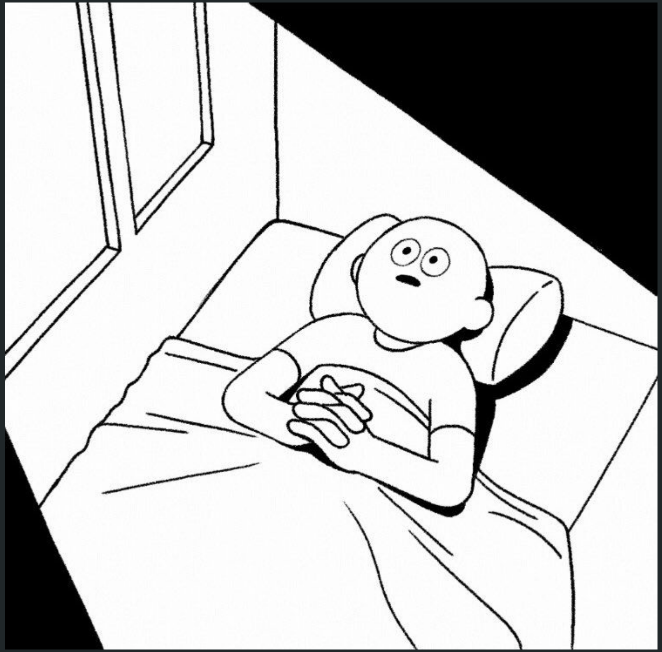
Mnemonic	Opcode	Immediates	Signature
i32.const	0x41	$value : varsint32	() : (i32)

```c
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;
}
```

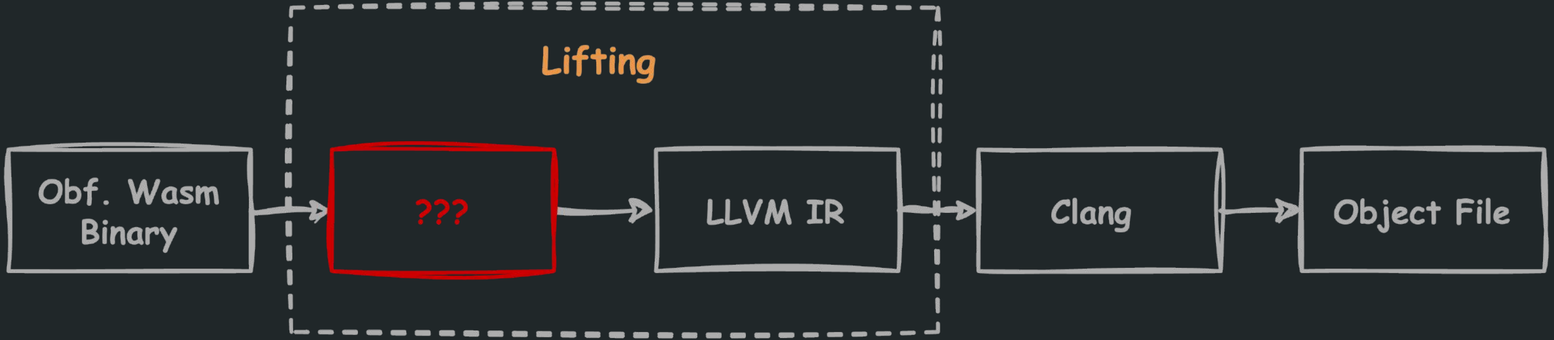
Wasm Code Lifting: Using WAMRC

- WebAssembly Micro Runtime (WAMR)
  - Lightweight, standalone Wasm runtime
  - WAMR Compiler (WAMRC)
    - The AOT compiler to compile Wasm file into AOT file
- Shortcomings
  - Symbols information is lost
  - Generated LLVM IR does not contain various tables (global's table, function table) => LLVM optimisations don't work





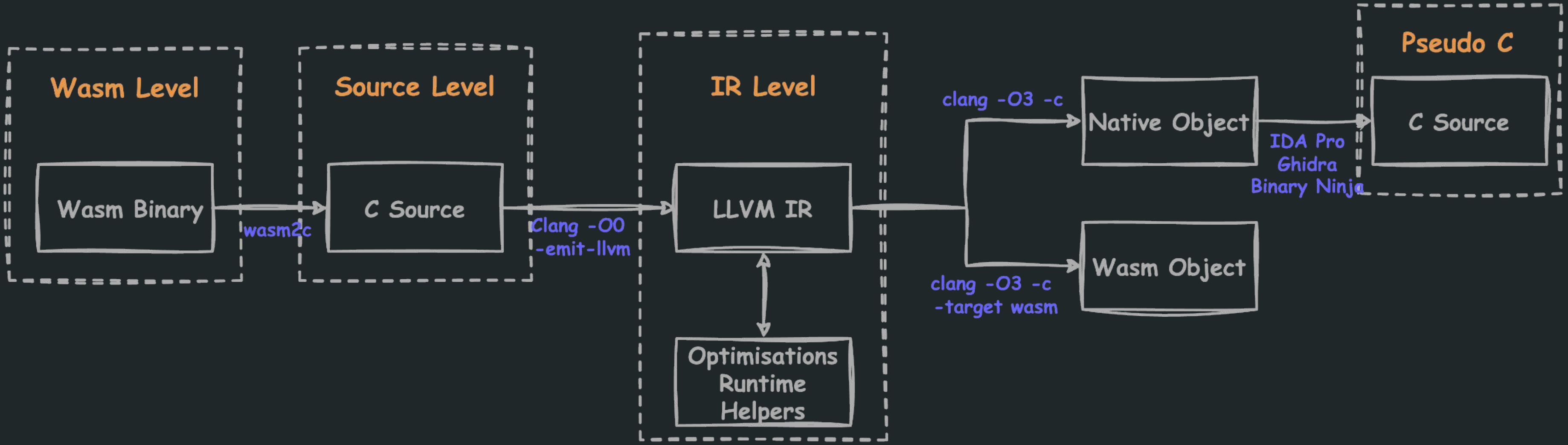
Wasm Lifter Problem Persists



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 and 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

- Shortcomings
  - Runtime information (tables…) not available for each function
  - Code doesn't fold

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)

```
        .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;
}
```

