POSTER: Towards Secure Execution of Untrusted Code for Mobile Edge-Clouds

Jiaqi Tan
Carnegie Mellon University
tanjiaqi@cmu.edu

Utsav Drolia
Carnegie Mellon University
utsav@cmu.edu

Rajeev Gandhi
Carnegie Mellon University
rgandhi@ece.cmu.edu

Priya Narasimhan
Carnegie Mellon University
priya@cs.cmu.edu

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Program Verification—*Safety Properties*

Keywords

Mobile Edge-Clouds, Safety Properties, Theorem Proving

1. INTRODUCTION

Mobile personal devices such as smartphones and tablets are ubiquitous today, and they are growing in storage, compute, and sensing capabilities. Collectively, these mobile devices in close physical proximity present a rich pool of both compute/storage resources and personal data. Mobile edge-clouds are clouds comprised entirely of mobile nodes in close physical proximity without any infrastructure support such as back-end compute clouds [1]. Mobile nodes serve as both the compute nodes, and the source of data for mobile edge-clouds. Mobile edge-clouds allow the compute/storage resources and data stored across multiple mobile devices to be pooled to form a single compute resource, and they enable applications across independent mobile devices, particularly when high-bandwidth, low-latency connections to the Internet may be degraded (e.g. in massive crowds in stadiums), or unavailable (e.g. during disaster response). However, a key security risk which may prevent users from participating in mobile edge-clouds is that their mobile devices need to execute code from other untrusted edge-cloud nodes [5]. Hence, we propose a system which allows nodes in a mobile edge-cloud to securely execute code from untrusted clients.

Example. Consider a "person-finder" application, where a client device desires photos from nearby edge-cloud nodes containing the face of a given person. The client device can pack the face of the target person with the code for a face recognition algorithm, and send the code to edge-cloud nodes. The edge-cloud nodes will then run the provided code on their stored photos, returning the matching photos to the client. Thus, nearby devices will send only matching photos to the client, eliminating the need for devices to send all their data to a central location for processing.

Assumptions and Threat Model. We call the providers of compute/storage resources and data the nodes of a mobile edge-cloud, and we call the initiators of computation clients. We assume that clients and nodes are mutually distrusting, and that clients and nodes have no prior knowledge of the identities of other nodes or clients. Clients can submit arbitrary processing tasks in the form of machine code. These tasks can be malicious, and may try to crash or alter the nodes they run on. We assume that nodes in the edge-cloud will faithfully execute client-submitted code, and we focus on the security of nodes running untrusted client code.

Proposed System. We propose a system for nodes in a mobile edge-cloud to securely execute untrusted code submitted by clients. We require client code to obey memory and control-flow safety properties which will isolate edge-cloud nodes from the client code. Our system will use theorem proving techniques to automatically prove that our desired safety properties hold in the submitted machine code. *Specifically, we apply theorem proving to machine code programs for ARM processors, which are widely used on mobile devices.* Thus, we enable untrusted clients to submit code to nodes in our mobile edge-cloud without code vetting or signing by a central authority. This provides great versatility, as clients can submit arbitrary tasks to nodes as needed.

Contributions. Our planned contributions are: (1) Develop a novel method to automatically generate a concrete safety policy comprising memory and control-flow safety properties sufficient to isolate untrusted user programs on mobile edge-cloud nodes. (2) Automatically prove theorems of conformance to a safety policy on unmodified machine code programs for a mobile platform (ARM Linux binaries). (3) Develop novel formalizations for reasoning about the effects of system calls as observed by user programs. (4) Develop a novel toolchain which passes information from theorem proving to a compiler to help developers write code which provably meets our desired safety policy. Thus, programmers will not need to write assertions or proofs.

2. DESIGN

Overall Approach. We plan to implement our secure edge-cloud execution system for ARM devices running Linux, and our system will allow clients to submit tasks as user programs in the form of binaries for execution by edge-cloud nodes. Then, we use theorem proving to prove that de-

WiSec'14, July 23–25, 2014, Oxford, United Kingdom.
ACM 978-1-4503-2972-9/14/07.

sired safety properties hold in the code. Client-submitted code must meet our desired safety properties, or the proof process will fail, and edge-cloud nodes will not execute the code. Concretely, these tasks are ARM Linux binary programs, and we rely on the operating system to provide isolation between the OS and user programs, and between the edge-cloud tasks and other user programs.

As a proof-of-concept, we currently implement the theorem proving process for safety properties on a desktop environment, and we envision running this process on mobile devices in future. We implement the proving of safety properties for ARM machine code programs. We prove safety properties in machine code, rather than in the source code of high-level languages such as C, so that we do not need to include the compiler in the Trusted Computing Base (TCB). We work with machine code rather than virtual machine bytecode (e.g. Dalvik bytecode) so that we can exclude the virtual machine from the TCB.

