A Novel Class of Covert Channels enabled by Power Budget Sharing

S. Karen Khatamifard†, Longfei Wang, Amitabh Das, Selcuk Kose, Ulya R. Karpuzcu†
University of Rochester, †University of Minnesota

Runtime power management (PM) is in charge of optimal distribution of the power budget – a very critical shared resource – among system components. Any system-wide shared resource can give rise to covert communication, if not properly managed, and power budget unfortunately does not represent an exception. In this talk, we will demonstrate how easily inevitable sharing of the power budget can enable covert communication [2] and how the corresponding design space for countermeasures looks like.

Threat Model: Fig. 1 summarizes the threat model. Let us assume that a source application, which has access to sensitive information, shares processor resources with a sink application (possibly along with other applications). The source and the sink are malicious entities which are not permitted to communicate with each other *overtly*. The transmitting end (source) has access only to sensitive information; the receiving end (sink), only to an overt channel (to legitimately communicate with the outside world). By communicating with the source *covertly* – as enabled by resource sharing – the sink can not only get, but also distribute, this sensitive information. As a representative example, the source can be a contacts manager; and the sink, a weather application, in a mobile system [3].

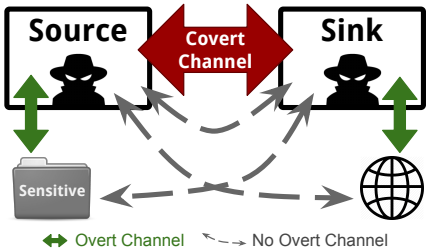


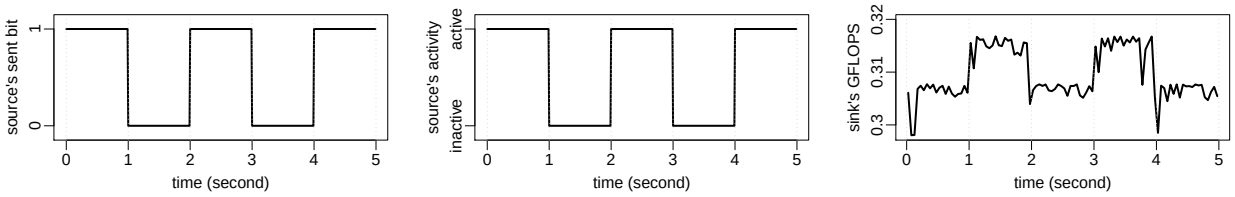
Figure 1: Threat model.

Covert Communication via Power Headroom Modulation (PHM): In distributing the power budget, PM must prevent budget overshoots (which can physically damage the system) while satisfying (possibly conflicting) performance requirements. Typically, a hierarchy of PM controllers coordinate to modulate the operating point (by, e.g., dynamic voltage frequency scaling, DVFS) periodically. In order to prevent budget overshoots, controllers also periodically monitor the impact of operating point adjustments on the instantaneous power consumption, by time-sampling activity monitors at regular intervals. The period of operating point adjustments, t_{PM}, is a function of the period of activity monitors along with the time it takes to perform the actual change in the operating point (which typically incurs the latency across the power/clock distribution networks and of voltage regulators [1]). The period t_{PM} is usually in the order of several processor clock cycles. By construction, the source and the sink are very well aware of this period. Moreover, typical PM algorithms are of predictive nature and extrapolate predictions from a history of monitored activity which are t_{PM} apart in time from each other.

If no other application resides in the system, but the source and the sink, the source can easily modulate the share of the power budget available to the sink, to encode binary information. This is possible simply because the chip-wide power budget is a shared resource. In the following, we will refer to the sink's share of the power budget as the *power headroom*. The source can easily control its own activity, and thereby, its own power consumption. Then, what is left to the sink is the power budget minus the source's power consumption, which forms the power headroom.

The procedure is straight-forward: To encode a logic 1, the source can run a power hungry virus. In this manner, the source can reach its own power budget limit, and taint its local activity history to trick the controllers. To prevent the instantaneous (system-wide) power consumption exceed the available budget as a result of excessive consumption at the source, the controllers will likely trigger emergency power throttling at the sink. The power headroom becomes practically zero. Under regular activity at the source (including no activity), on the other hand, such throttling events at the sink become much less likely. Therefore, by tracking the power headroom, the sink can clearly distinguish such extremely high activity from other activity levels at the source. If, for example, the source chooses to encode a logic 0 (1) by no (extreme) activity, the sink can decode a sizable positive power headroom as a logic 0, and any power headroom change (caused by source's activity) as a logic 1.

