Presents:
“The Quest for Memory Safety”
with Dr. Hamed Okhravi
MIT Lincoln Laboratory

April 13, 2016
3:00-4:00 PM
32 Vassar Street, Stata Center (building 32)
Star, Room D463

Please RSVP for this no-cost event.
Register at: bit.ly/23k9k2g

Speaker Bio:
Dr. Hamed Okhravi is a research staff at the Cyber Analytics and Decision Systems group of MIT Lincoln Laboratory, where he leads programs and conducts research in the area of systems security. Currently, his research is focused on analyzing and developing system security defenses. He has served as a program committee member for a number of academic conferences and workshops including ACM Computer and Communications Security (CCS), Symposium on Research in Attacks, Intrusions, and Defenses (RAID), ACM Moving Target Defense (MTD), and ACM CCS SafeConfig Workshop. Dr. Okhravi earned his MS and PhD in electrical and computer engineering from University of Illinois at Urbana-Champaign in 2006 and 2010, respectively.

Abstract:
Memory corruption attacks have been a primary vector of cyber-attacks against computer systems for the past few decades. Complete memory safety techniques that provide spatial and temporal safety properties have been proposed in the community, but they incur large performance overhead to legacy languages such as C/C++. As a result, there has been a race in the community to create lightweight, compatible, and effective memory corruption defenses. In this talk, we evaluate two such defensive paradigms called Code Pointer Integrity (CPI) and Control Flow Integrity (CFI). We show that an attacker can bypass CPI’s enforcement mechanism using information leakage attacks. We also show that the inaccuracies of static analysis make CFI bypassable in practice, and demonstrate attacks against real-world applications. Further, we build an automated tool to find such vulnerabilities, and evaluate the exposure of popular applications to CFI bypasses. Finally, we describe a lightweight defense that mitigates the impact of information leakage attacks by frequently re-randomizing the layout of memory at runtime. Our evaluations on standard benchmarks indicate that runtime re-randomization incurs a low performance overhead (~2% on average).