Enabling User-Defined Security Policies for Programmable IoT Systems
Samuel Berkun, Dr. Omar Chowdhury, Moosa Yahyazadeh

Introduction

• People use Internet of Things (IoT) systems to control their smart devices, in their smart homes
• IoT platforms rely on downloading third-party apps, which may be malicious
• PATRIOT (Policy Assisted Resiliency for IoT) ensures safety by filtering action requests in IoT systems

About PATRIOT Framework

• When applications want to control devices, they sent action requests to the IoT platform. For example, an app might send a request that says “Open the bedroom window”.
• PATRIOT runs on the IoT platform, where it controls the flow of requests. For each request, PATRIOT either allows or denies the request. This decision is based on the security policies defined by the user.
• Users can define policies directly in language, or use the Graphical User Interface (GUI) to automatically generate them.

Language and GUI Design

• Since PATRIOT is meant to be usable by everyone, language and GUI are designed to be as simple and intuitive as possible
• Language is high-level (close to English) and meant to translate from intuitive user expectations
• GUI consists primarily of selection menus, to make it simple to create invalid syntax
• GUI developed as a web interface using HTML/CSS and Bootstrap framework

Acknowledgments

I am grateful to everyone in the Computational Logic Center at the University of Iowa for welcoming me to their lab. I would like to thank Dr. Omar Chowdhury and Moosa Yahyazadeh for bringing me onto the project, as well as taking the time to explain everything to me and making sure I was comfortable with my work. Finally, I would like to thank the Belin-Blank Center for providing me this opportunity.

Traces and Policy Analysis

• Conditions in the PATRIOT language can have temporal formulas - essentially, filtering depends on past events as well as present
• PATRIOT treats the past as a sequence of states, which is called a trace. A state is stored whenever an action occurs. See Figure 3 for an example.
• Occasionally, it may be possible for the system to reach a point where it gets stuck: The policies are defined such that with the current trace, no actions are allowed.
• The PATRIOT policy analysis warns the user if the user-defined policies may cause the system to get stuck in the future
• Analysis is done by converting policies into SMT formulas, which are processed by the Z3 SMT solver.

SMT Solving

• SMT solvers, such as Z3, check whether a set of first-order logic formulas are satisfiable. See Figure 5 for an example of this.
• Each policy is essentially a formula of the form “action implies condition”

Conclusions and Future Work

• Use of PATRIOT language structured policies allows avoiding many issues with previous implementations
• Analysis can be expanded and improved in efficiency: Alternate definitions of “stuck” trace
• Adoption of system into smart home platforms; improving usability

References


Figure 1: PATRIOT system in action. The malicious application is trying to unlock the front door, but is unable to do so because the user has defined a security policy that restricts access to the front door lock. Figure 2 gives an example of how this policy can be defined.

Figure 2: A policy, defined in the PATRIOT language.

Figure 3: An example execution trace, being evaluated with respect to the current state.

Figure 4: A screenshot of the GUI developed for PATRIOT.

Figure 5: An example of satisfiable and unsatisfiable sets of formulas.

Figure 6: An example of a policy being converted into a sub-formula.