Multi-core IFC

Securing the space-time continuum

Gary Soeller and Deian Stefan
UC San Diego
{gsoeller, deian}@cs.ucsd.edu

Concurrent information-flow control (IFC) systems are a promising defense against information leakage and corruption in the presence of untrusted code. For example, the language-level IFC system LIO [6] has been used to build web platforms largely consisting of untrusted code [2]. In contrast to more traditional, sequential IFC systems, concurrent IFC systems must address covert channels—which arise due to concurrency—to be deemed secure. For example, LIO addresses the *internal timing covert channel*: a channel which allows a public thread to observe the timing behavior of a concurrent secret thread. Similarly, it addresses the *termination covert channel*: a channel which allows a public thread to observe whether a concurrent secret thread is still running.

Though LIO and other state-of-the-art concurrent IFC systems (e.g., [4]) eliminate these covert channels by construction, the underlying concurrency models do not account for threads running in *parallel*. To be truly useful, it is important for these systems to account for parallelism—this is especially the case for web servers which run on many-core CPUs. Unfortunately, running threads of varying sensitivity in parallel (e.g., on two cores) not only reintroduces the old covert channels, but also introduces new covert channels.

Though we believe these issues arise in most existing concurrent IFC systems, we focus on LIO as implemented atop the Haskell/GHC runtime for which we have running attacks.

Time-based attacks. Consider an LIO program running a public thread (p_0) on one core c_0 and, in parallel, two threads—one public (p_1) and one secret (s_1)—on another core c_1. In this setting, the secret thread can leverage the termination channel and collude with the public threads to leak secret data. Specifically, based on a secret, s_1 loops forever or terminates early. In doing so, the amount of time the runtime system will schedule p_1 on c_1 will either remain the same or double—a measurement that directly leaks the secret and is internally observable by the public threads. Alternatively, s_1 can employ a "fork bomb:" spawn n new threads on c_1, where n directly corresponds to a secret value. Here, again, the amount of time p_1 runs on c_1 (relative to an unperturbed p_0) will directly reveal the secret.

Space-based attacks. In addition to abusing runtime schedulers to leak information, in a parallel setting, memory management—the garbage collector (GC) in the case of LIO—can also be used to leak information. For example, s_1 can, based on a secret, exhaust all of memory (or not) to introduce a new termination channel. Alternatively, as observed in [5] for sequential programs, s_1 can force garbage collection on c_1 and directly affect p_1's run-time. As before, the public thread p_0 running on the other core can be used to race p_1 and leak the secret data internally to the program.

These covert channels exist because threads running in parallel share time (CPU time) and space (memory heap) which existing IFC models do not account for. Unsurprisingly, a secret thread can manipulate the state of these shared resources (e.g., via the scheduler or GC) and collude with public threads to leak sensitive data.

Unifying space and time. We propose a new design for a language runtime system that allows concurrent language-level IFC systems in the style of LIO [3] to run threads in parallel, on multiple cores. Our design is based on a key insight: since the label of a newly spawned thread must be at least as sensitive as the label of the parent [3, 6], i.e., a secret thread cannot create a public thread, we can safely dynamically partition both space and time hierarchically. We use this insight to create a hierarchical ownership model for resources that operate in space and time.

In our model, a thread must give up some of its resources to a child thread during thread creation. For example, when a thread spawns new threads, it relinquishes some of its resources—part of its CPU time and memory—to its children. Similarly, when these threads spawn new threads they will, in turn, give up part of their resources to their children, etc. By ensuring that each thread owns and executes in its own space-time continuum, a secret thread cannot affect public threads via the scheduler (e.g., by fork bombing) or garbage collector (e.g., by exhausting memory).

Since cleaning up memory is as crucial as not leaking it [1], our model also ensures that memory and CPU time can always be reclaimed. However, since reclaiming resources may leak information (e.g., whether a secret child thread terminated), our model imposes additional restrictions on the IFC system to prevent inadvertent flows.

We are in the process of implementing this model by extending LIO and its underlying Haskell runtime system, GHC. Specifically, we are **a)** extending the resource containers from [7] to hiearchically partition and reclaim thread heap space and **b)** modifying the GHC scheduler to implement a hierarchical scheduling algorithm that is synchronized across multiple CPU cores.

References

[1] S. Chong and A. C. Myers. End-to-end enforcement of erasure and declassification. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*. IEEE, 2008.

[2] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. Mitchell, and A. Russo. Hails: Protecting data privacy in untrusted web applications. In *Proceedings of the Symposium on Operating Systems Design and Implementation*. USENIX, 2012.

[3] S. Heule, D. Stefan, E. Z. Yang, J. C. Mitchell, and A. Russo. IFC inside: Retrofitting languages with dynamic information flow control. In *Proceedings of the Conference on Principles of Security and Trust*. Springer, 2015.

[4] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers. Fabric: A platform for secure distributed computation and storage. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009.

[5] M. V. Pedersen and A. Askarov. From trash to treasure: timing-sensitive garbage collection. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*. IEEE, 2017.

[6] D. Stefan, A. Russo, P. Buiras, A. Levy, J. C. Mitchell, and D. Mazières. Addressing covert termination and timing channels in concurrent information flow systems. In *International Conference on Functional Programming*. ACM SIGPLAN, 2012.

[7] E. Z. Yang and D. Mazières. Dynamic space limits for haskell. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014.