Poster Abstract: BUFS: Towards Bottom-Up Foundational Security for Software in the Internet-of-Things

Jiaqi Tan, Rajeev Gandhi, Priya Narasimhan
Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213
Email: tanjiaqi@cmu.edu, rgandhi@ece.cmu.edu, priya@cs.cmu.edu

I. Introduction

The Internet-of-Things (IoT) is a rapidly growing phenomenon. While IoT-enabled objects can provide rich features that can improve users' lives, security failures can lead to severe consequences, particularly in safety-critical domains such as medical devices and automobiles. In addition, IoT-enabled objects are often connected to the Internet, increasing their risk for external attacks. Thus, it is important for IoT systems to have strong security guarantees. Some of the security challenges IoT systems face include the need for lightweight cryptographic algorithms and secure communications protocols. In practice, security mechanisms are implemented in a software stack on IoT devices. This software stack needs to (i) provide security mechanisms correctly, and (ii) faithfully execute application logic, without being circumvented by attackers. Software vulnerabilities may allow external attackers to circumvent these security measures: over 250 vulnerabilities were discovered in the top 10 IoT devices in use today in a recent study [1].

We propose BUFS, a bottom-up and foundational approach for verifying the security of the software stack in an IoT system, to provide guarantees for how the software is secure. BUFS is a *secure-by-construction* approach that verifies that IoT software is secure in a bottom-up and foundational way. By bottom-up, we mean that the security of software needs to be established at every level of abstraction: at the OS, implementation, and functional design levels. By foundational, we mean that the security of IoT software should be verified using formal techniques, and that the artifacts that are verified should be the actual ones in use, rather than high-level designs. The BUFS approach provides tools for aiding and automating parts of the software development and verification process for programmers, and is intended for safety-critical domains, where high-assurance is required. Current techniques for achieving high-assurance in software are piecemeal and not bottom-up: they verify security properties at single levels of abstraction (e.g., verifying the correctness of a single protocol implementation [2] while assuming implementation-level security properties such as Control-Flow Integrity (CFI) [3]). In BUFS, we plan to verify the security of the actual (source and machine) code that will run on IoT devices.

II. Threat Model

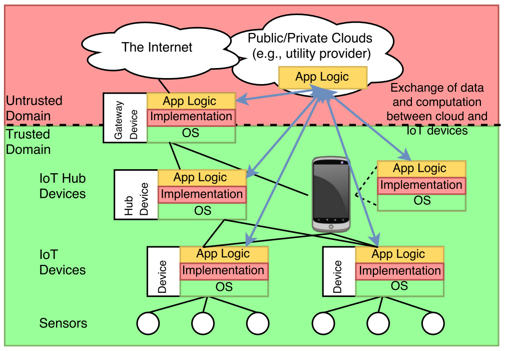

Figure 1. Threat Model: Green = trusted domain; Yellow = can be secured with other techniques; Red = untrusted components we need to secure. Trusted and untrusted components may communicate with each other.

Figure 1 illustrates our threat model for a typical IoT system, which consists of: (i) a trusted domain with sensors and IoT devices under a single administrative domain (e.g., in a single home, or in a single automobile), and (ii) an untrusted domain connected via the public Internet, including any cloud-based provider. We assume that physical security holds in the trusted domain, where: (i) IoT devices connect to sensors (e.g., embedded devices), (ii) IoT hub devices consolidate data from other IoT devices, e.g., smartphones, and may form a compute fog [4] or edge-cloud [5], and (iii) the gateway device provides connectivity to the Internet. The IoT software stack consists of the OS, the implementation of the application logic, and the application logic itself. Note that we consider the application logic to be separate from its implementation: while the design of the application logic can be separately verified, vulnerabilities can still be introduced due to implementation bugs. We assume the OS is secure, as there are verified microkernels [6] we can leverage.

While IoT hub devices and IoT devices are trusted in our threat model, IoT systems are likely to interact with the cloud (e.g., utility company or software provider). Even if IoT devices are not addressable externally (e.g., they have local IP addresses), the software on IoT devices may receive commands from application logic in the cloud. These inputs may be malicious or malformed, which may allow IoT applications in the trusted domain to be hijacked.

III. Bottom-up Software Security

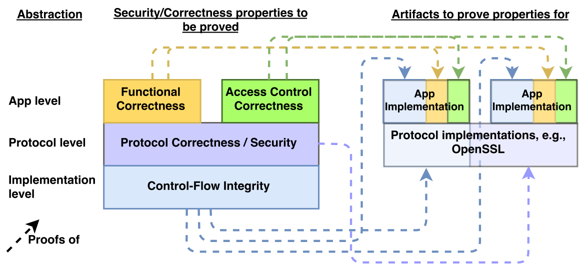
Abstraction
Security/Correctness properties to be proved
Artifacts to prove properties for
App level
Protocol level
Implementation level
Functional Correctness
Access Control Correctness
Protocol Correctness / Security
Control-Flow Integrity
App Implementation
App Implementation
Protocol implementations, e.g., OpenSSL
Proofs of

Figure 2. Bottom-up construction of security properties (left), and how these properties are proved in the software artifacts (right).