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;
};
```

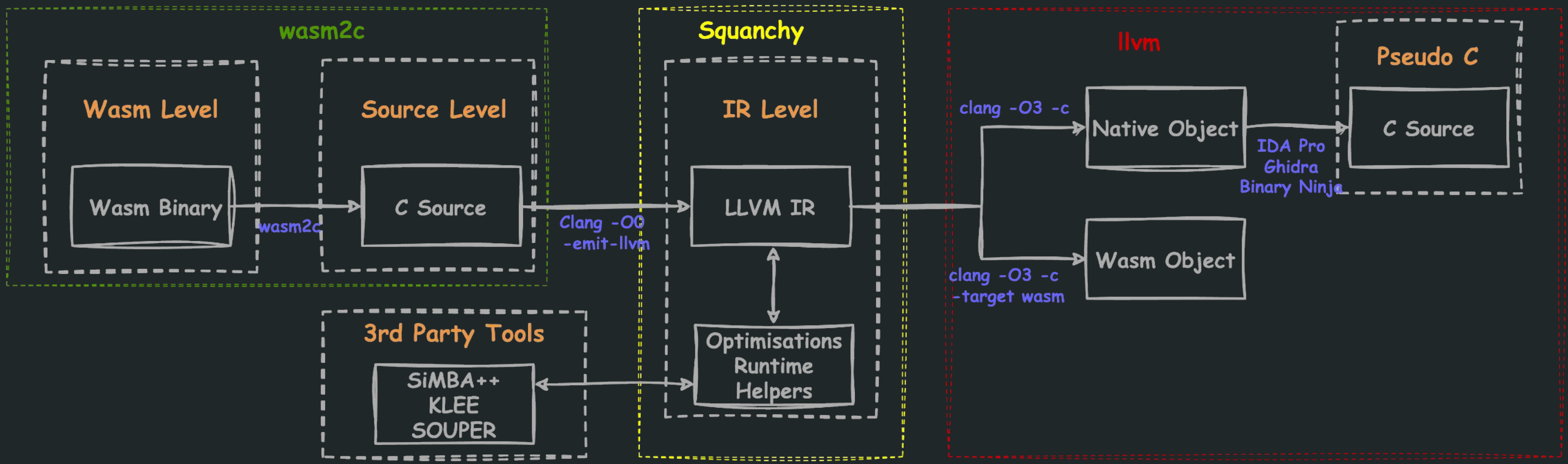
black hat ASIA 2025

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!

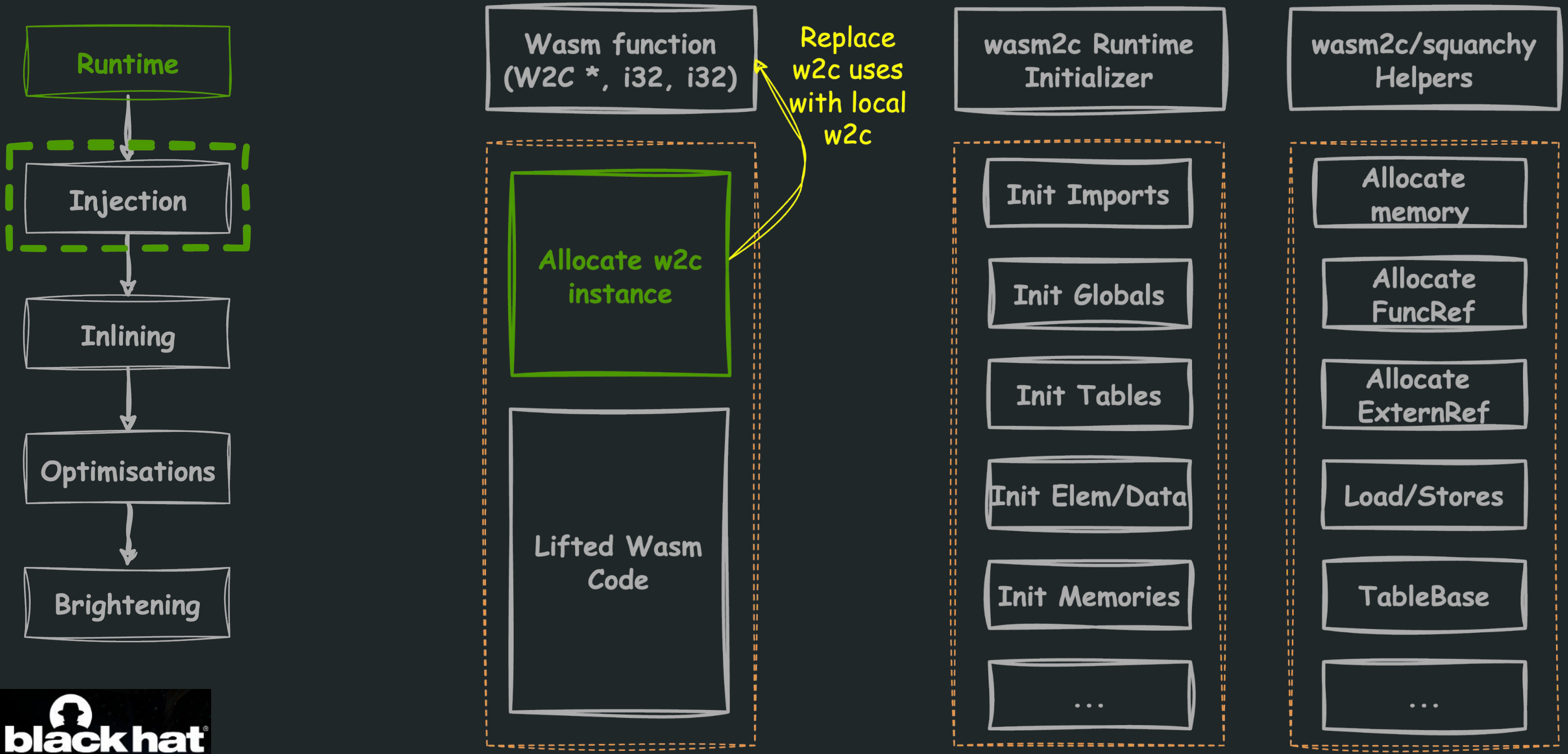


Wasm Code Lifting: Squanchy

- 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

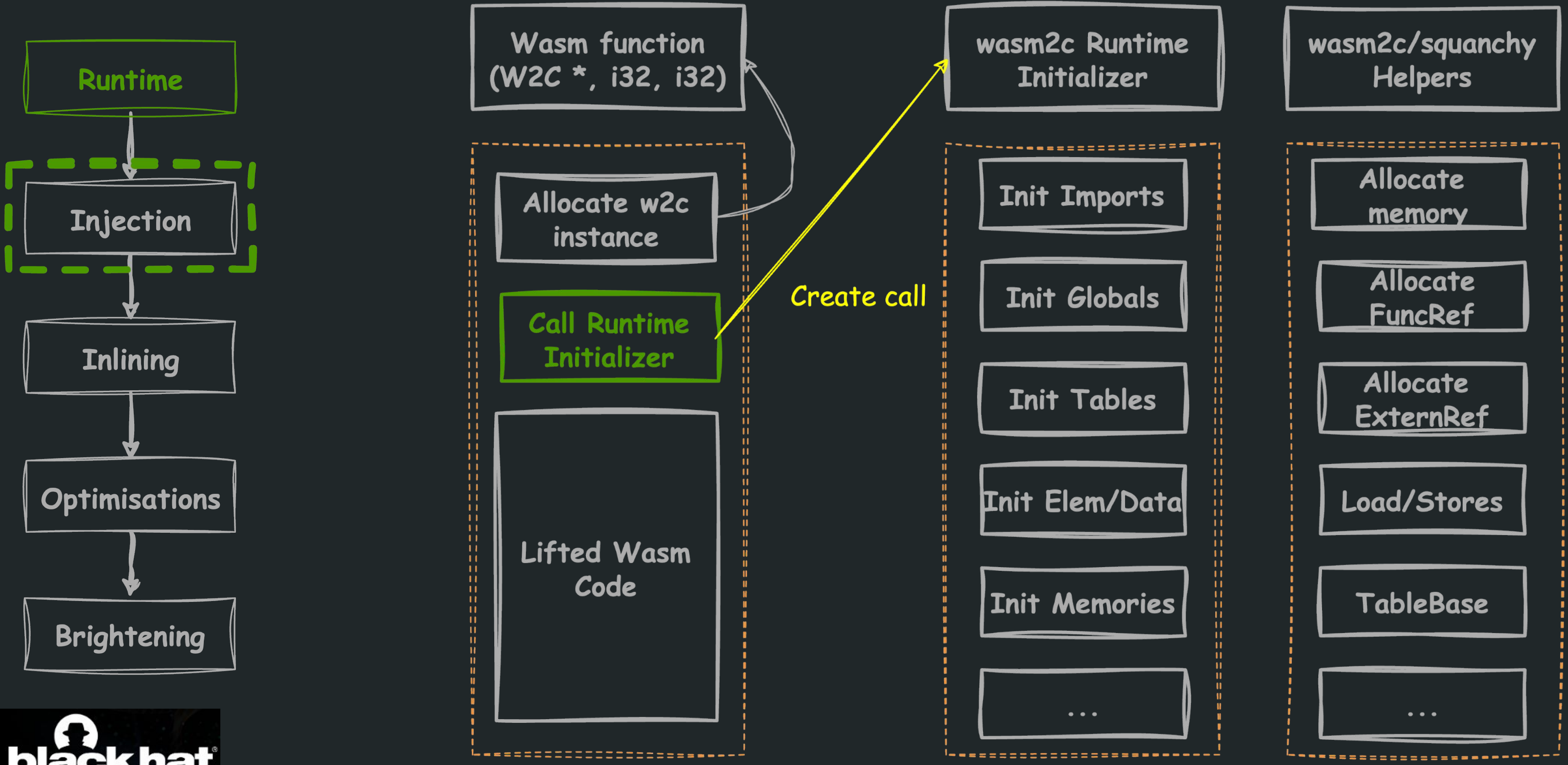
Wasm Code Lifting: Squanchy - *Runtime Modeling*

Wasm Code Lifting: Squanchy - Runtime Injection



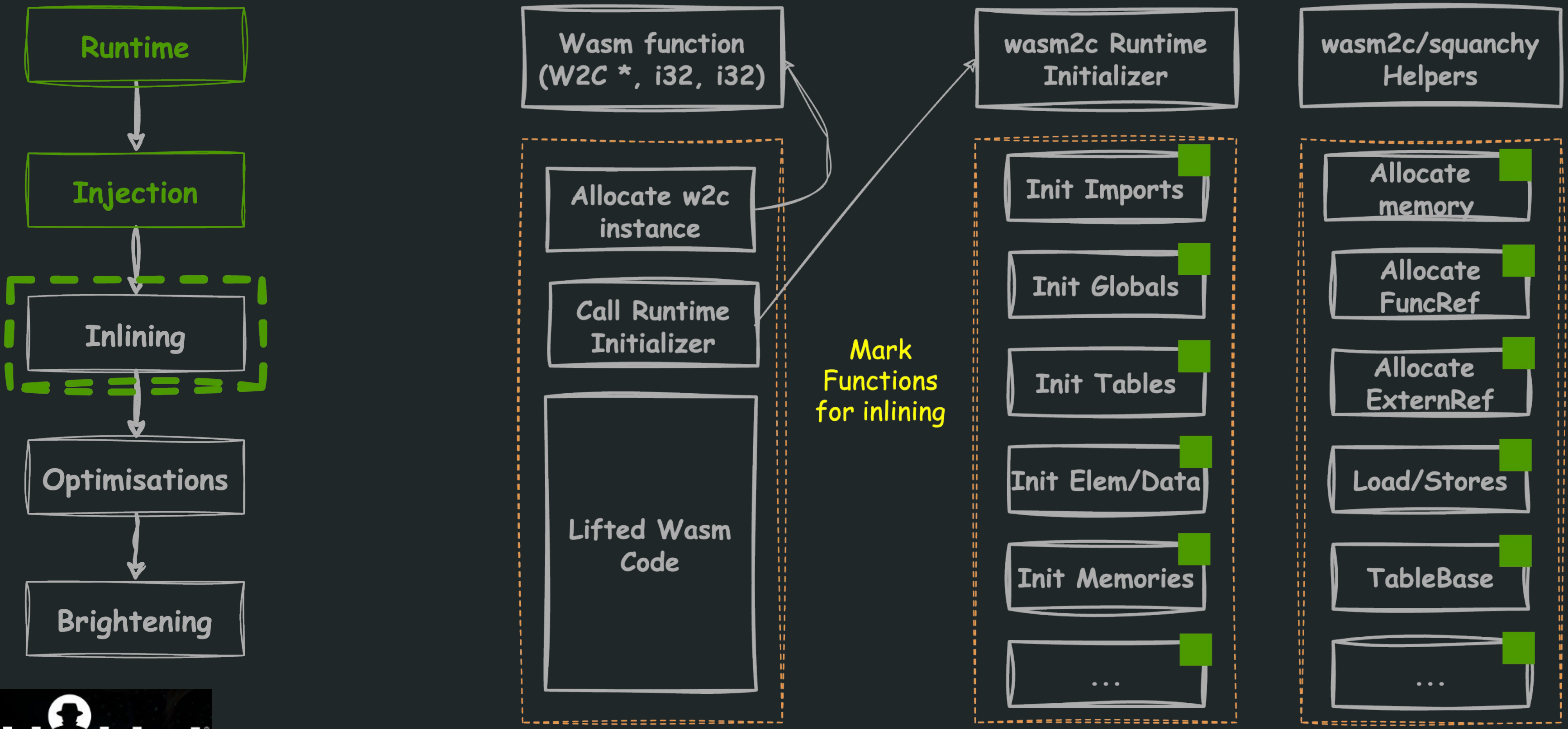
