Context-Aware Security Intelligence of Vulnerability Scanners in Cloud-native Environments

Simon Ammer, Jens Krösche
Mobility & Energy
University of Applied Sciences Upper Austria
Hagenberg im Mühlkreis, Austria
email: firstname.lastname@fh-hagenberg.at

Markus Gierlinger, Mario Kahlhofer
Dynatrace Research
Linz, Austria
email: firstname.lastname@dynatrace.com

***Abstract*—Even as black-box web vulnerability scanners help identify security vulnerabilities of web applications, they still have problems with false alarms, as they lack insight into the context of applications. Without this supplemental information like the topology of the underlying application or the runtime, scanners cannot precisely assess a threat's actual severity, leading to false alarms and a challenge for security experts to prioritize vulnerabilities. Especially with the increasing popularity of microservices and highly dynamic cloud environments, this prioritization task becomes more difficult due to this environment. This paper bridges this gap by enriching web vulnerability scanner reports with context information to understand security threats better and reduce false positives. To this end, we developed a rule-based system that is extensible for multiple use cases, and we propose a framework to evaluate the approach's effectiveness using the insecure web applications Unguard and Open Web Application Security Project (OWASP) JuiceShop.**

***Keywords*—cloud computing; web application security; distributed systems security; context-awareness; rule-based adaptation.**

I. Introduction

Cloud computing is a fundamental building block for today's digital transformation, and a cloud-first strategy is becoming the norm, according to Taleb and Mohamed [1]. However, according to Behl [2], cloud computing also adds new risks of being vulnerable to cyber-attacks and demands more than traditional security solutions. Even as cloud providers ensure security on a certain level based on the shared responsibility model like the ones from Amazon [3] and Microsoft [4], still, according to the models, the customers also have security responsibilities depending on the service model. Several methods exist to improve security. The Amazon Web Services (AWS) DevSecOps reference architecture from Manepalli [5] of AWS and also the DevSecOps primitives for a reference platform from Chandramouli [6] of the National Institute of Standards and Technology (NIST) list the following methods:

- Software Composition Analysis (SCA): Identify potential risks when using third-party software (i.e., libraries, packages, etc.).
- Static Application Security Testing (SAST): Analyze the source code during development to find issues.
- Dynamic Application Security Testing (DAST): Examine the outside security posture.
- Interactive Application Security Testing (IAST): Combine the working principle of SAST and DAST.
- Runtime Application Self Protection (RASP): Monitor applications in run-time to detect threats and block some execution paths if necessary.

Dynamic web application vulnerability scanners are in the category of DAST tools. These tools need minimal human interaction and have the advantage that they are programming language independent. Especially, the automatic aspect is vital with the growing number of web applications where penetration tests are limited due to resource and time constraints. Such tools can also support the DevSecOps approach of companies by running security scans automatically in a continuous integration and delivery (CI/CD) pipeline before releasing applications. The scanners themselves search for vulnerabilities by attacking and probing the application's web and API interfaces from an outside perspective.

Offered scanners also report false positives according to Bau, Bursztein, Gupta, *et al.* [7] and Alsaleh, Alomar, Alshreef, *et al.* [8], which adds to the challenge of security experts to prioritize vulnerabilities correctly. For example, false-positive Structured Query Language (SQL) injections where the detected technology does not match the actual technology. Some research, exemplified in Section II, tries to create new scanners that provide better results, whereas others emphasize techniques that combine static and dynamic security testing elements. The higher complexity of modern web applications and microservice architectures make this undertaking even more challenging because they use different technologies at different abstraction layers. Also, with the ever-changing nature of cloud environments, the prioritization of vulnerabilities for security experts becomes even more complex.

This work proposes a technique to support this prioritization and reduce false positives of security tools in a cloud-native environment by utilizing inherent contextual information of the environment, like topology and runtime. The proposed method is generally applicable, but the implemented prototype focuses on the open-source web application security scanner OWASP Zed Attack Proxy (ZAP), utilizes Kubernetes as container orchestration service, and uses Dynatrace as an observability software. However, other observability platforms can also be

