Overview

Go 1.15 adds support for enabling code generation adjustments to mitigate the effect of two variants of the Spectre family of CPU vulnerabilities. The compiler and assembler both have a new flag `-spectre` that is given a list of Spectre mitigations to enable, as in `-spectre=index` or `-spectre=index,ret`. The special case `-spectre=all` enables all available mitigations.

The `index` mitigation can be enabled in the compiler and changes code generation to emit protections against Spectre variant 1 ("bounds check bypass"). The mitigation ensures that the CPU does not access arbitrary memory, by masking the index to zero when speculating incorrectly. This change typically slows down execution by around 5-10%; the exact slowdown depends on the workload.

The `ret` mitigation can be enabled in both the compiler and the assembler and changes code generation to emit protections against Spectre variant 2 ("branch target injection"). This mitigation replaces every indirect call instructions with use of a retpoline gadget. This change typically slows down execution by around 10-15%; again, the exact slowdown depends on the workload.

Applicability

At time of writing, we do not use either of these mitigations for Go programs running at Google, nor do we anticipate doing so. They are included in the Go toolchain as a kind of "defense in depth" for users with very special use cases (or significant paranoia).

These mitigations would only be necessary when there is a potential Spectre attack against a Go program, which would require all of the following to be true. First, an attacker must be able to run arbitrary code on the same CPUs as a victim Go program containing a secret. Second, the attacker must be able to make some kind of HTTP or RPC requests to the victim Go program. Third, those requests have to trigger a potentially vulnerable code fragment to speculate into attacker-selected behavior. Most commonly this would mean using an arbitrary attacker-provided integer to index a slice or array. These three conditions are only very rarely all true at the same time.

Example

To build a program with both mitigations (and any future mitigations) enabled in all packages, use:

```
go build -gcflags=all=-spectre=all -asmflags=all=-spectre=all program
```

Acknowledgements

Thanks to Andrea Mambretti *et al.* for sharing their paper (linked below) ahead of publication. And thanks to them, Chandler Carruth, and Paul Turner for helpful discussions.

References

"Spectre Attacks: Exploiting Speculative Execution"
by Paul Kocher *et al.* (The definitive paper.)

"Speculative Buffer Overflows: Attacks and Defenses"
by Vladimir Kiriansky and Carl Waldspurger.

"Retpoline: a software construct for preventing branch-target-injection"
by Paul Turner.

"A Systematic Evaluation of Transient Execution Attacks and Defenses"
by Claudio Canella *et al.* (Good summary of variants.)

"Spectre is here to stay: An analysis of side-channels and speculative execution"
by Ross McIlroy *et al.* (These are not going away.)

"Spectre Returns! Speculation Attacks using the Return Stack Buffer"
by Esmaeil Mohammadian Koruyeh *et al.* (Even return prediction isn't safe.)

"Speculative Load Hardening"
by Chandler Carruth. (What LLVM does to prevent speculative out-of-bounds access.)

"Bypassing memory safety mechanisms through speculative control flow hijacks"
by Andrea Mambretti *et al.* (Examination of effects on memory-safe languages.)