GCC Is The New Pincc

Typo Intended

Marion Marschalek | Security Researcher

Hello, this is me

Disclaimer

The opinions and positions expressed herein are mine only and do not represent the views of any current or previous employer, including Intel Corporation or its affiliates.

This presentation has no intention to advertise or devalue any current or future technology.

```
printf("Hello world!\n");
```

```
// Iterating through basic blocks and Gimple sequences
FOR_EACH_BB_FN(bb, cfun) {

    for (gsi = gsi_start_bb(bb); !gsi_end_p(gsi); gsi_next(&gsi)) {

        gimple *statement = gsi_stmt(gsi);

        // Picking up on the printf within our helloworld.c
        if (gimple_code(statement) == GIMPLE_CALL){

            // Getting the first argument of printf
            tree arg = gimple_call_arg(statement, 0);

            // Building the new string argument
            tree satan = build_string(strlen("Hail Satan!!\n")+1, "Hail Satan!!\n");
            tree type = build_array_type(
                build_type_variant(char_type_node, 1, 0),
                build_index_type(size_int(strlen("Hail Satan!!\n"))));
            TREE_TYPE(satan) = type;
            TREE_CONSTANT(satan) = 1;
            TREE_READONLY(satan) = 1;
            TREE_STATIC(satan) = 1;

            // Replacing the helloworld string argument
            TREE_OPERAND(TREE_OPERAND((arg), 0), 0) = satan;
            gimple_call_set_arg(statement, 0, arg);
        }
    }
```

.. goes hail satan ..


WHY?

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel

programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and

The GNU Compiler Collection

https://gcc.gnu.org/

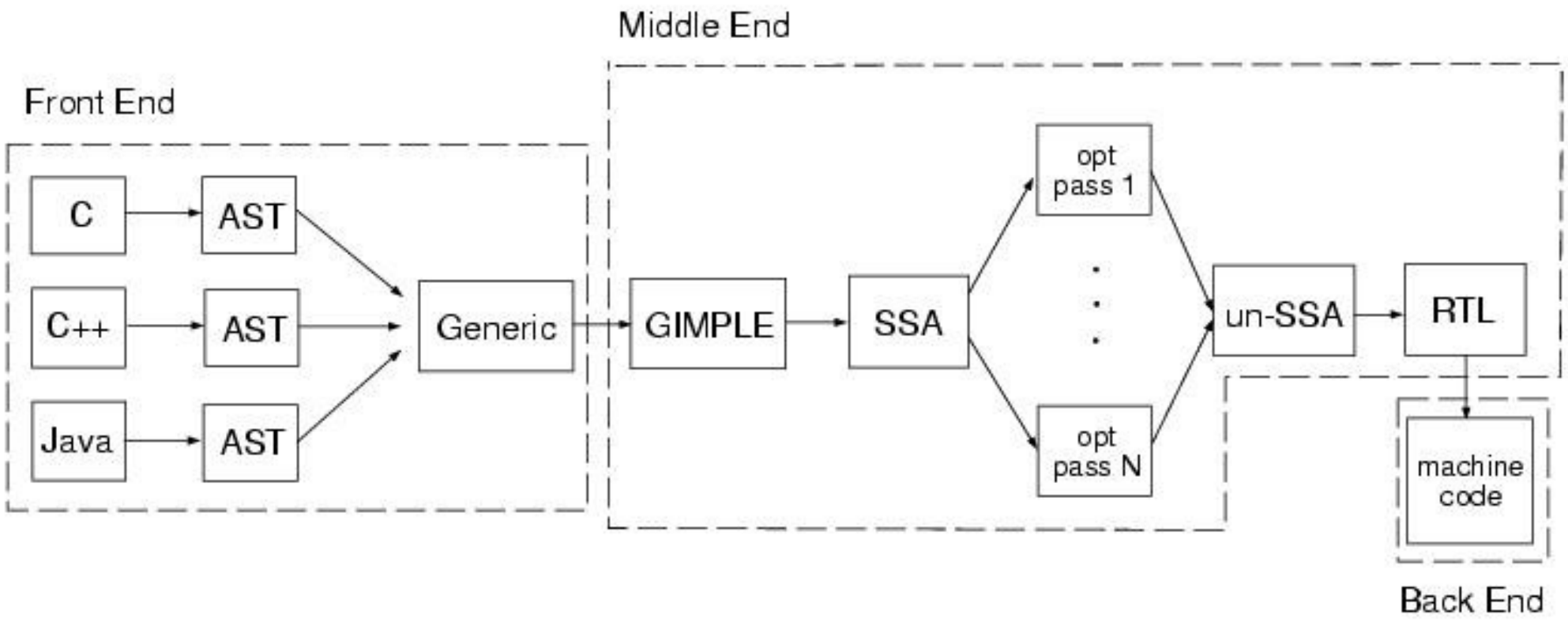
Has Front, Middle and Back End

Can compile code

But also:

