Information Leakage in Camouflage and Comparison to Our System

Recent work by Zhou et al. introduces Camouflage [1], a memory traffic-shaping design that extends MITTS [2] to mitigate timing channel attacks and covert channel communication through memory controllers. In this report, we demonstrate that Camouflage cannot fundamentally bound the amount of information leaked through the memory request time series, and we describe an example covert channel that breaks Camouflage at bandwidths on the order of 10 Kbps.

1 Camouflage Design

Camouflage builds on top of MITTS [2], which allows memory controllers to throttle traffic to match a fixed distribution over time. The baseline MITTS design ensures that an application's memory traffic does not exceed a specified bandwidth distribution per *replenishment window*. This distribution is a histogram of latency between two consecutive memory requests, with `credits` assigned to each bin of the histogram. When a memory request misses in the per-core cache, the time delta between the current request and the previous request is calculated and is checked against the bandwidth histogram. If the corresponding bin has credits left in the histogram, the credit is deducted from the histogram and the request is issued to the uncore — the traffic shaper *drains* the histogram until credits remain. At the end of each replenishment window, the histogram is replenished to its original state.

Camouflage leverages the hardware proposed in MITTS to shape traffic as a means of thwarting side and covert channel attacks. To further obfuscate contention for the underlying resource, Camouflage introduces *fake memory requests* on behalf of workloads that do not fully drain a histogram during a replenishment window. At the end of a replenishment window, the remaining credits are stored and sent out during the subsequent time replenishment window. These fake memory requests are sent at a lower priority than the program's real memory requests to not interfere with a program's performance.

2 Fundamental Limitations of Camouflage

However, a significant limitation of the method in Camouflage is ignoring dependencies across time. The random variable X in Camouflage, which corresponds to inter-arrival times obtained from program traces, actually has strong time dependency: memory traces are well-known to have bursts and hence small inter-arrival times likely follow other small inter-arrival times. A sequence of time-dependent random variables is called a *stochastic process*. Therefore, Camouflage's actual problem consists of estimating entropy and mutual information of two stochastic processes $X(t), Y(t)$. This is a significantly more challenging problem compared to estimating the entropy and mutual information of two *scalar* random variables X, Y from independent samples (which is how Camouflage estimates information leakage) [1, 3, 4, 5].

The problem of estimating the entropy rates of stochastic processes has been studied extensively (see [3] Chapter 4 for an accessible introduction and [4, 5] and references therein for various algorithms used for different types of stochastic processes). Without making any assumptions on the type of memory in the stochastic process $X(t)$, estimating entropy requires computing histograms of all possible combinations of t values, which scales exponentially in the number of observed timeslots.

Camouflage ignores time dependencies and claims to prevent leakage only over long-term timing information for which, perhaps, time dependencies are not as strong. Unfortunately, the time-scale for which trace time dependencies are not significant depends on the application under test and is very hard to quantify. Worse, such time dependencies cannot be ignored if the application creates a *covert channel*. We use this time dependencies to construct a high bandwidth covert channel.

3 Covert Channel

Camouflage assumes that an adversary or malicious process cannot determine timing information within the *replenishment window* over which it shapes memory traffic [1]. Therefore, we create a covert channel where the communicating parties can see timing information over a time window *larger* than the replenishment window of Camouflage. We transmit one bit over every such *communication window* through a memory channel shaped by Camouflage. We additionally assert that malicious processes may learn the enforced distribution by profiling over the communication window since Camouflage must shape workloads to a fixed inter-arrival time distribution [6]. We assume communication windows on the order of 1,000 memory cycles [1] and a memory clock speed on the order of hundreds of MHz. Sending 1 bit per 2 communication windows creates a covert channel with bandwidth on the order of 10 Kbps.

While transmission parameters may be optimized further, we assume that the communication window is equal to two or more replenishment windows. A *transmitter* sends one bit by draining the learned histograms either partially or fully over the communication window and letting the receiver sense the difference. At the same time, a *receiver* attempts to drain its histograms fully during each communication window. The transmitter draining its histograms fully ensures that the receiver will perceive contention throughout the communication window, which it measures by observing the average latency of its memory requests. If the transmitter only partially drains its histograms, there will be less contention throughout the window, as Camouflage only introduces fake accesses retroactively in the next replenishment window. As the communication window encompasses multiple such replenishment windows, fake requests due to the transmitter partially draining its histograms will not be present during a subsequent communication window. The receiver may therefore determine a threshold on the latency it perceives to distinguish the bits being sent over the channel. This is described in further detail in Algorithm 1. After conveying a bit through the channel, the transmitter will *clear* the channel of Camouflage-introduced fake requests by fully draining the histogram during the next window.

