Timing Information Flow Control

Weiyi Wu, Jose M. Faleiro, Bryan Ford
Yale University

Timing side-channel attacks have been effective in compromising applications such as cryptosystems, which rely on highly confidential data. Moreover, the timing side-channel problem is particularly exacerbated in the cloud due to the abundance of parallelism and the scenario of sharing resources with an attacker. It is thus impertive to address the threat of timing side-channels as cloud computing gains popularity.

Decentralized information flow control techniques preserve privacy and integrity of data. Unfortunately, they offer no protection against leaking information through timing side-channels. A naïve approach involves enforcing *strict resource partitioning* techniques such as fixed-reservation timeslicing. However, such an approach undermines cloud providers' dependence on aggressive over-subscription and statistical multiplexing of infrastructure to provide attractive pricing to clients. Any feasible solution to the timing side-channel problem must be sensitive to this fundamental aspect of the cloud business model. We are exploring the use of a combination of deterministic execution and an extension of decentralized information flow control (DIFC) techniques to reduce the rate of information leakage via timing side-channels. Our technique preserves cloud providers' ability to multiplex resources aggresively among clients.

Deterministic execution guarantees that the execution of a job is dependent only on its *explicit inputs*. Internal timing side-channels are not part of an application's explicit inputs. Hence, a cloud providers' OS or VMM which offers deterministic execution eliminates the "dependence" of a malicious process' execution on internal timing side-channels.

However, the outputs of a system may be a source of timing information. An *external* malicious application, *E*, could monitor the outputs of a victim, *V*, within out deterministic sandbox. Since we have no control over *E*, it could use its own high-resolution clock to exploit the timing information obtained from *V's* output events. Such a scenario can be avoided by restricting an application's output to a small set of trusted applications over trusted, secure channels. As Facebook, Twitter and other public APIs demonstrate, software services are more often than not consumed by unknown clients over untrusted channels, rendering the above approach impractical.

Our approach, *Timing Information Flow Control*, or TIFC, uses a specification of *timing information flow policy*, akin to DIFC techniques. We are inspired by the HiStar system, and re-use its notation of assigning *labels* to system objects such as processes, messages and files. Unlike conventional DIFC however, TIFC labels reflect information that may have affected the timing of *observable events* associated with that object – a process starting or stopping, a message being sent or received, and so forth. We can specify safe timing information flow policy to disallow timing information from being exploited by an external malicious application. If information passes TIFC's label checks, we can be safe in the knowledge that it cannot be a useful source of timing information for a malicious application.

While specifying and enforcing timing information flow policy is important, we introduce an enhancement to TIFC's declassification capabilties. We introduce a mechanism to *control* the flow of timing information, which significantly enhances our system. We leverage *pacing queues* to implement such a mechanism. We assume we can pace the output of a pacing queue, such that regardless of how input builds up in the queue, the queues output releases at most one message per tick of a recurring timer, ring at a configurable frequency *f*. To see why pacing queues are a significant enhancement, consider the case of a malicious application spying on the output of a victim. If the victim's output were allowed to leave the system at arbitrary rates, the malicious application could discern highly confidential information by monitoring the timing of output messages. If we recognized this problem and used an appropriate label at the system's output interface, strict TIFC enforcement would not allow it to leave the system. However, many applications rely on such aribtrary external output, for instance, Facebook and Twitter's public APIs mentioned earlier. With the help of a pacing queue, we can conveniently *control* the rate at which such information leaves the system, so that it conforms to TIFC label checks.

We are currently in the process of implementing our ideas as an extension to the Determinator OS. Determinator constrains inter-process communication and synchronization of all processes to behave deterministically. We have extended the Determinator kernel to allow a process to communicate with arbitrary processes on other nodes while satisfying the constraints on the rate of information flow imposed by TIFC labels using pacing queues. We are yet to implement mechanisms for coping with external I/O such as reading and writing data to disk.

There are still a number of open issues regarding the practical use of our proposal. How do we reason about arbitrary compositions of distinct, timing-hardened applications? Moreover, due to the performance impact of pacing queues, performance debugging of a composition of applications may prove challenging. This is an important problem given the widespread coupling of cloud based software services to build applications on top of them. We do not yet have answers to these questions, and likely many more that will arise. However, our approach has the key properties of not unduly restricting applications and preserving the cloud business model. Hence, we believe we have a promising approach to mitigating the security risks of timing side-channels.