



Compilers: The Old New Security Frontier

Brad Spengler
Open Source Security, Inc.
BlueHat IL
March 2022

whoami

- For ~14 years, by day, Windows malware analyst/reverse engineer/deobfuscator/sandbox developer/kernel developer
- For ~18 years, by night, Linux kernel security, grsecurity developer
- For ~3 years, full-time @ Open Source Security Inc.
 - “Managing” an insanely talented team
 - More realistically: providing the environment and resources needed to fully explore and tackle the most difficult computer security problems for customers with high security needs

Compilers + Security: Why Should I Care?

- Thesis: the next generation of security defenses necessarily involves compilers
- We've gotten as far as we can with NX/ASLR/etc
- Compiler plugins in particular provide unique defense benefits
- With advent of Spectre, manual approaches don't scale

I thought the Linux kernel already has mitigations in place to protect against transient execution attacks. So did Kasper actually find any gadgets?

Yep, it found 1379 previously unknown gadgets.

<https://www.vusec.net/projects/kasper/>

- Helps ensure security properties despite third-party changes
- Precise control allows for higher performance through optimizations than afforded by blanket naïve approaches

Compilers + Security: Why Should I Care?

- State of the art in offense from 2000 to 2005 is now commonplace
 - Better/more accessible explanations of the exploitation techniques (ROP etc)
 - Improved automation (ROP gadget finders)
- Mainstream defense has struggled to reach even 2003 state of the art
 - [PAGEEXEC/MPROTECT](#) in 2000
 - [ASLR](#) in 2001
 - KERNEXEC in 2003 (!)
 - Before XP SP2 even, the first big entry in the Trustworthy Computing initiative

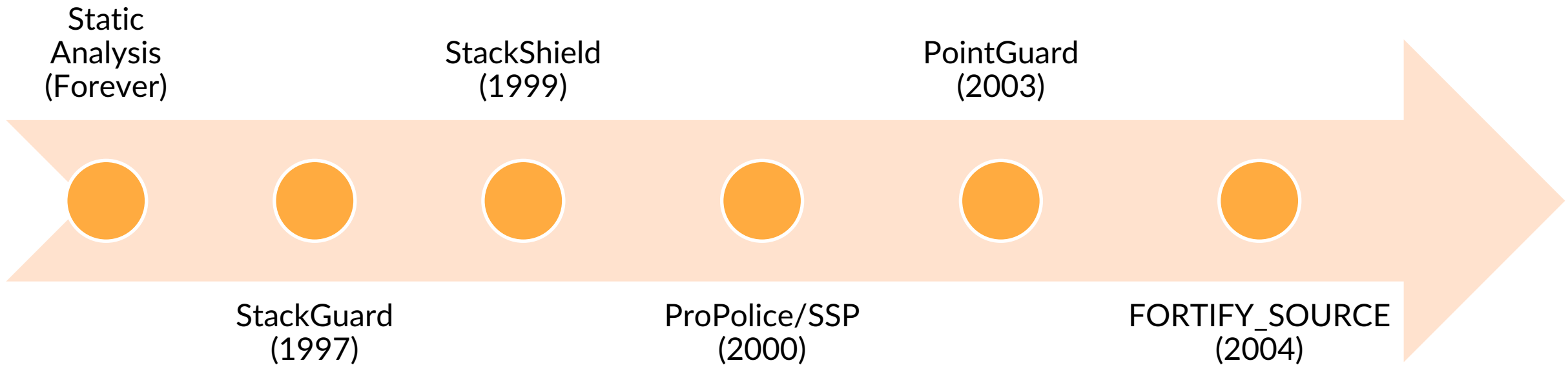
Compilers + Security: Why Should I Care?

- Major benefits to being ahead of the exploitation curve
- Cannot rest on laurels and let defense stagnate
- If exploit vectors/techniques get ahead, hammered with the same techniques for years
 - My `commit_creds()` technique is still used 13 years later, enshrined in books and university courses
 - Previously “hardened” OSes lose that designation
- Compiler-based defense is the only way to stay ahead and provide the same security guarantees across all users
 - SMEP vs KERNEXEC plugin / CET vs RAP plugin

Compilers + Security: Outline

- Incomplete Linux/GCC/C/C++-centric history
 - Early advances
 - Long lull
 - Plugins and new advances
- Advantages of compiler (plugin) defense
- Roadblocks/problems