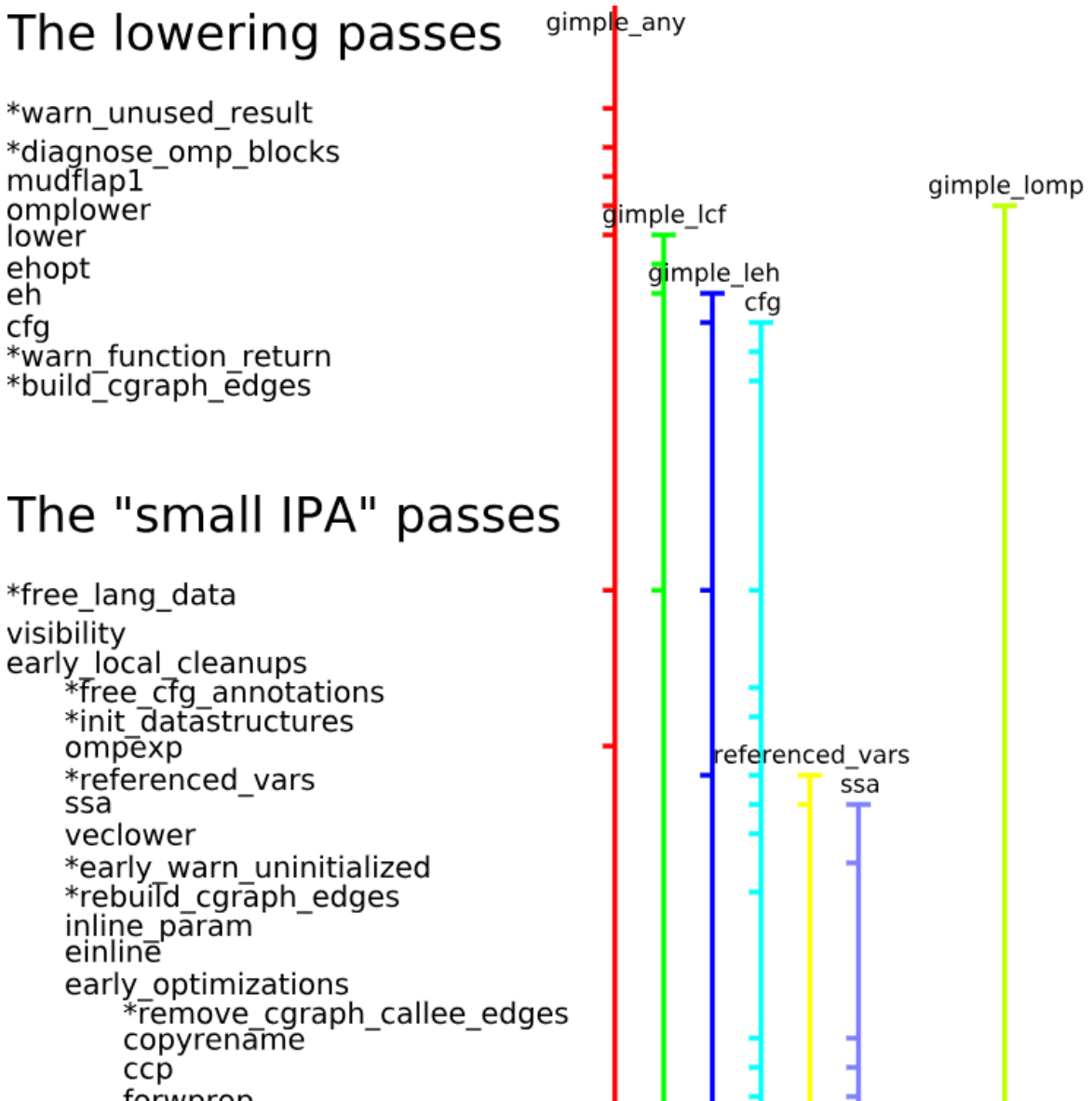
Exchange frontends/backends as in designing a new programming language or adding compiler support for an exotic CPU architecture, add optimization passes, perform static analysis in the compilation process, add compiler mitigations, search for optimization bugs, introduce “optimization bugs”, etc. etc.

Every presentation any researcher has ever done on GCC things starts with this picture.



I mean, almost?

GCC's Compiler Passes



GCC's compilation process is organized in passes

Neat explanatory graphic by David Malcolm

https://gcc-python-plugin.readthedocs.io/en/latest/tables-of-passes.html

The Debug Output

… is worth gold, and looks a bit like a “Matrix” screensaver when you scroll down fast

-fdump-passes

-fdump-tree-all, -fdump-ipa-all, -fdump-rtl-all

-fdump-tree-cfg-all

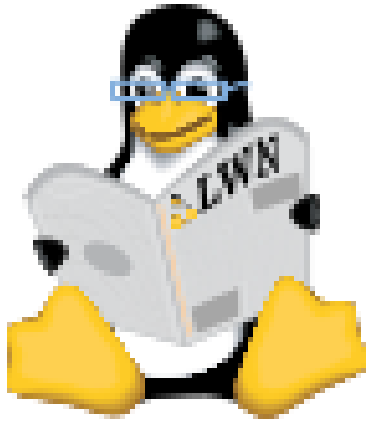
-fdump-rtl-MYAWESOMEPLUGIN

GCC Plugins

Since GCC 4.5 we can plug passes into the compilation process!

Benefits of plugins vs. modifying GCC itself?

- Plugins are shared objects, loaded by GCC as dedicated passes
- Maintained by pass manager
- Dependent on compiler version
- GCC plugin API defined in tree-pass.h



https://lwn.net/Articles/457543/

```
namespace {

const pass_data pass_data_MYAWESOMEPASS =
{
  RTL_PASS,       /* type */
  NAME,           /* name */
  OPTGROUP_NONE,  /* optinfo_flags */
  TV_NONE,        /* tv_id */
  PROP_rtl,       /* properties_required */
  0,              /* properties_provided */
  0,              /* properties_destroyed */
  0,              /* todo_flags_start */
  0,              /* todo_flags_finish */

};

class pass_MYAWESOMEPASS : public rtl_opt_pass
{
 public:
  pass_MYAWESOMEPASS(gcc::context *ctxt) : rtl_opt_pass(pass_data_MYAWESOMEPASS, ctxt)
  {}

  bool gate () { return true; }
  unsigned int execute (function *fun)  { return execute_MYAWESOMEPASS(); }
};

}

static rtl_opt_pass * make_pass_MYAWESOMEPASS(gcc::context *ctxt)
{
  return new pass_MYAWESOMEPASS(ctxt);
}


int plugin_init(struct plugin_name_args *plugin_info, struct plugin_gcc_version *version)
{
  struct register_pass_info pass_info;

  if (!plugin_default_version_check(version, &gcc_version))
    return FAILURE;

  pass_info.pass = make_pass_MYAWESOMEPASS(g);
  pass_info.pass->static_pass_number = 0;

  pass_info.reference_pass_name = "vartrack";
  pass_info.ref_pass_instance_number = 1;
  pass_info.pos_op = PASS_POS_INSERT_AFTER;

  register_callback(NAME, PLUGIN_PASS_MANAGER_SETUP, NULL, &pass_info);

  return SUCCESS;
}
```

Plugin types

GIMPLE

RTL

SIMPLE_IPA

IPA

LTO

Prior research that makes life a LOT easier

Emese Revfy https://github.com/ephox-gcc-plugins

Matt Davis https://github.com/enferex/

PaX team: RAP and more https://github.com/rrbranco/grsecurity-pax-history/tree/master/pax

