Behaviour-aware Security Smell Detection for Infrastructure as Code

Opdebeeck, Ruben; Zerouali, Ahmed; De Roover, Coen

Behaviour-aware Security Smell Detection for Infrastructure as Code

Presentation Abstract

Ruben Opdebeeck[1,*], Ahmed Zerouali[1] and Coen De Roover[1]

[1]Vrije Universiteit Brussel, Brussels, Belgium

Abstract

Infrastructure as Code (IaC) is a vital part of modern DevOps workflows, and the security of deployed infrastructures is of the upmost importance. In this presentation, we will highlight the importance of taking into account IaC script behaviour when detecting *security smells*. Specifically, we present GASEL, a security smell detector based on program dependence graphs, which takes into account the control and data flow of Ansible IaC scripts. GASEL supports 7 distinct security weaknesses, such as hardcoded passwords and missing integrity checks. Using an oracle of 243 real-world weaknesses, we show that GASEL outperforms the state-of-the-art detectors. Moreover, we perform an empirical study on more than 15.000 Ansible scripts to show that the inclusion of control and data flow information is vital to detect security smells in real-world code.

1. Presentation Abstract

Infrastructure as Code (IaC) enables practitioners to write scripts to provision, configure, and manage computing infrastructure such as cloud machines. Ansible has emerged as a popular automation platform that can be used to implement IaC practices. Since IaC is essentially code, it suffers from many of the same problems plaguing general-purpose code, including security weaknesses. Prior work [1] has uncovered several security smells that are applicable to Ansible code. However, their detection approach does not take control and data flow into account, causing it to report false positives and miss true instances of security weaknesses. For instance, Figure 1 depicts an Ansible script in which a hardcoded string, defined on line 4, indirectly flows into the usage of a password on line 10. We find that state-of-the-art tools cannot detect this smell, since it involves both data flow indirection through the use of variables, and control flow indirection because of several jumps in the execution order.

In this presentation, we will present our approach and research results [2] to alleviate these issues. We propose an approach based on program dependence graphs (PDGs) for Ansible [3]. These PDGs represent each Ansible task, the execution order between these tasks, and their data dependences. Figure 2 depicts the PDG corresponding to the motivating example of Figure 1.

BENEVOL'23, 22nd Belgium-Netherlands Software Evolution Workshop, November 27–28, 2023, Nijmegen, The Netherlands

*Corresponding author.

ruben.denzel.opdebeeck@vub.be (R. Opdebeeck); ahmed.zerouali@vub.be (A. Zerouali); coen.de.roover@vub.be (C. De Roover)

```yaml
- hosts: ...
  roles:
    - role: overdrive3000.percona
      root_password: _password_

# In overdrive3000/percona role
- name: Update MySQL root password
  mysql_user:
    name: root
    password: '{{ root_password }}'
```

Figure 1: Real-life instance of an indirect *Hardcoded Secret* smell.

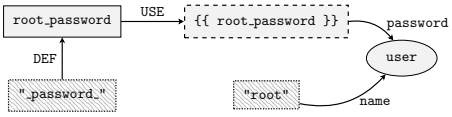

Figure 2: PDG of the example script.

To detect security smells in PDG, we leverage the Cypher graph query language and implement queries for 7 security smells applicable to Ansible.

We will describe the construction of an oracle of 243 real-life security weaknesses and use it to evaluate GASEL, the prototype implementation of our approach, and compare it to two state-of-the-art detectors. Our evaluation shows that GASEL achieves the highest precision and highest recall for 4 and 6 of the 7 smell types, respectively. We then apply the prototype to a curated dataset of over 15.000 Ansible scripts. We find nearly 8.000 weaknesses affecting nearly half of the studied repositories. Of these, 33% cross file boundaries and thus require control flow information to detect. Analogously, 55% involve data flow indirection, requiring knowledge of variables and their usages. These findings support the necessity for taking script behaviour into account to accurately detect security smells.

References

[1] A. Rahman, M. R. Rahman, C. Parnin, L. Williams, Security smells in Ansible and Chef scripts: A replication study, ACM Trans. Softw. Eng. Methodol. 30 (2021). doi:10.1145/3408897.

[2] R. Opdebeeck, A. Zerouali, C. De Roover, Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?, in: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), 2023, pp. 534–545. doi:10.1109/MSR59073.2023.00079.

[3] R. Opdebeeck, A. Zerouali, C. De Roover, Smelly variables in ansible infrastructure code: Detection, prevalence, and lifetime, in: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 61–72. doi:10.1145/3524842.3527964.