Figure 2 illustrates the bottom-up aspect of our BUFS approach, with examples of security properties at each level of abstraction. We break down the high-level goal of software security for IoT into specific security properties at each level of abstraction. In this example, at the implementation level, the machine-code needs to have Control-Flow Integrity [3] so that the software's execution cannot be circumvented by malicious or malformed inputs. At the protocol level, implementations of secure communications protocols, such as SSL, must behave according to the protocol specification (assuming that the protocol is secure). Finally, at the application level, the application and its access control mechanisms must correctly perform their intended functions.

Proofs of security for protocol implementations comprise of both proofs of implementation-level properties such as CFI, as well as protocol correctness. Proofs of security for application code must comprise implementation-level properties, and functional and access control correctness. Thus, BUFS ensures that the security guarantees at each level of abstraction are built up in a bottom-up manner.

IV. Foundational and Compositional Security

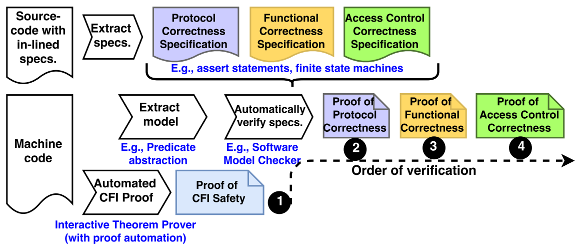
Source-code with in-lined specs.
Extract specs.
Protocol Correctness Specification
Functional Correctness Specification
Access Control Correctness Specification
E.g., assert statements, finite state machines
Machine code
Extract model
Automatically verify specs.
Proof of Protocol Correctness
Proof of Functional Correctness
Proof of Access Control Correctness
E.g., Predicate abstraction
E.g., Software Model Checker
2
3
4
Order of verification
Automated CFI Proof
Proof of CFI Safety
1
Interactive Theorem Prover (with proof automation)

Figure 3. Achieving bottom-up security foundationally from software artifacts ("Specs" = specifications).

Figure 3 illustrates the foundational aspect of our BUFS approach. First, at the implementation level, automated proofs of CFI are generated given the machine-code of the software using interactive theorem proving (ITP), augmented with proof automation. Then, at the protocol and application levels, specifications of properties such as protocol correctness, functional correctness of the application logic, and correctness of applications' access control mechanisms, are extracted from the software artifact, such as via in-lined assertion statements, or constructed manually using models such as finite state machines. Next, a model of the software's behavior is extracted from its source-code, and techniques such as software model checking (SMC) are used to verify that the software's specifications are met by its source-code implementation. BUFS aims to provide tools at the implementation-level for automated CFI proofs of machine-code via ITP, and tools at the source-code level for model extraction and property verification via SMC.

V. Initial Results and Future Work

We have developed a logic framework for automatically proving safety properties [7] such as CFI in ARM machine-code using the HOL4 theorem prover that also supports realistic embedded software features e.g., hardware I/O and system calls [8]. We have also developed a technique for enforcing CFI using source-code safety-checks [8] that can be proved automatically using our logic framework.

In future, we plan to: (i) develop techniques for programmers to specify program properties such as functional correctness and access control correctness in their source-code for automatic extraction, and (ii) adapt current SMC verification techniques for programs with source-code specifications inserted using our technique. We plan to demonstrate the foundational aspect of BUFS by verifying our security properties in a bottom-up way on the actual deployed source-code/machine-code artifacts of IoT software.

References

[1] "HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack," 2014, http://bit.ly/2943Cyp.

[2] S. Chaki and A. Datta, "ASPIER: An Automated Framework for Verifying Security Protocol Implementations," in *IEEE CSF*, 2009.

[3] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow Integrity," in *ACM CCS*, 2005.

[4] F. Bonomi et al., "Fog Computing and Its Role in the Internet of Things," in *ACM SIGCOMM MCC*, 2012.

[5] U. Drolia et al., "The Case for Mobile Edge Clouds," in *IEEE UIC*, Dec 2013.

[6] G. Klein et al., "seL4: Formal verification of an OS kernel," in *SOSP*, Oct 2009.

[7] J. Tan, H. Tay, R. Gandhi, and P. Narasimhan, "AUSPICE: Automatic Safety Property Verification for Unmodified Executables," in *VSTTE*, 2015.

[8] J. Tan, H. Tay, U. Drolia, R. Gandhi, and P. Narasimhan, "PCFIRE: Towards Provable Preventative Control-Flow Integrity Enforcement for Realistic Embedded Software," in *EMSOFT*, 2016.