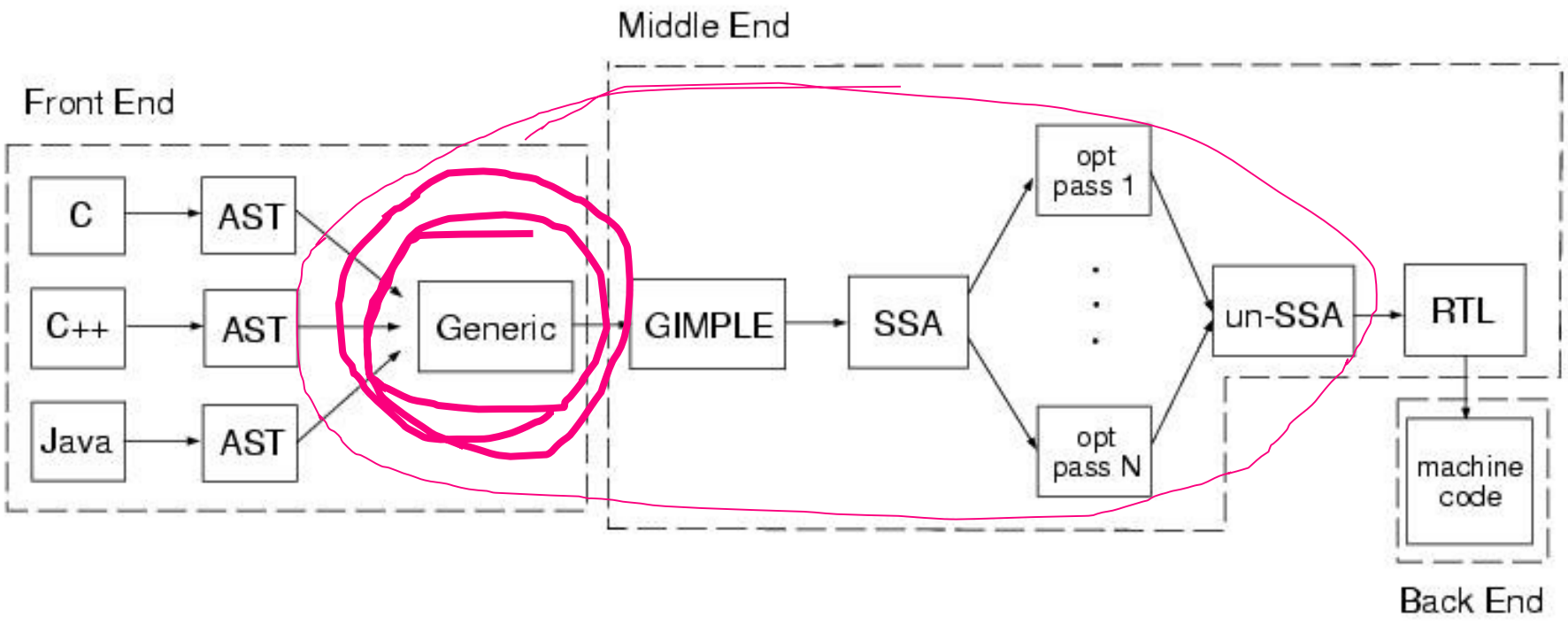
- H2HC 2012: https://pax.grsecurity.net/docs/PaXTeam-H2HC12-PaX-kernel-self-protection.pdf
  - PaX Untold Story (which includes the explanation of the first plugins)
- H2HC 2013: https://pax.grsecurity.net/docs/PaXTeam-H2HC13-PaX-gcc-plugins.pdf
  - PaX GCC Plugins
- H2HC 2015: https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf
  - RAP RIP ROP

KGuard https://github.com/pmoust/kguard

Roger Ferrer Ibanez https://github.com/rofirrim/gcc-plugins

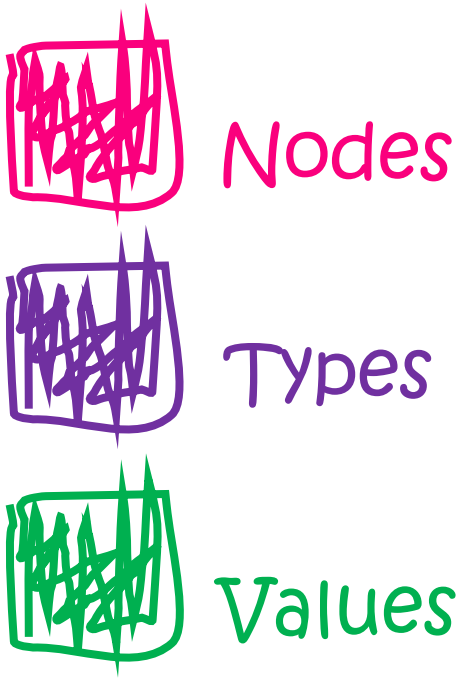
Stages of the compiler and what they mean for plugin writers

GENERIC or the mystic TREE



The mystic TREE

```
@3215   function_decl     name: @3217     type: @2980     scpe: @176
                          srcp: helloworld.c:3            link: extern
                          body: @3218

.............
@3217   identifier_node   strg: main      lngt: 4
@3218   statement_list    0    : @3222    1    : @3223
@3219   identifier_node   strg: __builtin_fork            lngt: 14
.............
@3222   bind_expr         type: @151      body: @3225
@3223   return_expr       type: @151      expr: @3226
@3224   function_decl     name: @3227     type: @3228     scpe: @176
                          srcp: <built-in>:0              chain: @3229
                          body: undefined                 link: extern
@3225   statement_list    0    : @3230    1    : @3231
@3226   modify_expr       type: @3        op 0: @3232     op 1: @2098
.............
                          link: extern
@3230   call_expr         type: @3        fn   : @3238    0    : @3239
@3231   return_expr       type: @151      expr: @3240
@3232   result_decl       type: @3        scpe: @3215     srcp: helloworld.c:3
                          note: artificial                size: @5
                          algn: 32

.............
@3238   addr_expr         type: @3243     op 0: @3159
@3239   nop_expr          type: @1932     op 0: @3244
@3240   modify_expr       type: @3        op 0: @3232     op 1: @2098
.............
@3243   pointer_type      size: @22       algn: 64        ptd : @3247
@3244   addr_expr         type: @3248     op 0: @3249
@3245   identifier_node   strg: __builtin_frob_return_addr
                          lngt: 26

.............
@3248   pointer_type      size: @22       algn: 64        ptd : @3255
@3249   string_cst        type: @3255     strg: Hello world!
  lngt: 14
```