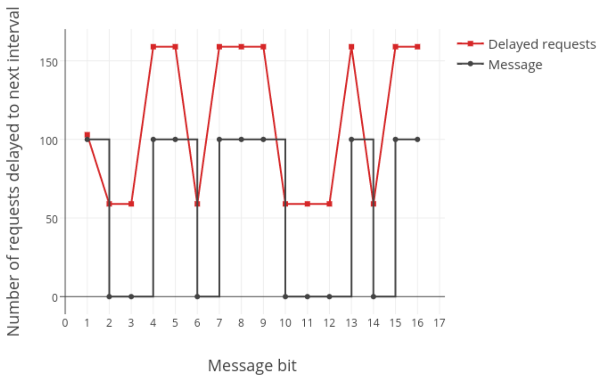

Figure 1: Contention over the *Camouflage*-obfuscated channel based on the number of memory requests sent during a communication window that were delayed to the subsequent window. The binary message being sent is overlayed on the change in contention perceived by the receiver. The bit being sent is clearly correlated with the contention, demonstrating the weakness of this shaping methodology.

The results of this methodology are shown in Figure 1 with the 16-bit binary string '1b1001101110001011' being sent across the channel. When the transmitter conveys the bit '1' during

Algorithm 1 Transmitter and receiver functions over a memory channel shaped by Camouflage.

```
function TRANSMIT(bit, histogram, window)
    cycle = 0
    while cycle < window do
        if bit and delta in histogram then
            sendMemReq ()
        else if not bit and delta in histogram / 2 then
            sendMemReq ()
        end if
        cycle++
    end while
end function

function RECEIVE(histogram, window)
    cycle = 0
    while cycle < window do
        if delta in histogram then
            sendMemReq ()
        end if
        cycle++
    end while
end function
```

a communication window, there is naturally contention over the channel as both the transmitter and the receiver send bursts of request simultaneously. Therefore, a simple thresholding algorithm may determine the value of a bit sent across the channel.

4 Comparison to STOPP-Q

In our report [7], we explicitly handle time dependencies and prove precise bounds on the amount of leaked information *even if the adversary has perfectly precise timing information at any resolution.* The key reason is the following: Camouflage is *draining* a histograms of requests and allowing credits from one replenishment window to affect an arbitrary number of following windows – this allows time dependencies from $X(t)$ to transfer to $Y(t)$. Hence, Camouflage fundamentally leaks information and is very hard to test or control for in an application. On the contrary, our approach is to learn a distribution for each time slot t in a program's execution, and then *sample* $Y(t)$ from these distributions. This allows us to generate a masked trace $Y(t)$ that is independent across time (i.e., is secure), but not identically distributed (and hence improves performance). By learning the statistics of $X(t)$ for each time t we ensure that our samples are quite close to the real trace, but privacy can be controlled by exploiting independence across time for the process $Y(t)$ that the adversary observes.

References

[1] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. Camouflage: Memory traffic shaping to mitigate timing attacks. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017*, HPCA '17, Piscataway, NJ, USA, 2017. IEEE Press.

[2] Yanqi Zhou and David Wentzlaff. Mitts: Memory inter-arrival time traffic shaping. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 532–544, Piscataway, NJ, USA, 2016. IEEE Press.

[3] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[4] Yun Gao, Ioannis Kontoyiannis, and Elie Bienenstock. Estimating the entropy of binary time series: Methodology, some theory and a simulation study. *Entropy*, 10(2):71–99, 2008.

[5] Yun Gao, Ioannis Kontoyiannis, and Elie Bienenstock. From the entropy to the statistical structure of spike trains. *CoRR*, abs/0710.4117, 2007.

[6] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. Understanding contention-based channels and using them for defense. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 639–650, Feb 2015.

[7] Isca2017 submission 448 technical report. `https://github.com/anonpapsub/QueuePrivacy/blob/master/techreport.pdf`, 2016.