black hat ASIA 2025

Wasm Code Lifting: Squanchy - Runtime Injection



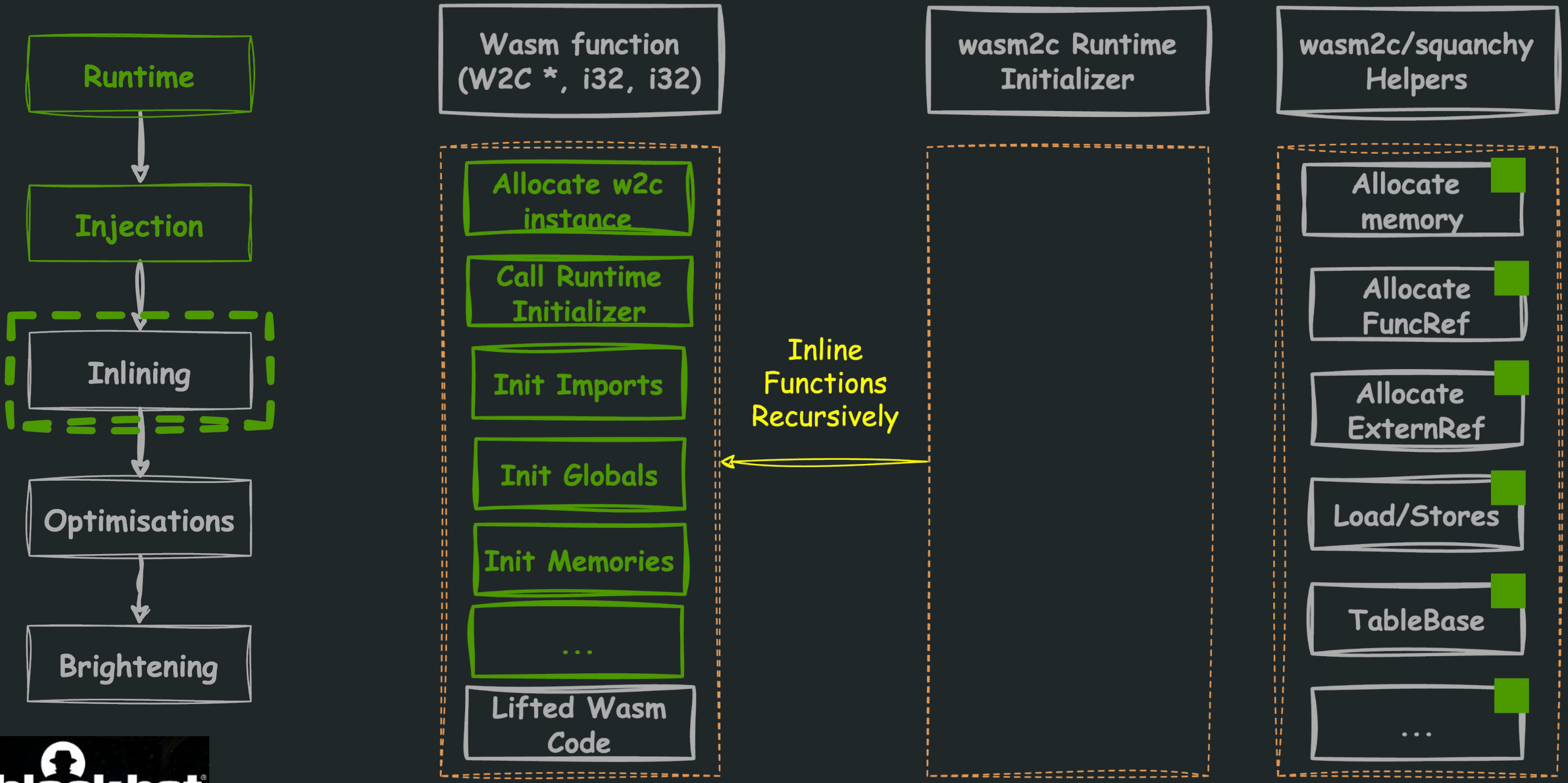
black hat ASIA 2025

Wasm Code Lifting: Squanchy - Inlining



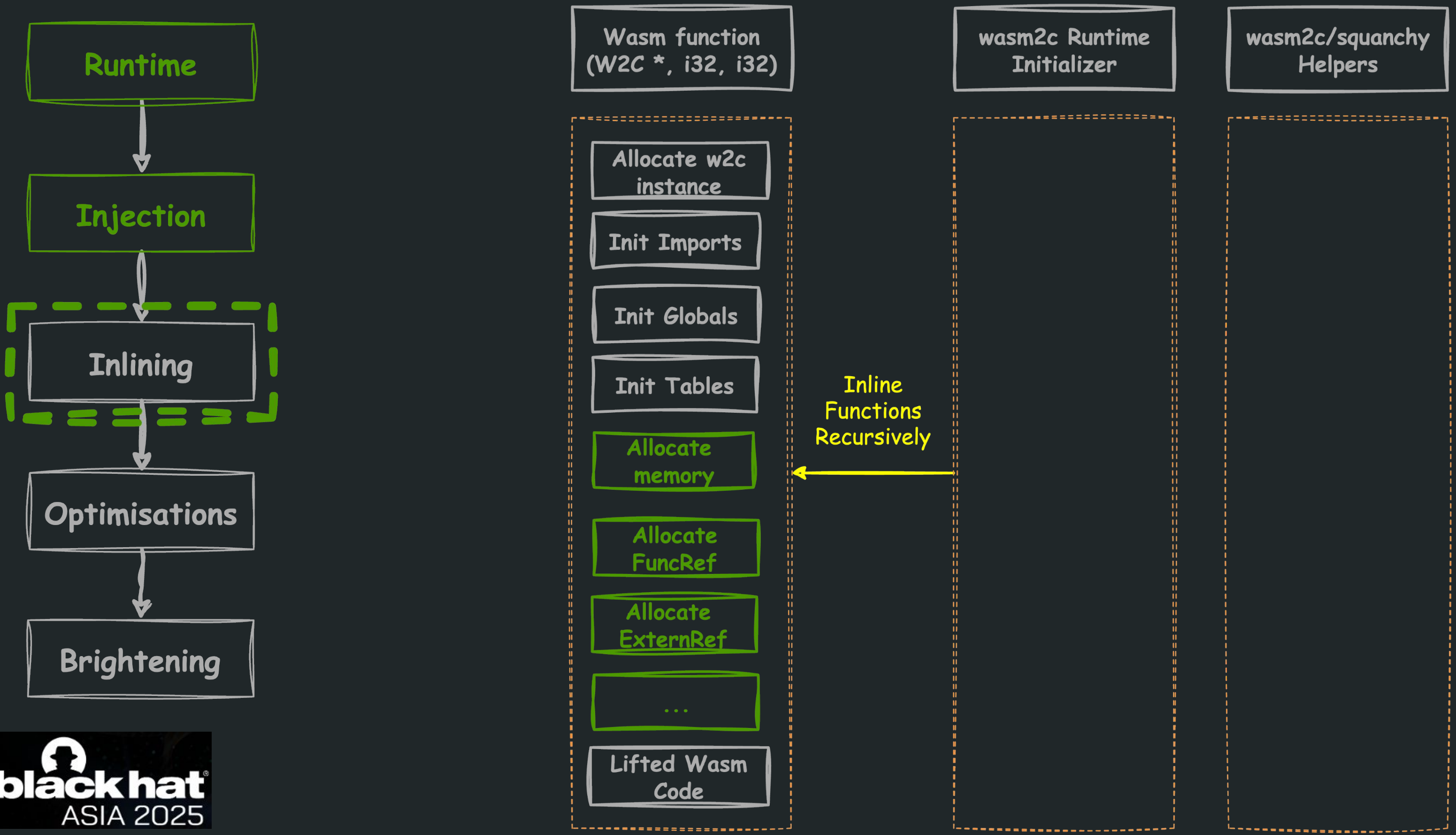
black hat ASIA 2025

Wasm Code Lifting: Squanchy - Inlining

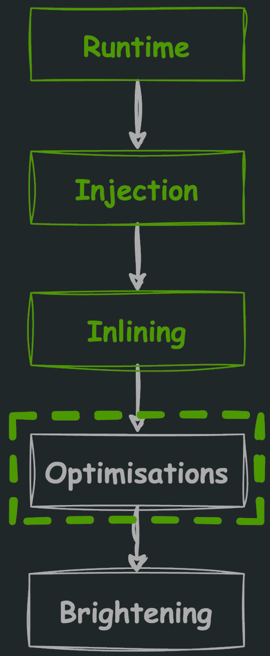


black hat ASIA 2025

Wasm Code Lifting: Squanchy - Runtime Injection

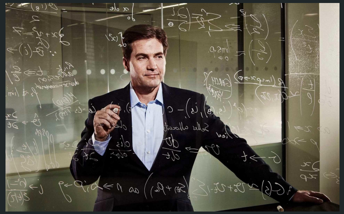


Wasm Code Lifting: Squanchy - Optimisation

