Research Statement

Baris Kasikci, barisk@umich.edu, University of Michigan

My research is focused on developing techniques to build more reliable and secure systems. My current research initiatives consist of two key thrusts: (1) automated failure diagnosis for in-production software, (2) forensics and avoidance for speculative execution attacks. In this statement, I will detail my main research thrusts by first describing what progress I have made so far and then discussing my future plans. Finally, I will discuss how I will use the funding provided by Microsoft Research to pursue my research agenda.

Automated Failure Diagnosis for In-Production Software. Developers rely on reproducing failures for bug diagnosis. Once a developer can reproduce a failure, they can reason about the execution that led to the failure and determine the bug's root cause. Alas, reproducing in-production failures is very challenging.

Prior work. I have worked on multiple aspects of failure detection and diagnosis. I developed the first in-production data race detector (SOSP'13 [6]). I also developed a technique to classify data races based on their severity (ASPLOS'12 [5]), with higher accuracy than state of the art. I then designed failure sketching (SOSP'15 [4]), a technique that produces a high-level trace that includes statements leading to a failure. I also developed a technique for diagnosing concurrency bugs called Lazy Diagnosis (SOSP'17 [3]), where we observed that in many cases, events involved in a concurrency bug (e.g., shared memory accesses) are coarsely-interleaved in time. Therefore, one can rely on a lightweight mechanism to track the order of such events, which is crucial for low overhead diagnosis. More recently, I and my collaborators at MSR developed a tool that leverages our insight about coarse interleavings, called REPT (OSDI'18 [2]). Given the core dump and the control flow trace of a program before a failure, REPT partially reconstructs the failing execution (i.e., read/written data). Developers can then use the reconstructed execution for debugging.

Future work. There are many open research problems in failure diagnosis. I will specifically focus on two problems, one in the near term, and a more challenging second problem in the longer term. The *first problem* is one we face with our REPT line of work, where the accuracy of reconstructed executions drops with the increasing size of the control flow trace. This is not ideal, because the longer and more accurate the reconstructed execution is, the more useful it is for the developer. The drop in accuracy manifests as execution information gets overwritten (e.g., data values) and cannot be recovered. The broader *second problem* is failure diagnosis in distributed systems, which is very challenging due to the resource heterogeneity of the nodes (hardware support, computational power, etc.) as well as the scale of the networks (number of nodes, bandwidth, latency, etc.). Unfortunately, existing approaches are either tailored to specific systems or they require drastic modifications to existing systems.

1. To address the first problem, I will use capabilities in new processors to perform *targeted* data recording. I will use static analysis to identify points in an execution that are susceptible to data overwriting (e.g., a call to `realloc`). However, even with data recording, execution reconstruction will be limited. I will therefore explore the use of symbolic execution to generate missing information in reconstructed executions. Symbolic execution has some limitations, namely *path explosion* and *constraint solving complexity*. I will address path explosion by having symbolic execution follow a recorded control flow trace. However, because hardware tracing has limited capacity, I will develop annotation and system support for selectively enabling recording (e.g., to elide recording for logging, bookkeeping). To address the challenges of constraint solving, I will leverage data recording capabilities and coredumps. My preliminary results suggest that using core dumps for constraint simplification can provide up to three orders of magnitude speedup in constraint solving.

2. I will pursue the second problem in the long term. My goal is to extend statistical bug diagnosis techniques I developed for single-node systems [4, 3] to distributed systems. I will pursue a new approach influenced by the insights from prior work [7] that sought to create behavioral classes for a network of nodes in supercomputing systems. The main idea is that it should be easier to determine normal versus erroneous program characteristics for distributed systems if one can define *behavioral classes* of applications. I will perform behavioral classification based on high-level system semantics (e.g., key operations a system performs). This will allow more effective use statistical methods such as sampling, because information missing in the samples can be deduced based on the patterns in a class (e.g., only sampling *receive* and *verify* events in a system with a pattern such as *receive, verify, send*, and inferring properties about the *send* operation). Behavioral classification also allows leveraging compression for more effective use of resources (e.g., bandwidth), because traces in a behavioral class will be similar. Once an anomaly is isolated to a node

(or a subset of nodes), I plan to employ the techniques I will develop for single-node failure diagnosis.