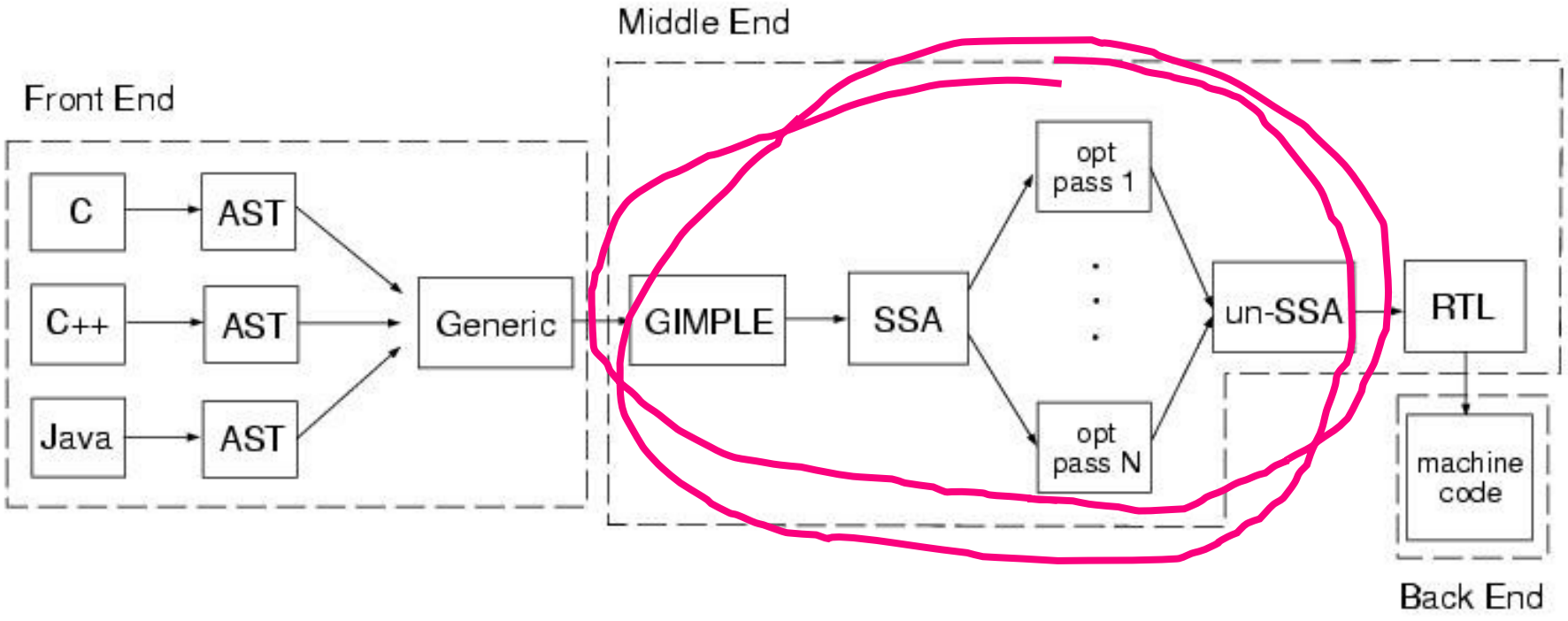
GENERIC

- Language-independent way of representing an entire function as trees
- Interface between parser and optimizers
- Superset of Gimple: imagine a language with a tree structure, similar to LISP
- Defined in gcc/tree.def

Important concepts:

Tree types and DECLs, expressions and statements

GIMPLE – A tree based representation



GIMPLE

The three address code

Target- and language independent optimization

```
# Calculate one solution to the [[quadratic equation]].
x = (-b + sqrt(b^2 - 4*a*c)) / (2*a)
```

```
t1 := b * b
t2 := 4 * a
t3 := t2 * c
t4 := t1 - t3
t5 := sqrt(t4)
t6 := 0 - b
t7 := t5 + t6
t8 := 2 * a
t9 := t7 / t8
x  := t9
```

https://en.wikipedia.org/wiki/Three-address_code

```c
int sub(void) {
    int x = 10;
    int y = 2;
    return x-y;
}

int add(void) {
    int a = 10;
    int b = 20;
    return a+b;
}

int main(void) {
    int d = add();
    int f = sub();

    int g = (d * 100 + 15) - (f * 10 - 50);
    return 0;
}
```

C

GIMPLE

```gimple
sub ()
{
  int D.1809;
  int x;
  int y;

  x = 10;
  y = 2;
  D.1809 = x - y;
  return D.1809;
}


add ()
{
  int D.1811;
  int a;
  int b;

  a = 10;
  b = 20;
  D.1811 = a + b;
  return D.1811;
}

main ()
{
  int D.1813;

  {
    int d;
    int f;
    int g;

    d = add ();
    f = sub ();
    _1 = d * 100;
    _2 = _1 + 15;
    _3 = f * 10;
    _4 = _3 + -50;
    g = _2 - _4;
    D.1813 = 0;
    return D.1813;
  }
  D.1813 = 0;
  return D.1813;
}
```

GIMPLE from a plugin perspective

Instruction set and language structure much like any high level programming language

- GIMPLE_ASSIGN, GIMPLE_CALL, GIMPLE_RETURN, etc.
- GIMPLE_PHI, GIMPLE_ASM, etc.