Compilers + Security: Not New



Compilers + Security: Not New

- Most early work (1997-2005) created as EGCS/GCC enhancements for the commercial [Immunix](#) distribution
 - Sold to Novell in 2005 for an undisclosed amount
 - Market for commercial hardened Linux distribution “never panned out” (eWeek)
- Prior to that, and parallel to Immunix’s existence were other projects tackling similar problems without compilers
 - [Openwall](#) – 1997
 - [PaX](#) – 2000
 - [grsecurity](#) - 2001
- After Immunix was gone, not much happening in the production compiler+security space for years
 - At least until 2010-2012

Compilers + Security: Newish

- GCC 4.5.0 released in April 2010
 - First version with plugin support, driven by LLVM competition
 - Eliminates a barrier to entry for compiler enhancements
- See PaX Team's 2013 H2HC presentation on GCC plugins:
 - <https://pax.grsecurity.net/docs/PaXTeam-H2HC13-PaX-gcc-plugins.pdf>
 - Covered [CONSTIFY](#) (2011), [KERNEXEC](#) (2011), [STACKLEAK](#) (2011), [LATENT_ENTROPY](#) (2012), [SIZE_OVERFLOW](#) (2012), [STRUCTLEAK](#) (2013)
- Since then (security-wise):
 - [RANDSTRUCT](#)
 - [RAP](#)
 - [RESPECTRE](#)
 - [AUTOSLAB](#)

Respectre Example

```
void victim_function_v01(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}
```

```
void victim_function_v01(size_t x) {
    if (x < array1_size) {
        size_t y = array_index_nospec(x, array1_size);
        temp &= array2[array1[y] * 512];
    }
}
```

Visual Studio 2022 /Qspectre

```
cmp     rcx, cs:array1_size
jnb     short locret_14000105D
lfence
lea     rdx, pe_baseaddr
movzx   eax, rva array1[rdx+rcx]
shl     rax, 9
movzx   eax, byte ptr [rax+rdx+30C0h]
and     cs:temp, al
```

Respectre

```
mov     rax, cs:array1_size
cmp     rax, rdi
jbe     short loc_57C
cmp     rdi, rax
sbb     rax, rax
and     rdi, rax
movzx   eax, ds:array1[rdi]
shl     eax, 9
cdq
mov     al, ds:array2[rax]
and     cs:temp, al
```

Optimal Ad-Hoc Fix (source above)

```
mov     rax, cs:array1_size
cmp     rax, rdi
jbe     short loc_57C
cmp     rdi, rax
sbb     rax, rax
and     rdi, rax
movzx   eax, ds:array1[rdi]
shl     eax, 9
cdq
mov     al, ds:array2[rax]
and     cs:temp, al
```

```
init/main.c: In function 'victim_function_v01':
init/main.c:1722:32: note: Spectre v1 array index bound '<unknown>'
    temp &= array2[array1[x] * 512];
                        ^
```

```
init/main.c:1722:32: note: Spectre v1 array index mask adjust: keep constbound: no
Spectre v1 bound def stmt [init/main.c:1721:12] array1_size.48_3 = array1_size;
```

See also: https://grsecurity.net/teardown_of_a_failed_linux_its_spectre_fix

Compilers + Security: Newish

- Plugins proved useful for more than just novel security features
 - [SANCOV](#)
 - [INITIFY](#)
 - Injecting inline assembly at source level in arbitrary locations
- Provided “backports” of newer compiler features to all plugin-capable versions
 - Retpolines
 - [KCOV_COMPARISONS](#)
- Fixed compiler bugs and bad user/dev experience cases
 - [__optimize__](#) (“no-stack-protector”)
 - GCC ≥ 4.7 (2012) && < 8 (2018) bug resulting in uninitialized padding bytes for local vars