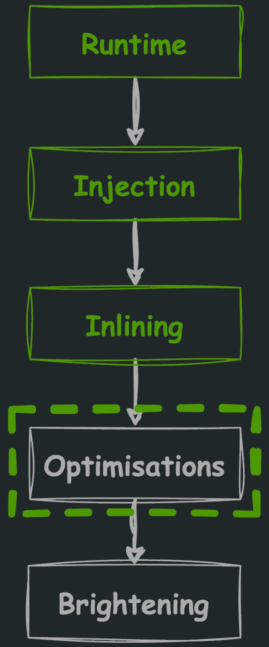


Apply LLVM O3 pipeline and preserve Control Flow Graph

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



Wasm Code Lifting: Squanchy - Optimizations



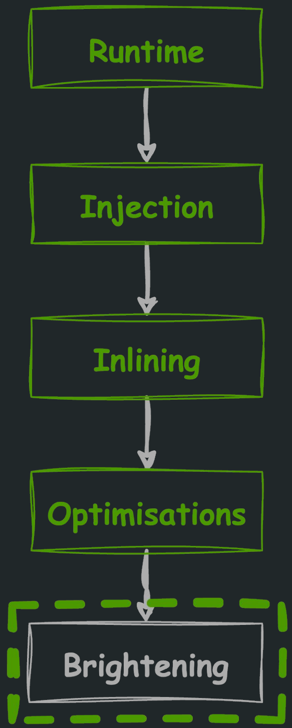
Override LLVM Thresholds

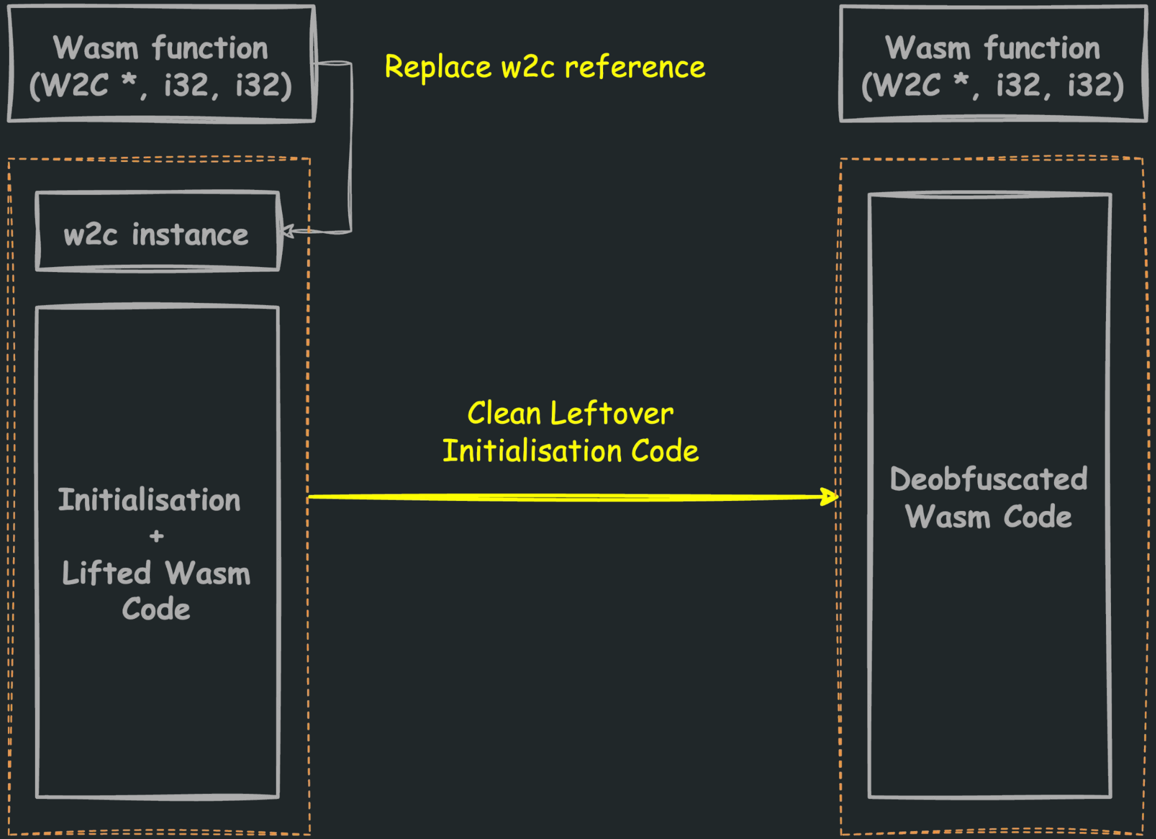
```
// DSE
-memdep-block-scan-limit=1000000
-dse-memoryssa-walklimit=1000000
-available-load-scan-limit=1000000
-dse-memoryssa-scanlimit=1000000