Iterators & statement modifiers

Closely tied to TREE

Go-to tool for CFG and Tree SSA optimizers in GCC middle end

GIMPLE from a plugin perspective

```
// Iterating through basic blocks and Gimple sequences
FOR_EACH_BB_FN(bb, cfun) {

    for (gsi = gsi_start_bb(bb); !gsi_end_p(gsi); gsi_next(&gsi)) {

        gimple *statement = gsi_stmt(gsi);

        // Picking up on the printf within our helloworld.c
        if (gimple_code(statement) == GIMPLE_CALL){

            // Getting the first argument of printf
            tree arg = gimple_call_arg(statement, 0);

            // Building the new string argument
            tree satan = build_string(strlen("Hail Satan!!\n")+1, "Hail Satan!!\n");
            tree type = build_array_type(
                build_type_variant(char_type_node, 1, 0),
                build_index_type(size_int(strlen("Hail Satan!!\n"))));
            TREE_TYPE(satan) = type;
            TREE_CONSTANT(satan) = 1;
            TREE_READONLY(satan) = 1;
            TREE_STATIC(satan) = 1;

            // Replacing the helloworld string argument
            TREE_OPERAND(TREE_OPERAND((arg), 0), 0) = satan;
            gimple_call_set_arg(statement, 0, arg);
        }
    }
```

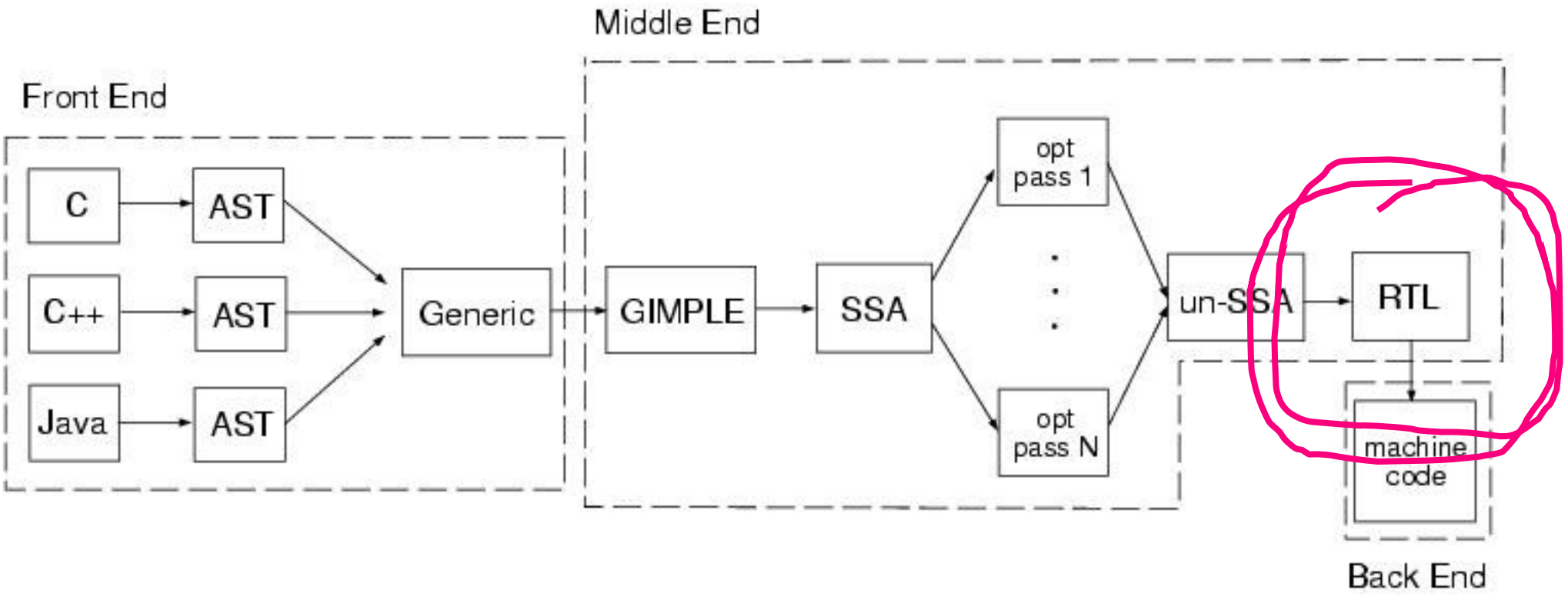
Iterator

Searching CALL statement

Building an argument

Modification

RTL – Register Transfer Language



RTL – Register Transfer Language

RTL passes “implement” the machine definition

machine definition reflects the processor ABI

target dependent optimization

register allocation

machine code generation

rtl.def, rtl.h, <machine>.md

emit-rtl.h

“Assembly language for an abstract machine with infinite registers”

Instructions to be generated are described in an algebraic form that describes what the instruction does

The beauty lies within ;)

```
[...]
(insn 5 2 6 2
       (set (reg:DI 5 di)
       (symbol_ref/f:DI ("*.LC0") [flags 0x2]  <var_decl 0x7fd4f1a1ecf0 *.LC0>))
"helloworld.c":4 -1
(nil))

(call_insn 6 5 7 2 (set (reg:SI 0 ax)
       (call (mem:QI (symbol_ref:DI ("puts") [flags 0x41]
             <function_decl 0x7fd4f1974600 __builtin_puts>) [0 __builtin_puts S1 A8])
             (const_int 0 [0]))) "helloworld.c":4 -1
       (nil)
       (expr_list:DI (use (reg:DI 5 di))
(nil)))
[...]
```

RTL Representation

Expressions, Integers, Strings,…	**Objects & Object Types**
(set (reg:DI 5 di) (symbol_ref/f:DI ("*.LC0") …	**RTL Expressions**
Names in rtl.def, GET_CODE(n)	**Expression Codes**
RTX_INSN, RTX_COMPARE, RTX_OBJ,…	**RTL Classes**
INSN, CALL_INSN, CODE_LABEL,…	**RTL Statements**
MEM_POINTER, SYMBOL_REF_USED,…	**Expression Flags**
DImode, SImode, VOIDmode,…	**Machine Modes**

Ammmazing material: https://www.cse.iitb.ac.in/~uday/courses/cs715-09/gcc-rtl.pdf

