Static Analysis for the OWASP IoT Top 10 2018

Pietro Ferrara[1], Amit Kr Mandal[2,3], Agostino Cortesi[2], and Fausto Spoto[4]

[1] JuliaSoft Srl, Verona, Italy
pietro.ferrara@juliasoft.com
[2] Università Ca' Foscari, Venezia, Italy
cortesi@unive.it, amitkr.mandal@unive.it
[3] BML Munjal Univesity, Gurgaon, Haryana, India
amitmandal.nitdgp@gmail.com
[4] Università di Verona, Verona, Italy
fausto.spoto@univr.it

Static Analysis and OWASP Top 10 2017

Static analysis detects bugs at compile time without executing the code. While dynamic analysis (*e.g.*, testing) needs specific execution states in order to expose different execution paths, static analysis can abstractly reason about all different paths of execution. However, it needs to introduce some form of approximation in order to represent the execution of a program and prove properties on them. In particular, static analysis tools build a semantic model of a software at compile time without executing it, and then check various properties on that model. Nowadays, several standards and regulations (*e.g.*, MISRA DO-178C, IEC 61508, ISO 26262, and IEC 62304) require the application of such tools.

Static analysis has been already widely applied to detect security vulnerabilities on the back-end of web servers [1, 2] and, in particular, to several OWASP Top 10 2017 categories. Figure 1 reports the OWASP Top 10 2017 categories, and which ones can be (partially) addressed by means of static analysis, since they concern the software implementation.

	Title	
A1	Injection	✓
A2	Broken Authentication	✓
A3	Sensitive Data Exposure	✓
A4	XML External Entities (XXE)	✓
A5	Broken Access Control	
A6	Security Misconfiguration	
A7	Cross-Site Scripting (XSS)	✓
A8	Insecure Deserialization	
A9	Using Components with Known Vulnerabilities	✓
A10	Insufficient Logging&Monitoring	

Fig. 1. OWASP Top 10 2017.

In particular, A1 (Injection), A3 (Sensitive Data Exposure), A4 (XXE) and A7 (XSS) can be detected through taint analysis [3], that is, an analysis that tries to detect if a value coming from a source (*e.g.*, methods retrieving some user input) flows into a sink (*e.g.*, methods executing SQL queries) without being sanitized (*e.g.*, properly escaped). Instead specific tools such as the OWASP Dependency Check[1] have been developed to detect A9 (Using Components with Known Vulnerabilities). Finally, static analysis was

[1] https://www.owasp.org/index.php/OWASP_Dependency_Check

able to partially address A2 (Broken Authentication) when a program adopts weak or broken cryptographic algorithms, or contains hardcoded passwords.

OWASP IoT Top 10

Few months ago, the OWASP IoT Top 10 2018 categories were released. Figure 2 reports them. As for the OWASP Top 10 2017 categories, that figure reports the categories (partially) detected by static analysis.

First of all, some IoT Top 10 categories are quite similar, if not identical, to some of the Top 10 categories. In particular, I5 (Use of Insecure or Outdated Components) is equivalent to A9, I1 (Weak Guessable, or Hardcoded Passwords) is part of A2, and I6 (Insufficient Privacy Protection) partially corresponds to A3. Instead, I3 (Insecure Ecosystem Interfaces) is defined as follows: "Insecure web, backend API, cloud or mobile interfaces in the ecosystem outside the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering". This comprises many different OWASP Top 10 2017 categories (A1, A3 and A7 among those addressed by static analyzers).

	Title	
I1	Weak Guessable, or Hardcoded Passwords	✓
I2	Insecure Network Services	
I3	Insecure Ecosystem Interfaces	✓
I4	Lack of Secure Update Mechanism	
I5	Use of Insecure or Outdated Components	✓
I6	Insufficient Privacy Protection	✓
I7	Insecure Data Transfer and Storage	
I8	Lack of Device Management	
I9	Insecure Default Settings	
I10	Lack of Physical Hardening	

Fig. 2. OWASP IoT Top 10 2018.

Open Challenges and Opportunities

Since static analysis has been applied to various OWASP Top 10 2017 categories, and many IoT Top 10 2018 categories are equivalent or similar to those, could we conclude that the application of static analysis to detect security IoT vulnerabilities is straightforward? No, we cannot!

In particular, the IoT ecosystem comprises quite diversified types of software. As clearly stated by the description of I3, this comprises "web, backend API, cloud or mobile interfaces", and embedded software as well. Therefore, this ecosystem comprises software written with different programming languages, and where different software components interact. Unfortunately, existing static analyzers are mostly focused on a single programming language; if programs written in different languages interact with each other, then the analysis considers each program "in isolation", missing some potential vulnerabilities or producing many false alarms. In addition, existing static analyzers do not possess any knowledge or interface to specify the physical world of an IoT system. However, the industrial impact achieved by static analyzers applied to OWASP Top 10 2017 vulnerabilities has shown that this approach can be effective for detecting and fixing these issues. But, how to adapt OWASP IoT Top 10 is still an open topic.

References

1. Analyzing with sonarQube Scanner. https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner.
2. F. Spoto. The Julia Static Analyzer for Java. In *Proceedings of SAS '16*, Lecture Notes in Computer Science. Springer, 2016.
3. Omer Tripp, Marco Pistoia, Stephen J. Fink, Manu Sridharan, and Omri Weisman. TAJ: Effective Taint Analysis of Web Applications. In *Proceedings of PLDI '09*. ACM, 2009.