Capabilities Statement – ReSCIND GrammaTech Inc.

GrammaTech has an extensive background in the area of cybersecurity and defense, as well as specific experience related to the ReSCIND problem statement in terms of systems modeling, monitoring, and mitigation. This document briefly highlights two recent projects in these areas, specifically an autonomic monitoring and mitigation framework and a system for modeling and verifying system-level security properties. Please direct any questions or teaming inquiries to zfry@grammatech.com.

Monitoring and Mitigation

Runtime monitoring for anomalous behavior of software systems is a critical validation technique as part of defense-in-depth cybersecurity solutions. Such techniques compliment static analyses and runtime pattern-matching based approaches that only identify known attacks and are thus likely to miss unknown or 0-day attacks. There exist many policy verification languages to support runtime validation and anomaly detection, but each is often tailored to a specific domain, making it difficult to express policies that embody multiple verification methods for a variety of problem areas. The specificity of these languages can often make it difficult for non-experts to understand, create and modify policies. Furthermore, the associated monitoring engines are often specific to a narrow set of programs and runtime environments, limiting broad applicability.

We introduce a new language, Tiffin, designed to express widely-scoped program behavior specifications as enforceable runtime policies. We also introduce a toolset that uses Tiffin policies as input to produce concrete application monitors encoding the developed policies. These monitors check adherence to the specification and respond appropriately when policies are violated. Tiffin is designed to express a wide variety of program behavior characteristics from extended finite automata and invariants, to enforcing interfaces and memory access constraints as well as guiding component-based fuzzing. The developed toolset consists of a policy compiler, mgen, with multiple backends, that can produce monitors in the form of application-level dynamic translation wrappers for monitoring specific standalone targets; hypervisors that can monitor firmware or operating systems; or binary rewriters that can permanently weave a policy into an application. Together, the developed technologies aim to provide autonomous runtime validation for a variety of program and deployment types that is easy to specify and deploy, even by non-experts. This presentation specifically describes use cases for application-level policy enforcement, firmware-level policy violation detection and response, and how said policies can be used to identify bugs by fuzzing components to detect policy violations.

This material is based upon work supported by the Office of Naval Research and the Air Force Research Laboratory under contracts N68335-19-C-0200, FA8650-20-C-1106, and FA8650-17-F-1056. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research or the Air Force Research Laboratory.

System Modeling and Validation

Modern software systems, ranging from home networks to nation-critical infrastructures, are assembled from general COTS and open-source components. These components are built to be versatile - usable in a variety of different contexts. For a system to accomplish its mission, its components must be properly

configured to interoperate. However, system configuration is complicated in practice: local configuration changes often have a profound system-wide effect. Misconfiguration can both hamper system functionality and have severe security repercussions.

Formal Methods enable users to systematically reason about the global configuration of a system - to detect and diagnose cases where the configuration needs to be tightened to improve security or relaxed to enable the desired functionality. A few techniques for formally modeling various configuration aspects have been proposed over the last several decades. The most recent surge of research activity focuses on the configuration of cloud applications. Existing techniques vary in scope and level of detail (e.g., range from detailed modeling of configuration for a single Apache web server instance to high-level modeling of network configuration in a large enterprise network). Each technique only supports a narrow set of configuration surfaces and employs custom semantics definitions for system primitives, making it difficult to extend or combine these techniques for unified, system-wide configuration reasoning.

We propose an extensible modeling framework that decouples the modeling of implementation-specific system aspects from backend analyses. Central to our approach is an extensible library of reusable system-building primitives, such as, Linux user, Windows group, Ethernet network interface, and OpenSSH service. Models of arbitrarily complex systems can be automatically constructed by stitching together these building blocks. We designed a formal language that uses general programming concepts, such as abstract data types, strict static typing, and hierarchical composition of components to effectively build low-level library primitives and capture complex system concepts, such as network access, service availability, and remote system and file access. Our approach offers the following advantages:

Our language and modeling methodology allows users to effectively extend the library with support for additional systems and services. Given the multitude of software components and services actively used in real-world systems and the pace at which new technologies are developed, easy extensibility is paramount.

Client analyses have a uniform and extensive query language for checking system properties. This allows our modeling framework to be employed in many important application areas, including configuration management, penetration testing, red teaming, and forensic analysis.

The semantics of low-level system primitives (e.g., Linux directory) and high-level concepts (e.g., remote file access) is uniformly captured as a deductive database (a database system that stores a set of facts and uses a set of well- defined rules to derive additional facts from stored facts), allowing for an effective query evaluation using either existing off-the-shelf solvers or newly developed mechanisms.

The current version of our library supports a subset of Linux file access controls, remote login services, simplified web services, and basic networking. We have built models for several notional and real-world systems with sufficient accuracy to detect and diagnose non-trivial security issues, such as non-authorized system access due to an exposed private key and a sensitive information leak due to a symbolic link misuse. To provide a sense of scale, a model we auto-constructed for a system with 20 devices connected to half a dozen of subnets has 2K configuration parameters, 4K facts, and over 200 rules. Configuration parameters include firewall-rule lists, file ownership and permissions for selected files, a user-to-group assignment, a subset of SSH configuration options, etc. To query the model, we compile it into a Prolog program and use an off-the-shelf prolog interpreter (SWI-Prolog). We chose this approach to enable rapid prototyping. Much more efficient query evaluation mechanisms can be

adopted in practice, such as stratified datalog solvers (e.g., Souffle) or SMT solvers (e.g., Z3). However, even our current solution takes under 2 minutes to evaluate all stated access-control requirements for the system described above (10K individual queries).

Acknowledgements. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.