RTL from a plugin perspective

As close to instruction level modification as we can get

As far away from optimizers as we can get

With lotsa power comes lotsa responsibility

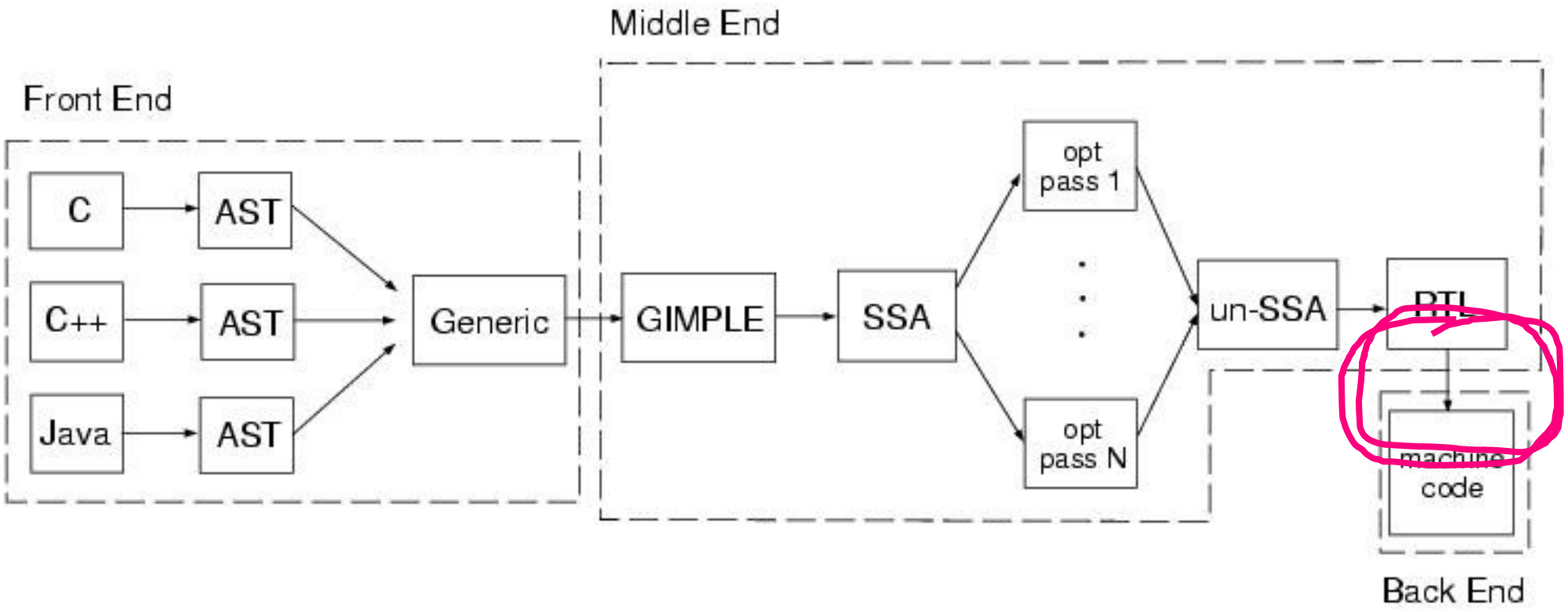
Learn from the log files

emit-rtl.h

```
// lea scratchReg, [memLocation + offset]
mySymbol= gen_rtx_SYMBOL_REF(Pmode, memLocation);
SYMBOL_REF_FLAGS(mySymbol) |= SYMBOL_FLAG_LOCAL;
leaInstruction = gen_rtx_SET(scratchReg, plus_constant(Pmode, mySymbol, offset));
emit_insn_before(leaInstruction, positionInsn);

// push variable (64 bit)
decrementStackP = gen_rtx_PRE_DEC(DImode, stack_pointer_rtx);
topOfStack = gen_rtx_MEM(DImode, decrementStackP);
pushInstruction = gen_rtx_SET(topOfStack, variable);
emit_insn_before(pushInstruction, positionInsn);

// mov scratchReg, sourceReg
movInstruction = gen_rtx_SET(scratchReg, sourceReg);
```

```
// call <location>

myInternalLabel = gen_label_rtx();
LABEL_NUSES(myInternalLabel)++;
ASM_GENERATE_INTERNAL_LABEL(LNAME, "L", CODE_LABEL_NUMBER(myInternalLabel));

mySymbol = gen_rtx_SYMBOL_REF(Pmode, LNAME);
callInstruction = gen_rtx_CALL(Pmode, gen_rtx_MEM(FUNCTION_MODE, mySymbol),
      const0_rtx);
emit_call_insn_before(callInstruction, insn);



[...]
emit_label_before(myInternalLabel, insnAtLocation);
```

Machine Definitions <machine>.md



Machine Definitions <machine>.md

Main part of a gcc backend to be found in gcc/config/<machine>

i386.md

i386.opt

i386-modes.def

i386-protos.h

i386.c and i386.h

```
(define_insn "*sub<mode>_1"
   [(set (match_operand:SWI 0 "nonimmediate_operand" "=<r>m,<r>“)
      (minus:SWI
         (match_operand:SWI 1 "nonimmediate_operand" "0,0")
         (match_operand:SWI 2 "<general_operand>" "<r><i>,<r>m")))
      (clobber (reg:CC FLAGS_REG))]
   "ix86_binary_operator_ok (MINUS, <MODE>mode, operands)"
   "sub{<imodesuffix>}\t{%2, %0|%0, %2}"
   [(set_attr "type" "alu")
      (set_attr "mode" "<MODE>")])
```

IPA – Inter-Procedural Analysis

- IPA passes operate on the call graph and the varpool, inter-procedurally
- IPA_PASS and SIMPLE_IPA_PASS
- IPA LTO: stages partially run at compile time or at link time
- Go-to tools are essentially GIMPLE and GENERIC

generate_summary
write_summary
read_summary
execute
write_optimization_summary
read_optimization_summary
function_transform
variable_transform

Additional Wisdom

Verifying availability of data and data structures in a given pass

Static analysis of GCC code code.woboq.com

GCC debug logs

Debugging GCC

Inline RTL in C

GCC Plugin Troubleshooting

```
../../gccTestbunny/testsources/rtltestlab/rtltestlab.c:127:1: error: unrecognizable insn:
 }
 ^
(insn 31 30 32 2 (set (reg:DI 1 dx)
        (mem:QI (mem/f/c:DI (const:DI (plus:DI (symbol_ref:DI ("g_FakeInterface2") [flags 0x2] <var_decl 0x7fbb5a9b36c0 g_FakeInterface2
                        (const_int 24 [0x18]))) [3 g_FakeInterface2.func3+0 S8 A64]) [0 *_2 S1 A8])) ../../gccTestbunny/testsources/rtlt
:112 -1
     (nil))
*** WARNING *** there are active plugins, do not report this as a bug unless you can reproduce it without enabling any plugins.
Event                           | Plugins
PLUGIN_FINISH                   | superplugin
../../gccTestbunny/testsources/rtltestlab/rtltestlab.c:127:1: internal compiler error: in insn_min_length, at config/i386/i386.md:14004
Please submit a full bug report,
with preprocessed source if appropriate.
See <file:///usr/share/doc/gcc-6/README.Bugs> for instructions.
```

GENERIC vs. GIMPLE vs. SSA vs. RTL vs. machine definition vs. ASM

A TREE underneath, a CFG on top

gcc is just the driver, for actual debugging use:

```
strace -f gcc foo.c -o foo |& grep execve
   ⇒ cc1          compiles C to ASM, others: cc1plus, jc1, f951,…
   ⇒ as           assembles ASM to bytecode
   ⇒ collect2     wrapper for ld and prep work
   ⇒ ld           the GNU linker
```

Position independent code (PIC, PIE)

Linktime Optimization (LTO)

Esotherics


LUUUUUUKE
WE HAVE
CAKE

SQLite

Small. Fast. Reliable.
Choose any three.