Other applications sharing the same processor resources can challenge this type of covert communication. Inevitably, sharing induces noise in the covert channel, which can reduce the correlation between distinct activity patters at the source (used to encode information by the source) and the power headroom (used to



(a) bits sent by source (b) source's activity (c) sink's performance (GFLOPS)

Figure 2: Example attack.

decode information by the sink).

Anatomy of the Attack: In modern power-limited comuting platforms of today, the power budget cannot accommodate all components operating at the peak performance point at the same time. Hence, when multiple cores run compute-intensive workloads simultaneously, PM has to assign a lower operating frequency to them than the rated peak frequency, in order to meet the power budget.

Usually, when only one compute-intensive application is running on one of the cores, PM lets that core run at the rated maximum frequency. The common outcome for two compute-intensive applications running at the same time on two different cores is a lower frequency than the rated maximum, enforced on both cores. Thereby, a given application's activity pattern can directly affect the performance of other applications running on the system. Applications like the source and the sink can thereby communicate with each other covertly, by affecting the operating frequency and/or voltage, hence the performance, of each other.

Fig. 2 illustrates an example: The source sends a "1" through the covert channel by running a compute-intensive workload, and a "0", by going into sleep. In order to capture source's activity pattern, the sink constantly runs a compute-intensive workload. As a result, PM slows down the sink when the source is sending a "1" (i.e., running a compute-intensive workload), compared to when the source is sending a "0" (i.e., going into sleep). The sink therefore can retrieve bits sent by the source by just tracking its own performance. To measure its own performance, the sink can simply periodically check its own progress. Neither the sink, nor the source does need any system level privilege to this end, which challenges detecting (and potentially blocking) the attack.

In this example, the source sends 5 bits through the covert channel at a communication rate of 1bit/s (the relatively low communication frequency here is to ease illustration), as Fig. 2a captures. The source becomes active when sending a "1", and goes to sleep otherwise. Fig. 2b shows source's activity pattern corresponding to the sent bits. Finally, Fig. 2c depicts sink's performance, as measured by the sink itself. Y-axis represents GFLOPS (Giga Floating Point Operations per Second); the x-axis, time. In this particular example the compute-intensive sink application comprises a floating point heavy loop, therefore, GFLOPS is a good proxy for checking the rate of forward-progress at the sink. We observe that the GFLOPS rate of the sink decreases by around 2.5% on average when the source is active (sending "1"), compared to when the source is sleeping (sending "0"). The sink can therefore retrieve information from this covert channel by simply checking its GFLOPS rate.

Initial Results: We characterized these channels on two representative platforms from industry featuring different microarchitectures and control hierarchies. We also experimented with multi-bit data encodings: i.e., how to encode information into four different levels of source's activity, enabling us to send two bits of information per channel use, instead of the simple 1-bit encoding using 0 and 1 only (as it is the case in the above example). We also considered various sharing and activity scenarios. Our analysis revealed a peak channel capacity of 121.6 bits per second (bps).

Countermeasures: One way to avoid such attacks is to assign a separate, fixed and safe power budget to each entity and thereby to exclude any power budget sharing. Another way that PM can limit the communication bandwidth is by imposing random noise on the GFLOPS signal, simply by adding random noise to the operating frequency. Yet another way to slow-down covert communication is by increasing PM's decision-making period t_{PM}. Unfortunately, all of these approaches can incur a significant degradation in power efficiency, which the PM is in charge of rectifying!

References

[1] P. Bose, , et al. Power management of multi-core chips: Challenges and pitfalls. *DATE*, 2012.

[2] S. K. Khatamifard et al. POWERT Channels: A Novel Class of Covert Communication Exploiting Power Management Vulnerabilities. *HPCA*, Feb 2019.

[3] H. Ritzdorf. Analyzing Covert Channels on Mobile Devices. *M.S. Thesis, ETH*, 2012.