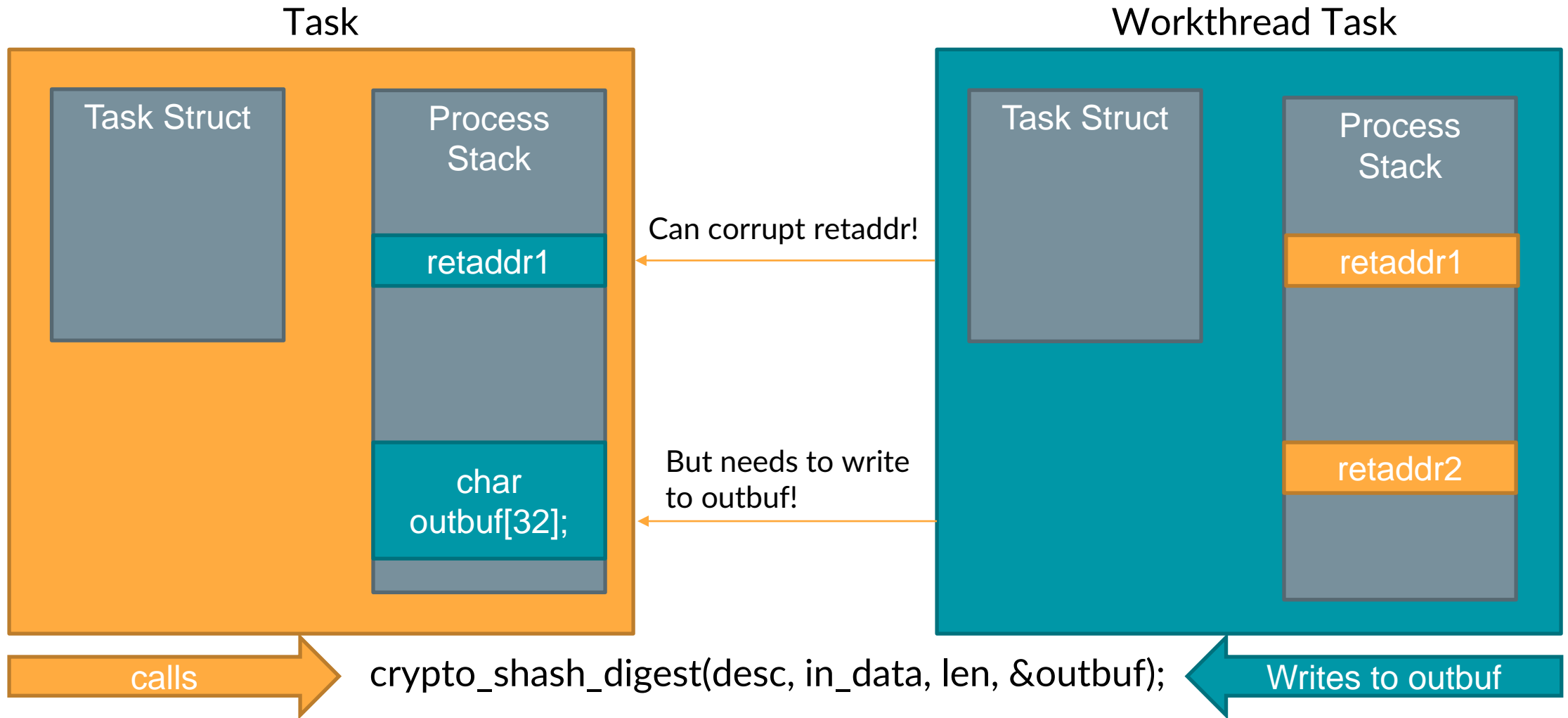
Compilers + Security: Newish (Outside)

- Google's sanitizers: ASan/TSan/MSan/UBSan
 - **Huge** boon for fuzzing
- [Clang CFI](#)
 - Used in Android, on ARM64 in Linux kernel
- [AFL LLVM Plugin](#)
- Upstream Linux getting their feet wet even
 - Plugin for ARM per-task SSP canary
 - Present since Linux 5.0
 - Vs global/shared/unchanged canary on SMP
 - Newer version that uses TLS register