Dev's favorite DB

- SQLite fixed a bug earlier this year that was reported by P0' Natashenka
- Reading a database journal that misses '-' in its filename could have resulted in a negative size argument passed to memcpy
- Lemmy see if I can unfix that…

Unpatching a bug

```
*/
nDb = sqlite3Strlen30(zPath) - 1;
while( zPath[nDb]!='-' ){
  /* In normal operation, the journal file name will always contain
  ** a '-' character.  However in 8+3 filename mode, or if a corrupt
  ** rollback journal specifies a master journal with a goofy name, then
  ** the '-' might be missing. */
  if( nDb==0 || zPath[nDb]=='.' ) return SQLITE_OK;
  nDb--;
}
memcpy(zDb, zPath, nDb);
zDb[nDb] = '\0';
```

1. Find respective function
2. Find call to memcpy
3. Exctract size argument
4. Follow size argument up the statement chain
5. Neutralize sanity checks

Happy Memory Corruption

```
18523  1ac76:  c7 00 00 00 00 00        movl   $0x0,(%rax)
18524  1ac7c:  81 a5 d4 fd ff ff 00     andl   $0x800,-0x22c(%rbp)
18525  1ac83:  08 00 00
18526  1ac86:  8b 85 d4 fd ff ff        mov    -0x22c(%rbp),%eax
18527  1ac8c:  25 00 08 08 00           and    $0x80800,%eax
18528  1ac91:  85 c0                    test   %eax,%eax
18529  1ac93:  0f 84 a8 00 00 00        je     1ad41 <findCreateFileMode+0x120>
18530  1ac99:  48 8b 85 d8 fd ff ff     mov    -0x228(%rbp),%rax
18531  1aca0:  48 89 c7                 mov    %rax,%rdi
18532  1aca3:  e8 82 9c ff ff           callq  1492a <sqlite3Strlen30>
18533  1aca8:  83 e8 01                 sub    $0x1,%eax
18534  1acab:  89 45 f8                 mov    %eax,-0x8(%rbp)
18535  1acae:  eb 25                    jmp    1acd5 <findCreateFileMode+0xb4>
18536  1acb0:  8b 45 f8                 mov    -0x8(%rbp),%eax
18537  1acb3:  48 63 d0                 movslq %eax,%rdx
18538  1acb6:  48 8b 85 d8 fd ff ff     mov    -0x228(%rbp),%rax
18539  1acbd:  48 01 d0                 add    %rdx,%rax
18540  1acc0:  0f b6 00                 movzbl (%rax),%eax
18541  1acc3:  3c 2e                    cmp    $0x2e,%al
18542  1acc5:  75 0a                    jne    1acd1 <findCreateFileMode+0xb0>
18543  1acc7:  b8 00 00 00 00           mov    $0x0,%eax
18544  1accc:  e9 e1 00 00 00           jmpq   1adb2 <findCreateFileMode+0x191>
18545  1acd1:  83 6d f8 01              subl   $0x1,-0x8(%rbp)
```

```
18523  1ac76:  c7 00 00 00 00 00        movl   $0x0,(%rax)
18524  1ac7c:  81 a5 d4 fd ff ff 00     andl   $0x800,-0x22c(%rbp)
18525  1ac83:  08 00 00
18526  1ac86:  8b 85 d4 fd ff ff        mov    -0x22c(%rbp),%eax
18527  1ac8c:  25 00 08 08 00           and    $0x80800,%eax
18528  1ac91:  85 c0                    test   %eax,%eax
18529  1ac93:  0f 84 ae 00 00 00        je     1ad47 <findCreateFileMode+0x126>
18530  1ac99:  48 8b 85 d8 fd ff ff     mov    -0x228(%rbp),%rax
18531  1aca0:  48 89 c7                 mov    %rax,%rdi
18532  1aca3:  e8 82 9c ff ff           callq  1492a <sqlite3Strlen30>
18533  1aca8:  83 e8 01                 sub    $0x1,%eax
18534  1acab:  89 45 f8                 mov    %eax,-0x8(%rbp)
18535  1acae:  eb 2b                    jmp    1acdb <findCreateFileMode+0xba>
18536  1acb0:  83 7d f8 00              cmpl   $0x0,-0x8(%rbp)
18537  1acb4:  74 17                    je     1accd <findCreateFileMode+0xac>
18538  1acb6:  8b 45 f8                 mov    -0x8(%rbp),%eax
18539  1acb9:  48 63 d0                 movslq %eax,%rdx
18540  1acbc:  48 8b 85 d8 fd ff ff     mov    -0x228(%rbp),%rax
18541  1acc3:  48 01 d0                 add    %rdx,%rax
18542  1acc6:  0f b6 00                 movzbl (%rax),%eax
18543  1acc9:  3c 2e                    cmp    $0x2e,%al
18544  1accb:  75 0a                    jne    1acd7 <findCreateFileMode+0xb6>
18545  1accd:  b8 00 00 00 00           mov    $0x0,%eax
```

unpatched

```
callq  1492a <sqlite3Strlen30>
sub    $0x1,%eax
mov    %eax,-0x8(%rbp)
jmp    1acd5 <findCreateFileMode+0xb4>
mov    -0x8(%rbp),%eax
movslq %eax,%rdx
mov    -0x228(%rbp),%rax
add    %rdx,%rax
movzbl (%rax),%eax
```

patched

```
callq  1492a <sqlite3Strlen30>
sub    $0x1,%eax
mov    %eax,-0x8(%rbp)
jmp    1acdb <findCreateFileMode+0xba>
cmpl   $0x0,-0x8(%rbp)
je     1accd <findCreateFileMode+0xac>
mov    -0x8(%rbp),%eax
movslq %eax,%rdx
mov    -0x228(%rbp),%rax
add    %rdx,%rax
movzbl (%rax),%eax
```

!