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

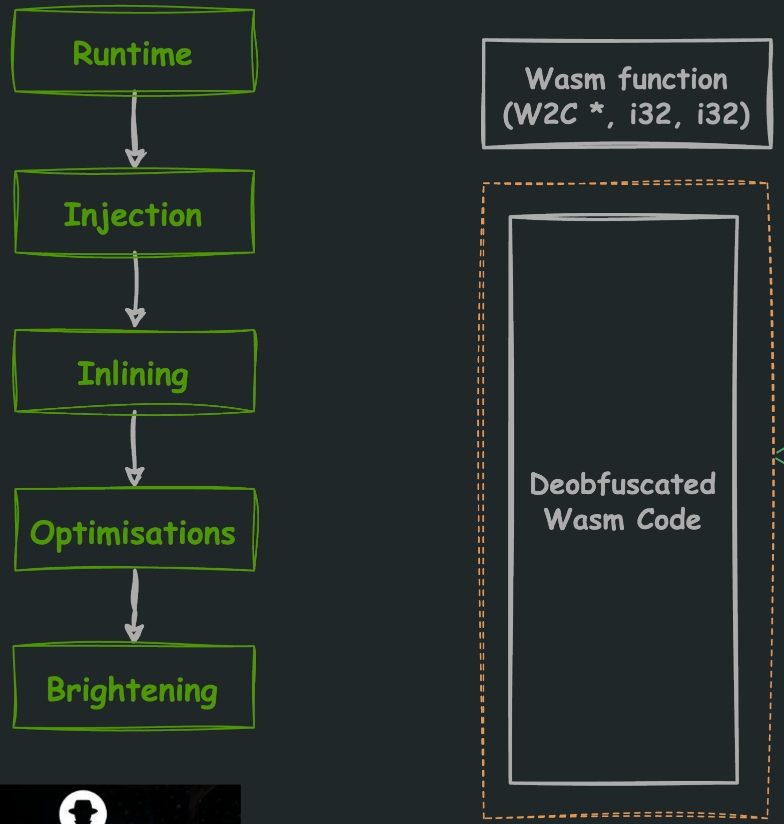
black hat ASIA 2025

Wasm Code Lifting: Squanchy - Brightening





Wasm Code Lifting: Squanchy - Recompilation



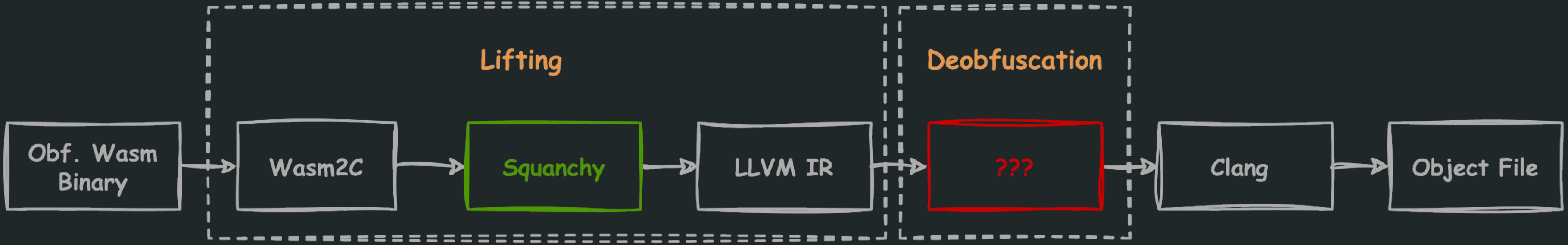
LLVM IR

```
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
```

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;
}
```

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) ^
                                | uVar10 & 0xd74ea659) | (param1 | uVar11) ^ 0xffffffff;
  uVar1                         & 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);
}
```

Unsigned Integer (compiler-specific size)
Length: 4

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

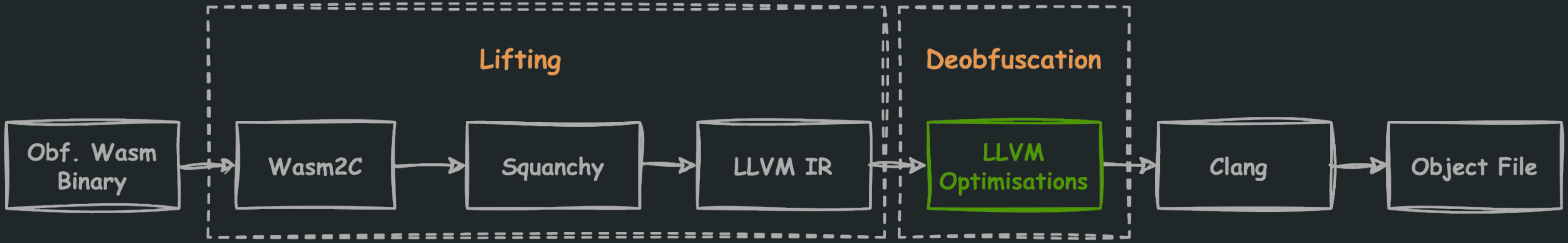
2

3

4

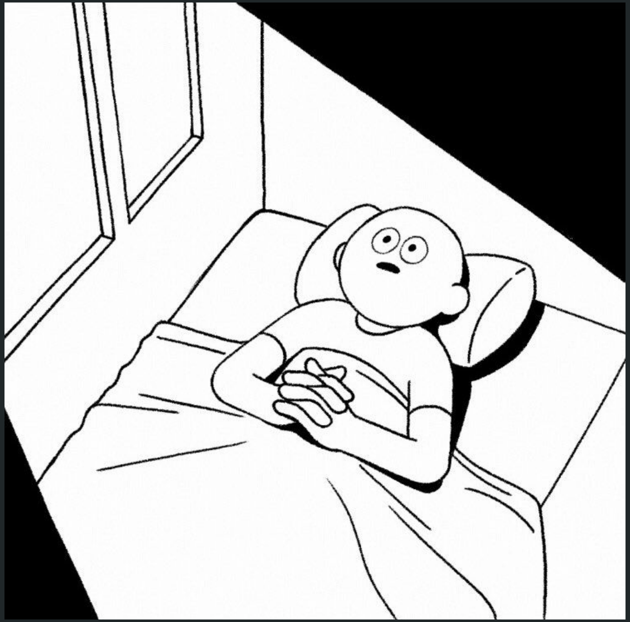
Original Expression Recovered

Deobfuscation: Progress...



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
  - . . .



Beyond LLVM: Solving MBAs

$$(x \oplus y) + 2 \times (x \land y) = x + y$$

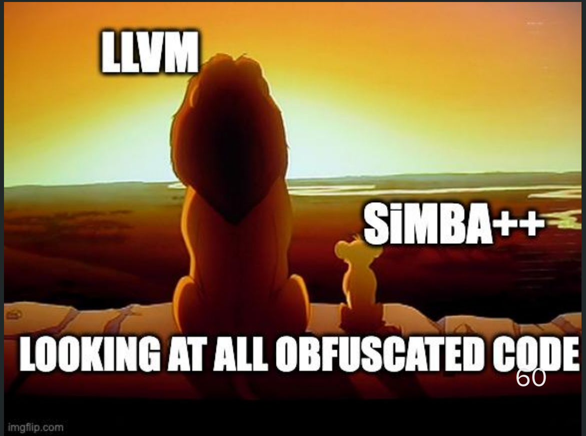
- Mixed Boolean Arithmetic (MBA) expressions
  - Expressions mixing arithmetic operators (+,-,x) with boolean operators (¬, ⊕, ∧, ∨)
  - Difficult to analyze - no general rules for interaction b/w operators (no distributivity, no associativity etc.)
  - With complex MBAs, SMT solvers may not able to 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