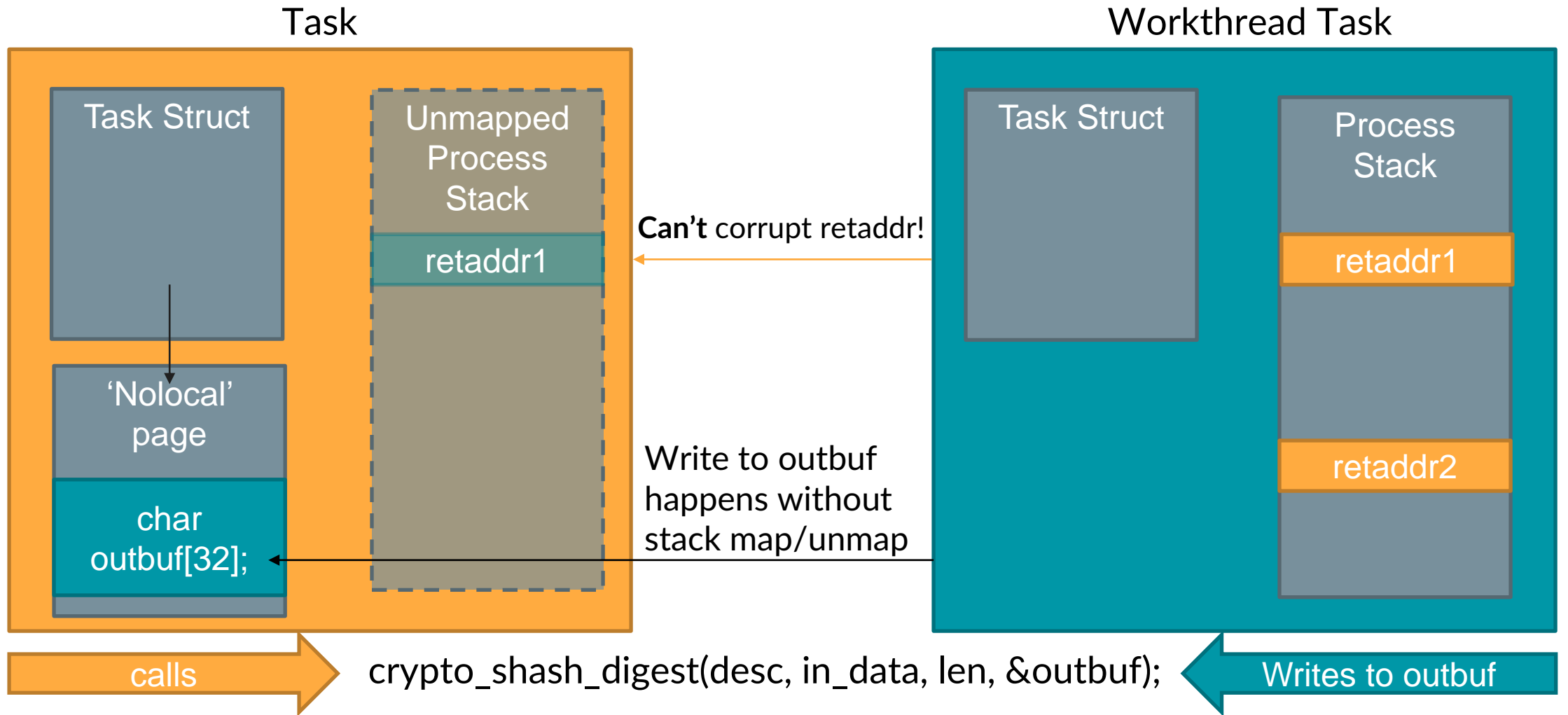
Compilers + Security: The New New

- Probabilistic backward-edge CFI checks ([PAX_RAP_XOR](#)) – April 2020
- Defense against Speculative Blind ROP ([BlindSide](#))
 - BlindSide paper – Sept 2020
 - PAX_RAP_CALL_NOSPEC – Dec 2020
- “Private” kernel stacks – February 2022
 - Addresses attack that caused [RFG](#) to be [shelved](#), without requiring [CET](#)/HW shadow stacks
 - All-but-current process stacks not mapped
 - IRQ handlers cannot view other CPU IRQ stacks
 - `__nolocal` attribute for on-stack vars intended to be accessed by remote tasks
 - `__nolocal_arg()` attribute for functions to document/convert local vars used by callers

PAX_PRIVATE_KSTACKS Example (without)



PAX_PRIVATE_KSTACKS Example (with)



Compilers + Security: The New New (Outside)

- -fanalyzer
 - Still a lot of noise, but improving
 - <https://www.youtube.com/watch?v=b1fO-RLtDME>
 - Extendable by plugins!
- Shadow Call Stack support
- __builtin_dynamic_object_size()
 - Can make use of some non-const allocation sizes vs basic/older `__builtin_object_size()` that severely limited coverage/usefulness
- -mzero-caller-saved-regs
- -ftrivial-auto-var-init
- AFL++ GCC plugin

Compiler (Plugin) Defense Advantages

- Backporting is generally trivial
 - Plugin code can mostly be copied to older kernel versions verbatim
 - Compare to trying to backport hundreds/thousands of manually-created [struct_size\(\)](#) conversions and wasted developer time
- Maintainable, scalable, adaptable to third-party modifications
 - Especially important for fast-moving, highly modified projects like the Linux kernel
 - Try manually converting 40k+ calls to `k*alloc()`, backporting it all, and maintaining it forever

Compiler (Plugin) Defense Advantages (Cont.)