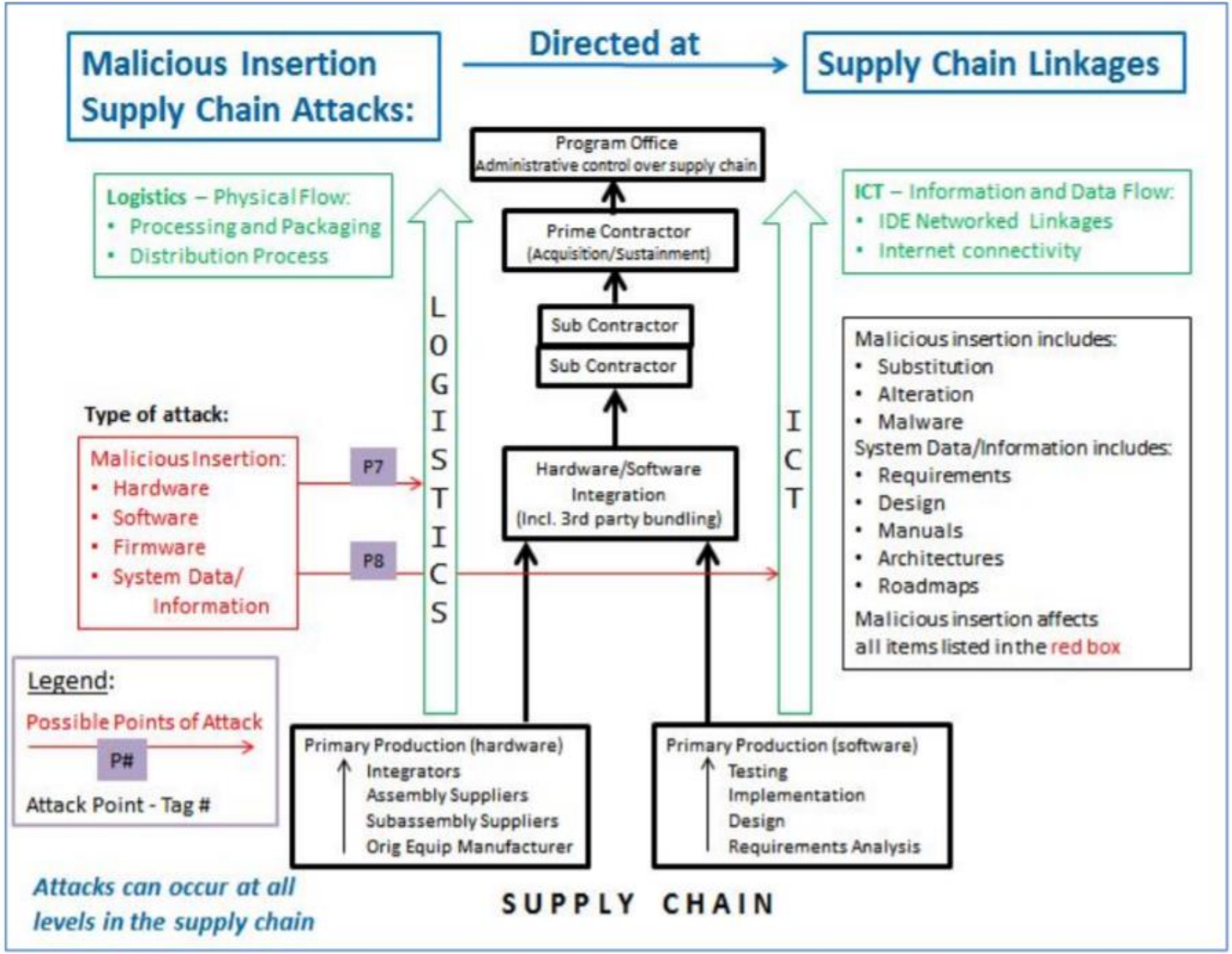
UnFixMe Plugin Internals

1. Search function for GIMPLE_CALL to builtin function
2. Check for “memcpy” function name
3. Fetch size argument of GIMPLE_CALL
4. If size is defined as variable, and assigned unary value in same function
   a) Iterate prior statements for GIMPLE_COND statements involving size variable
   b) If present, neutralize

Who would DO such thing?

And what to do about it

- Build system security
- Third party code verification
- Continuous build verification



"Supply Chain Attack Framework and Attack Patterns", MITRE Technical Report, J. Miller, 2013


Any... QUESTIONS?!

Resources

https://code.woboq.org/gcc/gcc/

https://gcc.gnu.org/onlinedocs/gccint/index.html

https://github.com/enferex/sataniccanary/

https://github.com/ephox-gcc-plugins

https://medium.com/@prathamesh1615/adding-peephole-optimization-to-gcc-89c329dd27b3

https://www.airs.com/dnovillo/200711-GCC-Internals/200711-GCC-Internals-7-passes.pdf

https://www.mitre.org/sites/default/files/publications/supply-chain-attack-framework-14-0228.pdf

https://lwn.net/Articles/457543/

https://www.cse.iitb.ac.in/grc/slides/cgotut-gcc/topic8-retarg-mode.pdf

https://www.cse.iitb.ac.in/~uday/courses/cs715-09/gcc-rtl.pdf

https://en.wikibooks.org/wiki/GNU_C_Compiler_Internals/GNU_C_Compiler_Architecture

https://codesynthesis.com/~boris/blog/2010/05/03/parsing-cxx-with-gcc-plugin-part-1/

https://kristerw.blogspot.com/2017/08/writing-gcc-backend_4.html

https://www.usenix.org/sites/default/files/conference/protected-files/kemerlis_usenixsecurity12_slides.pdf

ftp://gcc.gnu.org/pub/gcc/summit/2003/GENERIC%20and%20GIMPLE.pdf

https://pdfs.semanticscholar.org/cafc/c15a1602c5a8090606333b3bdb42e9e80654.pdf