Deobfuscation: LLVM Opt + SiMBA + GAMBA

INPUT

```
if (iVar1 == 0) {
  uVar10 = param1 & 0xc9e645ce | (param1 ^ 0xffffffff) & 0x3619ba31;
  uVar11 = ((uVar10 ^ 0x3619ba31) & 0x5c2ea1f5 | (uVar10 ^ 0xc9e645ce) & 0xa3d15e0a) ^ 0xf1571e9d
           | uVar10 ^ 0x3619ba31;
  uVar12 = ((param1 ^ 0xffffffff) & 0xad79bf68 | param1 & 0x52864097) ^ 0xffffffff |
           param1 ^ 0xffffffff;
  uVar4 = uVar11 | uVar12;
  uVar11 = (uVar12 ^ 0xffffffff) & uVar11 | (uVar11 ^ 0xffffffff) & uVar12;
  uVar12 = uVar11 ^ 0xffffffff;
  uVar11 = (uVar4 & 0xaea378c3 | (uVar4 ^ 0xffffffff) & 0x515c873c) ^
           (uVar12 & 0xaea378c3 | uVar11 & 0x515c873c) | (uVar4 | uVar12) ^ 0xffffffff;
  uVar12 = ((uVar11 ^ 0xffffffff) & 0xcb73214a | uVar11 & 0x348cdeb5) ^ 0xcb73214a | 0x604af11c;
  uVar11 = uVar11 ^ 0xffffffff | 0x9fb50ee3;
  uVar11 = (uVar12 & 0x5835bf98 | (uVar12 ^ 0xffffffff) & 0xa7ca4067) ^
           (uVar11 & 0x5835bf98 | (uVar11 ^ 0xffffffff) & 0xa7ca4067) |
           (uVar12 | uVar11) ^ 0xffffffff;
  uVar11 = (uVar11 ^ 0xffffffff) & 0x88666134 | uVar11 & 0x77999ecb;
  uVar12 = uVar10 ^ 0x3619ba31 | 0xbaaad0bf;
  uVar10 = (uVar10 ^ 0x3619ba31) & 0x565b27a0 | (uVar10 ^ 0xc9e645ce) & 0xa9a4d85f;
  uVar4 = uVar10 ^ 0xecf1f71f;
  uVar12 = (uVar12 & 0xc4fa1585 | (uVar12 ^ 0xffffffff) & 0x3b05ea7a) ^
           (uVar4 & 0xc4fa1585 | (uVar10 ^ 0x130e08e0) & 0x3b05ea7a) |
           (uVar12 | uVar4) ^ 0xffffffff;
  uVar10 = (((uVar12 ^ 0xffffffff) & 0xbe3c86ad | uVar12 & 0x41c37952) ^ 0xbe3c86ad | 0xe6842217)
           ^ 0xffffffff;
  uVar12 = (uVar12 ^ 0x197bdde8) & uVar12;
  uVar10 = uVar10 & uVar12 | uVar10 ^ uVar12;
  uVar10 = (uVar10 ^ 0xffffffff) & 0x9d11e123 | uVar10 & 0x62ee1edc;
  uVar10 = (uVar10 ^ 0x846a3ccb) & 0x61675078 | (uVar10 ^ 0x7b95c334) & 0x9e98af87;
  uVar10 = (uVar10 ^ 0x61675078) & 0xa63617bd | (uVar10 ^ 0x9e98af87) & 0x59c9e842;
  uVar12 = (uVar11 ^ uVar10 ^ 0xa63617bd) & uVar11;
  uVar10 = ((uVar11 ^ 0xffffffff) & 0xbcc6bdd4 | uVar11 & 0x4339422b) ^
           ((uVar10 ^ 0xa63617bd) & 0xbcc6bdd4 | (uVar10 ^ 0x59c9e842) & 0x4339422b);
  uVar11 = uVar12 ^ 0xffffffff;
  uVar4 = uVar10 ^ 0xffffffff;
  uVar5 = (param1 ^ 0xffffffff) & 0xaf91567b | param1 & 0x506ea984;
  uVar6 = uVar5 ^ 0xaf91567b;
  uVar5 = (uVar6 & 0xe30d84a7 | (uVar5 ^ 0x506ea984) & 0x1cf27b58) ^ 0xe30d84a5 |
          (uVar6 | 0xfffffffd) ^ 0xffffffff;
  uVar6 = (((param1 ^ 0xffffffff) & 0x7ce0ffb1 | param1 & 0x831f004e) ^ 0xea109553) & 0x96f06ae2;
  uVar7 = (param1 ^ 0xffffffff | 0x96f06ae2) ^ 0xffffffff;
  uVar8 = uVar6 ^ uVar7;
  uVar6 = (uVar8 ^ 0xffffffff) & 0x690f951d | uVar6 & uVar7 | uVar8 & 0x96f06ae2;
  uVar6 = uVar6 & 0xba8c5c70 | (uVar6 ^ 0xffffffff) & 0x4573a38f;
  uVar6 = (uVar6 ^ 0x4573a38f) & (uVar6 ^ 0xba8c5c72);
  uVar7 = (uVar5 ^ 0xcb0a5819) & uVar5;
  uVar8 = (uVar5 | 0x34f5a7e6) ^ 0xffffffff;
  uVar9 = (uVar6 ^ 0x34f5a7e6) & (uVar6 ^ 0xffffffff);
  uVar2 = (uVar6 ^ 0x34f5a7e6) & uVar6;
  uVar7 = uVar7 & uVar8 | uVar7 ^ uVar8;
  uVar8 = uVar9 ^ 0xffffffff;
  uVar3 = uVar2 ^ 0xffffffff;
  uVar8 = (uVar8 & 0xd3d541ce | uVar9 & 0x2c2abe31) ^ (uVar3 & 0xd3d541ce | uVar2 & 0x2c2abe31) |
          (uVar8 | uVar3) ^ 0xffffffff;
  uVar7 = ((uVar7 ^ 0xffffffff) & 0xefd84b11 | uVar7 & 0x1027b4ee) ^
          ((uVar8 ^ 0xffffffff) & 0xefd84b11 | uVar8 & 0x1027b4ee);
  uVar5 = ((uVar5 ^ 0xffffffff) & 0xffb73ee8 | uVar5 & 0x48c117) ^
          (uVar6 & 0xffb73ee8 | (uVar6 ^ 0xffffffff) & 0x48c117) |
          (uVar5 ^ 0xffffffff | uVar6) ^ 0xffffffff;
  uVar5 = (uVar5 ^ 0xffffffff) & 0xa95ee2be | uVar5 & 0x56a11d41;
  uVar6 = uVar5 ^ 0xa95ee2be;
  uVar8 = uVar7 ^ 0xffffffff;
  local_10 = ((uVar11 & 0x1e98326 | uVar12 & 0xfe167cd9) ^
              (uVar4 & 0x1e98326 | uVar10 & 0xfe167cd9) | (uVar11 | uVar4) ^ 0xffffffff) *
             ((uVar8 & 0xc8a77567 | uVar7 & 0x37588a98) ^
              (uVar6 & 0xc8a77567 | (uVar5 ^ 0x56a11d41) & 0x37588a98) |
             (uVar8 | uVar6) ^ 0xffffffff);
}
```

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);
}
```

SiMBA

```
if (uVar2 == 0) {
  uVar2 = (param_2 & 0xa81b62f7 | 0x17000408) ^ param_2 & 0x57e49d08;
  *piVar1 = ((uVar2 ^ 0xadaad4b7) & (param_2 & 0xbaaad0bf ^ 0xffffffff) |
            (uVar2 ^ 0x12000008) & param_2 & 0xbaaad0bf) *
            (param_2 & 0xfffffffd ^ (param_2 & 2 | 0x4fa5c831) ^ 0x4fa5c833);
  return;
}
```

Gamba

```
if (uVar2 == 0) {
  *piVar1 = ((param_2 & 0x77b39989 | 0x88006252) ^ (param_2 ^ 0xed32f2d0) & (param_2 ^ 0x9a816b59)
            ) * (param_2 | 0xbaaad0bf);
  return;
}
```

Deobfuscation: LLVM Opt + SiMBA + GAMBA

Polaris
Instruction substitution (loop 3)

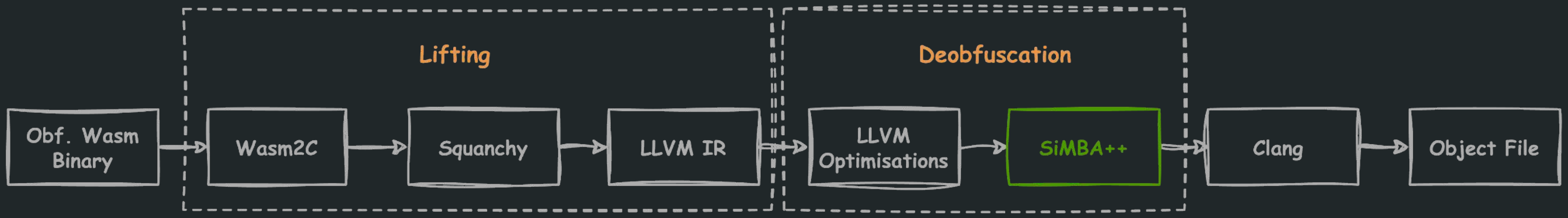
```c
uVar1 = param1 & 3;
if (uVar1 == 0) {
  uVar1 = ((param1 ^ 0xffffffff) & 0x19495cff | param1 & 0xe6b6a300) ^ 0x5c1c73bf;
  uVar1 = (uVar1 ^ param1 ^ 0xffffffff) & uVar1;
  uVar5 = (param1 ^ 0x625558ec) & (param1 ^ 0xffffffff);
  uVar2 = (param1 ^ 0x625558ec) & param1;
  uVar5 = uVar5 & uVar2 | uVar5 ^ 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) {
  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) {
  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 {
  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

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

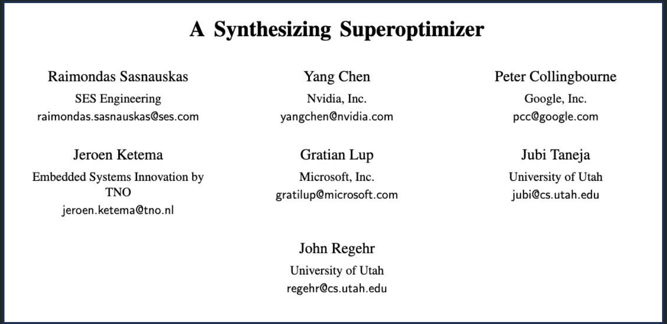
Completely Recovered

Deobfuscation: Progress...