- Deeper integration with codebase under compilation/instrumentation (emit calls to functions you provide, etc)
- Codebase-specific static analysis that would never be shipped with the compiler proper
- Plugins can evolve with the codebase, making the latest codebase automatically use the latest security functionality vs an additional developer/admin requirement on toolchain versions
- Otherwise complex changes can be implemented with surprisingly few lines of code

Compiler (Plugin) Defense Advantages - Example

```
diff --git a/scripts/gcc-plugins/rap_plugin/rap_retpoline.c b/scripts/gcc-plugins/rap_plugin/rap_retpoline.c
index 0e97a9fd9b66..988629adb0fa 100644
--- a/scripts/gcc-plugins/rap_plugin/rap_retpoline.c
+++ b/scripts/gcc-plugins/rap_plugin/rap_retpoline.c
@@ -313,6 +313,18 @@ static unsigned int rap_retpoline_execute(void)
     if (INSN_DELETED_P(insn))
         continue;

+
+     // put a trap after returns to stop speculation past them
+     if (returnjump_p(insn)) {
+         rtx_insn *stuffing;
+
+         start_sequence();
+         expand_builtin_trap();
+         stuffing = get_insns();
+         end_sequence();
+         insn = emit_insn_after(stuffing, insn);
+         continue;
+     }

     if (JUMP_P(insn) && !returnjump_p(insn)) {
         insn = rap_handle_indirect_jump(insn, false);
     }
}
```

Compiler (Plugin) Defense Problems

- “Canonicalization of expressions”
 - `var - 1 -> var + 0xffffffff`
 - `var & 0xffff -> (unsigned short)var`
 - Can be indistinguishable from real overflows/truncations and are difficult/impossible to fix via the plugin itself
 - This happens in the front-end prior to any plugin invocation
- Not everything can be fixed/improved via a plugin
- Minimal documentation
 - “Use the source, Luke!”
 - `-fdump-tree-all/-fdump-ipa-all/-fdump-rtl-all` are your friends

Compiler (Plugin) Defense Problems - Example

```
__attribute__((rap_hash (1666116122)))
victim_function_v01 (size_t x)
{
    long unsigned int array1_size.48_3;
    unsigned char _5;
    int _6;
    int _7;
    unsigned char _8;
    unsigned char temp.49_9;
    unsigned char _10;

    <bb 2>:
    array1_size.48_3 = array1_size;
    if (x_4(D) < array1_size.48_3)
        goto <bb 3>;
    else
        goto <bb 4>;

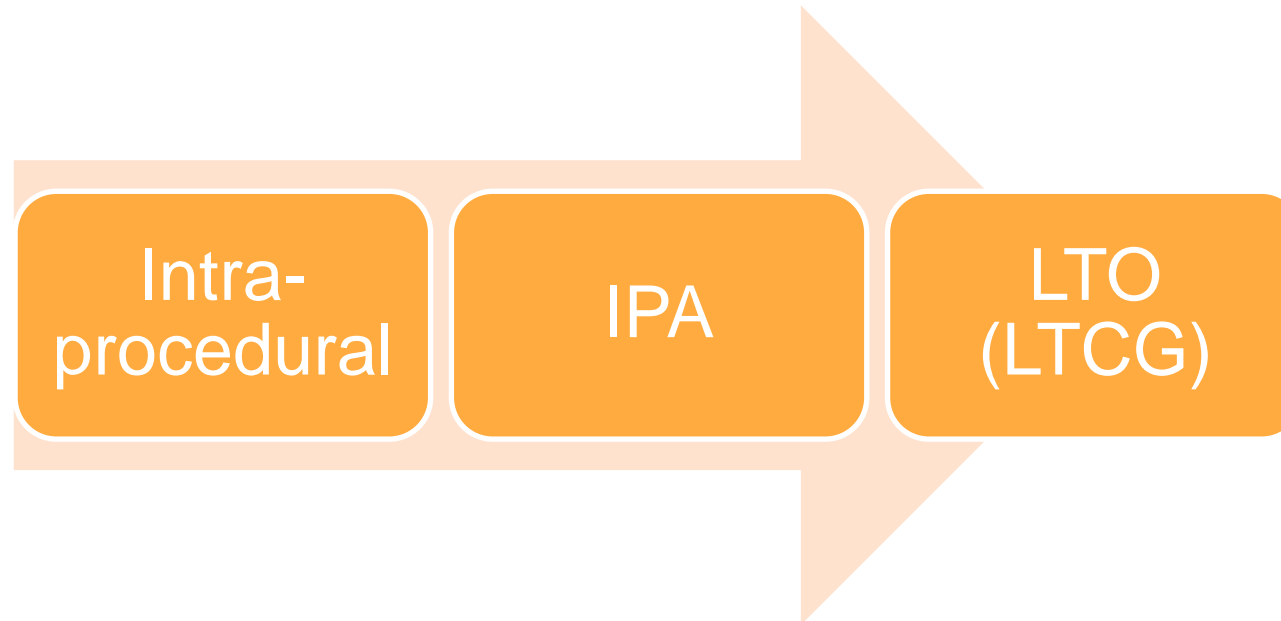
    <bb 3>:
    x_12 = array_index_nospec_u64 (x_4(D), array1_size.48_3, 0);
    _5 = array1[x_12];
    _6 = (int) _5;
    _7 = _6 * 512;
    _8 = array2[_7];
    temp.49_9 = temp;
    _10 = _8 & temp.49_9;
    temp = _10;

    <bb 4>:
    return;
}
```