Forensics and Avoidance for Speculative Execution Attacks. 2018 saw many high-impact microarchitectural side-channel attacks. Meltdown, Spectre, and Foreshadow demonstrated that microarchitectural state can have side effects that can be controlled and observed above the instruction set architecture abstraction. Attackers can leverage these side-effects (e.g., in the CPU cache) to bypass fundamental protection mechanisms such as memory isolation.

Prior work. I and my collaborators discovered and reported the Foreshadow [1] vulnerability to Intel in Jan 2018. Foreshadow demonstrated that it is possible to bypass the hardware-enforced confidentiality and integrity guarantees of Intel SGX. Further analysis by Intel revealed that techniques used in Foreshadow can be used to bypass the virtual memory abstraction by directly exposing cached physical memory contents to unprivileged applications and guest VMs.

Future work. Researchers demonstrate the feasibility of new speculative execution attacks by the day. I posit that we need to develop techniques and tools in the short term that will help us better understand the effects of these attacks. I also think that in the long term, we need to fundamentally rethink how to design systems that are resilient to such attacks. I plan to explore the following two projects:

1. I will focus on the problem of auditing the effects of a speculative execution attack once the attack has been discovered. Broadly, I seek to answer the following question: Given a new speculative execution attack, can we determine to what extent a system has been subject to the attack? I propose to investigate whether the attack pattern can be inferred by monitoring a program and gathering information at various levels of detail. The attack pattern is subtle for microarchitectural side-channels, because attacks' side-effects may not be observable above the instruction set architecture-level, or they may be transient (e.g., due to speculative memory accesses). I propose to initially study and understand the limitations of existing auditing capabilities (e.g., record/replay, checkpointing, logging). I will then augment existing capabilities to enable auditing the effects of speculative execution attacks. I will explore lightweight monitoring/tracing techniques that could always be turned on in production to facilitate auditing for future attacks.

2. Speculative execution attacks leak secrets using covert channels that connect the speculative world to the non-speculative world. Attempting to block covert channels is a ceaseless arms race: as soon as one covert channel is stopped, another channel is usually found. Instead, I argue that leaks must be stopped when secrets are accessed during speculation. Alas, determining what is a *secret* during speculation is prohibitively expensive. Instead, I propose to treat any data produced during speculation as a secret. In this scheme, speculation may only progress as long as it is not dependent on secret data. Depending on the anticipated threat model, this scheme can be implemented with varying performance and security tradeoffs. In this project, I will explore the wide design space of defenses against speculative execution attacks under different threat models.

Microsoft Funding A long-term strong collaboration with MSR is extremely important and valuable for me. I plan to use the funding from MSR to support two students for 3 academic semesters. I expect each student to complete the first project of each of the above thrusts by the end of the second semester. In each thrust, the second project partially builds on the first project. Therefore, the students will have made reasonable progress on the second project in each thrust by the end of the third semester. This will help me bootstrap my research agenda. The fellowship will allow my students to also have strong ties with MSR researchers and help me further strengthen my already-close ties with MSR.

References

[1] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *USENIX Security*, 2018.

[2] W. Cui, X. Ge, B. Kasikci, B. Niu, U. Sharma, R. Wang, and I. Yun. Rept: Reverse debugging of failures in deployed software. In *OSDI*, 2018.

[3] B. Kasikci, W. Cui, X. Ge, and B. Niu. Lazy diagnosis of in-production concurrency bugs. In *SOSP*, 2017.

[4] B. Kasikci, B. Schubert, C. Pereira, G. Pokam, and G. Candea. Failure sketching: A technique for automated root cause diagnosis of in-production failures. In *SOSP*, 2015.

[5] B. Kasikci, C. Zamfir, and G. Candea. Data Races vs. Data Race Bugs: Telling the Difference with Portend. In *ASPLOS*, 2012.

[6] B. Kasikci, C. Zamfir, and G. Candea. RaceMob: Crowdsourced data race detection. In *SOSP*, 2013.

[7] I. Laguna, T. Gamblin, B. R. de Supinski, S. Bagchi, G. Bronevetsky, D. H. Anh, M. Schulz, and B. Rountree. Large scale debugging of parallel tasks with automaded. In *SC*, 2011.