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
- Can run as an LLVM optimization pass
- Synthesise optimisations
  - Counterexample guided inductive synthesis (CEGIS)
    - Multiple RHS generated, cheapest among them is chosen.
  - Dataflow
- Can resolve opaque predicates
- **Good results with control flow obfuscation**


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

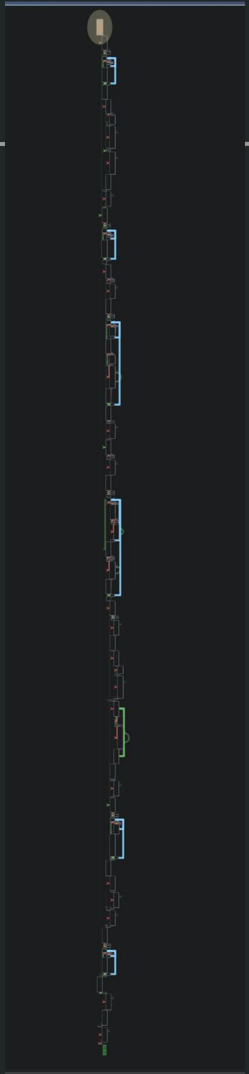
John Regehr
University of Utah
regehr@cs.utah.edu

Deobfuscation: SOUPER

Hikari
Bogus Control Flow (loop 2)

Opaque Predicate

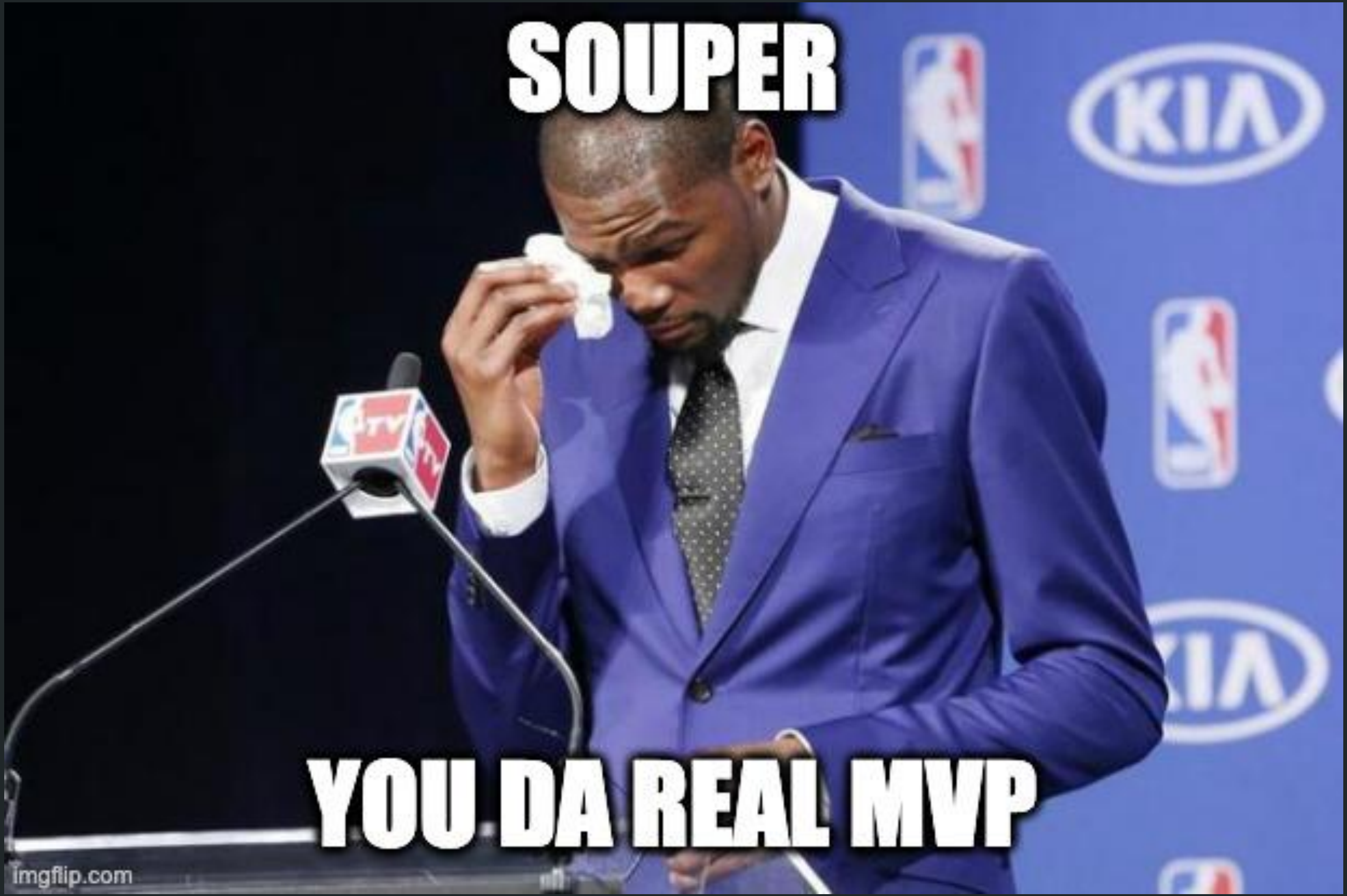
```
if (uVar2 == 0) {
  if ((((uRam00010ae8 | uRam00010aec) ^ 0x3c2e5570) & 0x97ff2bd7) + 0x64293ba9 < 0xe0465c21) {
    bVar1 = true;
  }
  else {
    bVar1 = false;
  }
  while( true ) {
    while (bVar1) {
      bVar1 = false;
    }
    if (0x5fd76e94 < ((iRam00010af0 + iRam00010af4 ^ 0x41005e8aU) + 0x63d028ff) * 0x65edfa51)
    break;
    bVar1 = true;
  }
  if ((iRam00010af8 * iRam00010afc + 0xd9ef92c7U | 0xba1e4315) + 0xce83d3a0 < 0x8bb488b1) {
    do {
    } while (((uRam00010c30 ^ uRam00010c34) + 0x6f44ee27 & 0x7ca03c77) * 0x5ed0b58a == -0x29120a04
            );
  }
  do {
    local_c = (param1 | 0xbaaad0bf) * (param1 ^ 2);
  } while (((uRam00010b00 / uRam00010b04 | 0xbabf5164) * -0x32ee4c95 & 0x67c7c119) < 0x20a96022);
  do {
  } while ((uRam00010b10 / uRam00010b14 + 0xc8de4516) / 0x936b17aa == 0xa9f0a4ac);
}
```



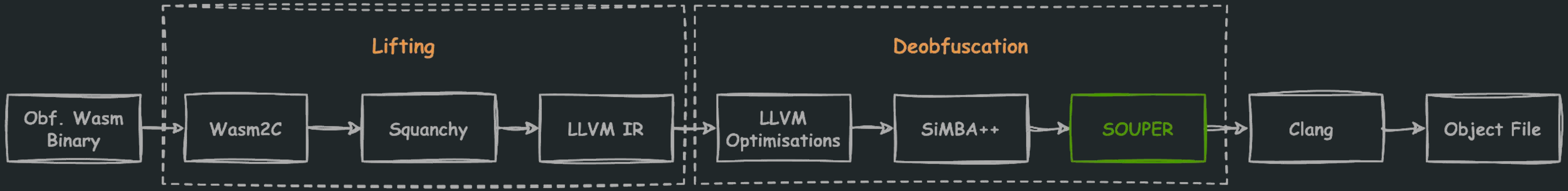
Deobfuscation: SOUPER

SOUPER

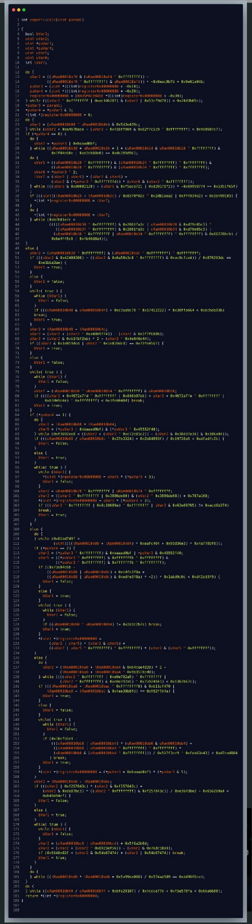
```
uVar1 = param_2 & 3;
iVar3 = (param_2 ^ 0xbaaad0bf) * (param_2 | 4);
if (uVar1 != 2) {
  iVar3 = (param_2 & 5) * (param_2 + 0xbaaad0bf);
}
iVar2 = (param_2 | 0xbaaad0bf) * (param_2 ^ 2);
if (uVar1 != 0) {
  iVar2 = (param_2 & 0xbaaad0bf) * (param_2 + 3);
}
if (uVar1 < 2) {
  iVar3 = iVar2;
}
return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


SOUPER
YOU DA REAL MVP
imgflip.com

Deobfuscation: Progress…



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



Long complex obfuscated code