Compiler (Plugin) Defense Problems (Cont.)

- Limitations of static analysis
 - One of the most important things to keep in mind
 - Otherwise end up exaggerating effectiveness of feature (e.g. FORTIFY_SOURCE)

Less information
Weaker analysis
Low(er)-hanging fruit
Easier implementation



More information
Stronger analysis
More complex findings
Harder implementation

Compiler (Plugin) Defense Problems (Cont.)

- Requires lots of testing / defensive programming
 - Test with debug/checked versions of the compiler to catch issues that release versions won't expose
 - Only operate based on explicitly-matched patterns, use `gcc_assert()` liberally
 - These patterns can and will change from one GCC version to another
 - Requires verifying expected instrumentation exists, vs just “the code works”
 - Code-gen bugs are no joke
- Supporting all plugin-capable GCC versions

Compiler (Plugin) Defense Problems – “Mixed Binaries”

- The “Mixed Binary” problem, in essence:
 - Take C/C++ codebase with security-relevant instrumentation inserted by compiler
 - Modernize/secure subset of above codebase in a memory-safe language like Rust
 - Execute both in the same address space
 - Create a whole new world of problems for yourself as the “safe” code is abused to enable exploitation for the unsafe C/C++
- We are funding [GCC Rust](#) to help address this problem

Compiler (Plugin) Defense Problems – “Mixed Binaries”

- <https://www.cs.ucy.ac.cy/~elathan/papers/tops20.pdf>
- “Our assessment concludes that CFI can be completely nullified through Rust or Go code by constructing much simpler attacks than state-of-the-art CFI bypasses.”
- MS specifically identified this problem:
 - <https://msrc-blog.microsoft.com/2020/08/17/control-flow-guard-for-clang-llvm-and-rust/>

Whilst this is a very contrived example (and hopefully no real codebase contains this code), it illustrates the possibility of an attacker finding a memory corruption vulnerability in the linked C/C++ code and using it to violate control flow integrity in the (safe) Rust code. Even though the C/C++ code is compiled with CFG enabled, we also need to enable CFG for the Rust code to mitigate this vulnerability.

Compiler (Plugin) Defense Problems (Cont.)

- Mixed binary problem will become much more important in the future for other reasons
- Certain defenses introduce new ABI
 - If you want CFI, you can't necessarily mix and match
 - Now standards and politics get involved
 - LLVM vs GCC vs Visual Studio
 - Do late arrivals get stuck with inferior early-adopted solutions?
- Integration/evolution benefits of compiler plugins can't necessarily be realized unless you control the compilation of the entire project in question
 - Kernel is near ideal, userland not so much (unless it is a specifically-tailored distribution)
 - May be possible to work around, but will result in non-optimal instrumentation
- Inline/external assembly needs adapted for protection coherence

Compiler + Security: Takeaways

- Developer resources are already spread thin, shift manual/error-prone conversions/hardening to the compiler wherever possible
- Important to understand compiler theory/design, seek out ways to develop related skills
 - Necessary to stay ahead in the future of memory corruption
 - Help ensure good ideas with solid security properties in upstream compilers
 - Compiler developers need your help: <https://llsoftsec.github.io/llsoftsecbook/>
- Getting to the right solution is important in security, but it's even more important to get to that solution at a time where it matters, rather than long after the fact
- Security performance budgets aren't increasing, compiler-based defense helps cram more in
- If you're interested in Rust and compilers, opportunities exist to carve out your place in the relatively new GCC Rust project

Thank you!

Grsecurity is created by



OPEN
SOURCE
SECURITY