Memory and Control-Flow Safety. We would like edge-cloud tasks to have memory and control-flow safety. We require that all memory read and write instructions have memory address targets restricted to the user-addressable parts of the virtual memory address space, i.e. the program stack and heap areas, and data and constants (`bss`) sections. We also require that all jump targets are restricted to addresses in the text section of the binary of the user-level task. We disallow edge-cloud tasks from calling code that is not present in the task binary, effectively isolating the task. This also implies that user tasks must be statically linked. User tasks are also not allowed to make any calls to functions in shared libraries. Additionally, to prevent stack-smashing attacks, we also require that target addresses of memory writes do not alter return addresses in relevant registers.

Mechanization and Proof Automation. To formally prove that our desired safety properties hold in the machine code of client-submitted tasks, we use a validated formalization of ARM machine code instructions [2, 3]. We use the HOL4 [4] proof assistant to carry out our proofs. We decompile ARM machine code to obtain Hoare triple theorems for each instruction [3], which describe the machine state before and after each instruction. Then, we augment these theorems to include assertions that our memory and control-flow safety properties hold. Next, we plan to use a logic rule to syntactically indicate that the single-instruction theorems have been strengthened with our memory and control-flow safety assertions, inspired by ARMor [6]. Then, we plan to use hierarchical judgments [6] to compose the per-instruction safety theorems to obtain a whole-program safety theorem. The novelty of our proof process is that we aim to perform the safety proof without requiring an explicit user-supplied safety policy, as was required by ARMor [6].

Programmer Assistance. As we require that edge-cloud clients must submit code which meets the safety properties desired by edge-cloud nodes, developers of mobile edge-cloud tasks must write code which meets these safety properties. We plan to build a toolchain to help developers write code which meets these safety properties using our theorem proving process. This toolchain will try to infer, from the locations of proof failures in the machine code, the corresponding source-code statements which are responsible for the proof failures. Then, the toolchain will suggest locations in the code where safety checks (e.g. array bounds checks, input sanitization) can help meet the safety properties.

3. RELATED WORK

Our system builds on the formalization developed by Myreen and Fox [2, 3], and the logic rules and proof style developed by Zhao et al. [6]. Like ARMor [6], we prove that safety properties hold in machine code programs. A key difference is that ARMor is designed for embedded programs running directly on hardware, while we aim to prove safety properties in user programs running in the presence of an OS. Also, ARMor first uses binary rewriting to insert safety checks, while we operate on unmodified user programs, and assist developers in writing safe programs based on information from failed safety property proof attempts.

4. INITIAL RESULTS

We have identified compiler and linker options necessary for compiling C programs statically while minimizing the number of system library (`libc`) functions linked. We used a custom linker script to bypass the default `libc` initialization functions and pick a custom entry-point to the user code. Currently, we also aim to minimize the number of system calls invoked to simplify the proof process, and we have limited our compiled programs to a single system call (`_exit`). We have also successfully decompiled ARM machine code programs to Hoare triple theorems [3], and we are currently in the process of strengthening these theorems with safety assertion predicates, and introducing syntactic tags [6] to indicate the presence of these safety assertions.

5. CHALLENGES AND FUTURE WORK

First, our current safety property proof process is unable to handle system calls. We plan to build an initial formalization of the effects of system calls as observed by user programs to help us prove safety properties in the presence of system calls. Second, we need to implement the proof mechanization process to help us automatically prove safety properties, as proof assistants typically require user interaction. Third, we intend to develop our toolchain to help developers write code which provably meets the desired safety properties of our mobile edge-clouds. Finally, we intend to implement the proof process on mobile platforms such as Android, so that mobile nodes can independently prove the safety properties of client code.

6. REFERENCES

[1] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan. The Case for Mobile Edge Clouds. In *IEEE Ubiquitous Intelligence and Computing (UIC)*, Dec 2013.

[2] A. Fox. Formal specification and verification of ARM6. In *International Conference in Theorem Proving in Higher Order Logics (TPHOLs)*, 2003.

[3] M. Myreen, A. Fox, and M. Gordon. Hoare Logic for ARM Machine Code. In *Fundamentals of Software Engineering (FSEN)*, 2007.

[4] K. Slind and M. Norrish. A Brief Overview of HOL4. In *International Conference in Theorem Proving in Higher Order Logics (TPHOLs)*, 2008.

[5] J. Tan, R. Gandhi, and P. Narasimhan. Challenges in Security and Privacy for Mobile Edge-Clouds. Technical Report CMU-PDL-13-113, Oct 2013.

[6] L. Zhao, G. Li, B. D. Sutter, and J. Regehr. ARMor: Fully Verified Software Fault Isolation. In *International Conference on Embedded Software (EMSOFT)*, 2011.