```
uVar1 = param_2 & 3;
if (uVar1 < 2) {
  iVar2 = (param_2 | 0xbaaad0bf) * (param_2 + 2);
  if (uVar1 != 0) {
    iVar2 = (param_2 + 3) * (param_2 & 0xbaaad0bf);
  }
  return iVar2;
}
if (uVar1 == 3) {
  return (param_2 & 5) * (param_2 + 0xbaaad0bf);
}
return (param_2 & 0xbaaad0b8 ^ 0xbaaad0b9) * (param_2 & 0xbaaad0bf) +
       (param_2 & 0x45552f44 ^ 4) * (param_2 | 0xbaaad0bf);
}
```

Real World Application

WebAssembly Malwares

WebAssembly Is Abused by eCriminals to Hide Malware

October 25, 2021 | Mihai Maganu | Engineering & Tech

The dark side of WebAssembly

Aishwarya Lonkar & Siddhesh Chandrayan

Symantec, India

Copyright © 2018 Virus Bulletin

Recent Study Estimates That 50% of Websites Using WebAssembly Apply It for Malicious Purposes

2019

- 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.

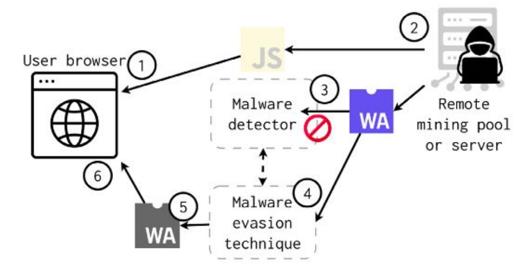


Fig. 1. WebAssembly evasion in practice. The user visits a webpage containing cryptojacking malware that uses network resources to operate. A malware detector blocks identified malicious WebAssembly binaries. The attacker, using a malware oracle, creates a WebAssembly cryptojacking malware variant that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

Malware Diversification: *wasm-mutate*

- Carbera-Arteaga et. al. demonstrate use of *wasm-mutate* to evade detection

- *wasm-mutate* transforms binary into a variant binary program that preserves the original functionality.

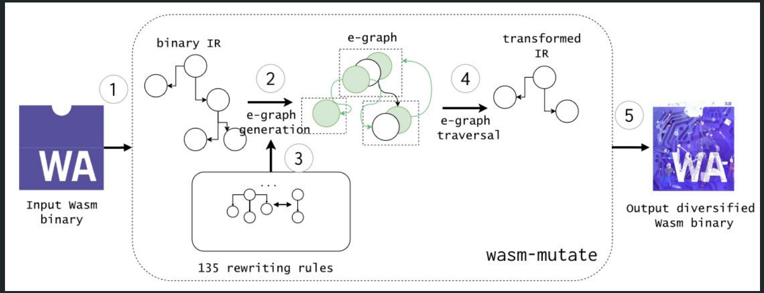
- 3 kind of transformations
  - Peephole
    - ~135 rewrite rules.
  - Module structure transformation
    - Add new type, new function, new export etc.
  - Control flow graph
    - Loop unrolling, swap conditional branches.

- *wasm-mutate* output needs to verified with *wasm-validate*
  - Some transformation break the WASM file


WebAssembly diversification for malware evasion

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

KTH Royal Institute of Technology, Stockholm, Sweden



DEMO!!

Use Case: wasm-mutate

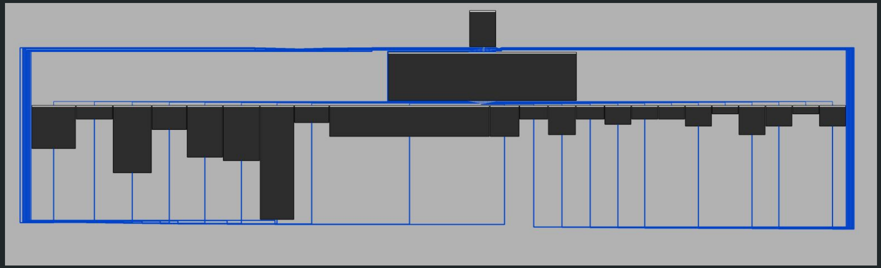
- wasm-mutate
  - 3000 (real) iterations are applied
  - *100%* of mutations are removed
  - Code is normalized and matches 100% the original code!
    - Our approach fully recovers the function (and optimizes it!)

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

Use Case: hCaptcha

- hCaptcha uses obfuscated Wasm
- Small and medium size obfuscated functions can be simplified in <1-2min.

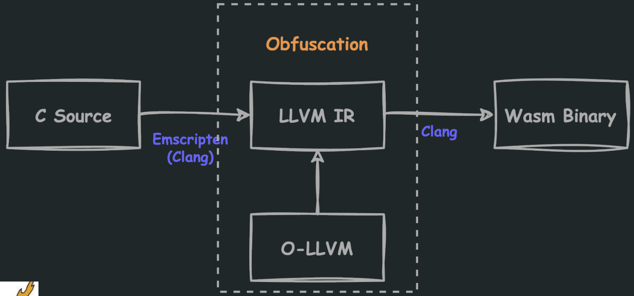


Unflattens control flow, simplifies and inlines functions.


BREATHE
JUST BREATHE

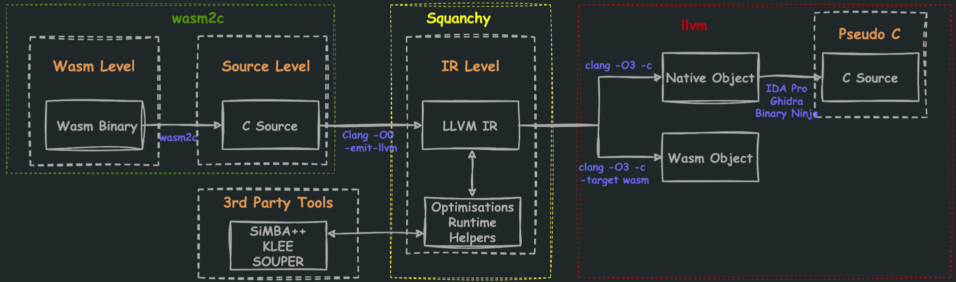
Conclusion

- Wasm obfuscation
  - LLVM IR based tools - O-LLVM, Polaris



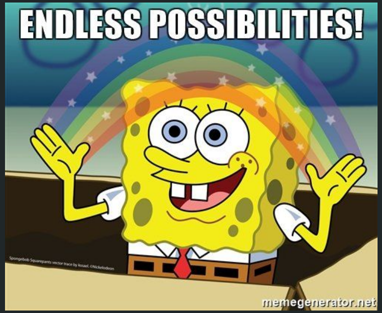
- Squanchy: Lifting Wasm to LLVM IR
  - Wasm2c + Squanchy works great.

- Deobfuscation using LLVM
  - LLVM + SiMBA + GAMBA + SOUPER + …



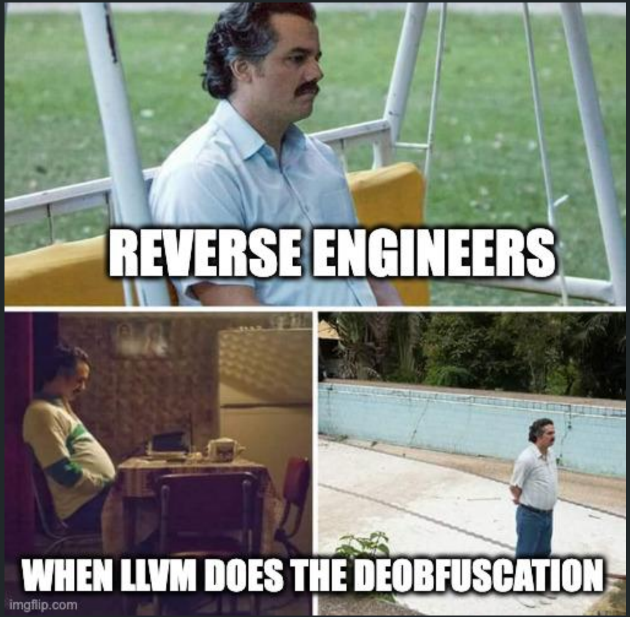
Conclusion

- Real World Application
  - Malware Normalisation
    - Wasm-mutate output can be simplified
    - Cryptonight malware simplified
  - Deobfuscating hCaptcha binary



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

Thank You!!



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