 ISBN: 978-1-61208-951-5

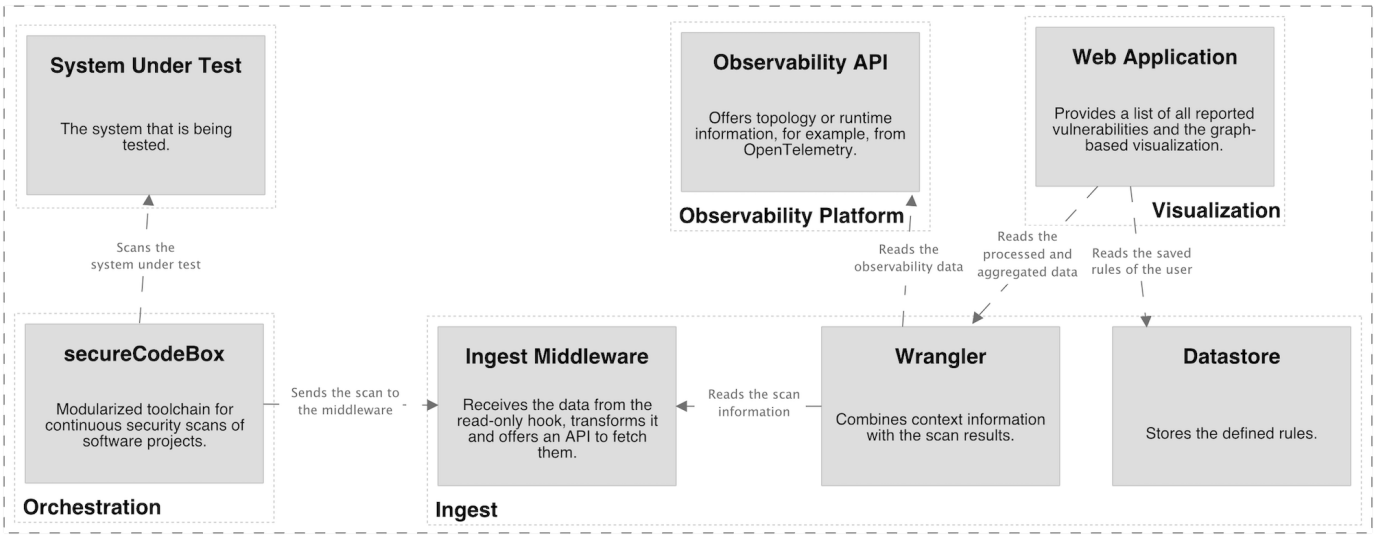

Fig. 1. The proposed architecture of the system.

used if they offer data like the expected data structure shown in Section III.

Specifically, our contributions are:

- *Context-aware ruleset engine*: DAST is known to report false-positive alerts according to Doupé, Cova, and Vigna [9]. A context-aware rule-based filtering supports the prioritization of threats and the reduction of false-positives.
- *Graph-based security posture visualization*: The method of visualizing the relationship between application assets and topology information from an attacker's perspective. This visualization assists security experts in prioritizing threats.

Second, we propose a framework to evaluate the methods using the insecure web application Unguard (not public at the time of this writing) and OWASP JuiceShop.

The rest of the work is organized as follows. Section II discusses related work. The proposed concept is described in Section III. Section IV presents a framework to evaluate the approach with two insecure web applications. Finally, in Section V, we summarize and discuss, and give an outlook for future work.

II. Related Work

As proposed to Doupé, Cova, and Vigna [9], web application scanners are comprised of three modules in most cases: a crawler, an attacker, and an analysis module. The crawler tries to find all the reachable pages and input points of an application by following redirects and links. The attacker module then uses the result to generate values that may trigger a vulnerability. As last step, the analysis module analyzes the application's responses to detect vulnerabilities. A systematic review by Seng, Ithnin, and Said [10] analyzed the methodology of existing academic manuscripts for quantifying scanners' quality and found no standard methods. Only a common practice, namely calculating the number of detected vulnerabilities, was found. Mburano and Si [11] used additional metrics like the True Positive Rate (TPR) and False Positive Rate (FPR) and carried out a benchmark of open-source scanner results using the Open Web Application Security Project (OWASP) Benchmark and the Web Application Vulnerability Scanner Evaluation Project (WAVSEP).

Studies from 2010 of Bau, Bursztein, Gupta, *et al.* [7] and Doupé, Cova, and Vigna [9] show that scanners offer good results for simple historical vulnerabilities but have difficulties with advanced or second-order forms. Another finding was that the cost of a scanner and the provided functionality have no strong correlation. False positives occurred due to the disclosure of server paths and absolute paths of anchors, but there were also genuine ones. More recent studies provided similar results with Alsaleh, Alomar, Alshreef, *et al.* [8] recommending better verification checks of reported false positives of investigated open source scanners to simplify the manual verification process. Also, Anhar and Suryanto [12] state that further research is necessary for modern web applications based on frameworks like React, Angular, and similar ones.

According to Gartner [13], context-aware security (CAS) is described as the usage of additional contextual information to improve security decisions, which is used in our work. To the best of the authors' knowledge, there is not much research on web application scanners utilizing CAS. Even so, some work on intrusion detection systems (IDS) like the following relies on contextual information. Chergui and Boustia [14] list the following information for compromised entities as helpful: network configuration, protocols, operating system, services, and applications. The latter three are also used in the prototype of this work. Eschelbeck and Krieger [15] use a similar approach by including services, ports, operating systems, and vulnerabilities as information to eliminate noise from IDS.

 ISBN: 978-1-61208-951-5

III. METHODOLOGY

This paper utilizes the two chosen context information, topology and runtime, relying on the monitoring and observability platforms or systems to provide the necessary information. Such systems know about the topology of the applications but do not necessarily have access to the actual code. Therefore, SAST and IAST are not the focus of this work. The main focus is DAST because they attack an application from the outside and do not have run-time information which the observability platform can provide.

A. Architecture

Our system Themis, shown in Figure 1, consists of four main components. The *orchestration* automates the scheduling of security testing tools in Kubernetes. The *observability platform* collects all contextual information. The *ingest* fuses data and integrates multiple data sources afterwards, and the *visualization* displays the information for security experts.

1) Orchestration: Runs different security scans, either on demand or in a specified interval. It also extracts the scan result and parses it to a unified format finding format of the secureCodeBox project [16]. Afterwards, the data is sent to the receiving *ingest* components using a webhook.

2) Observability Platform: The observability platform has constant 'insight' into the applications and collects the monitored data, for example, from OpenTelemetry. Listing 2 shows the expected data structure of such an observability platform as a JavaScript Object Notation (JSON) object. The attributes $toRelationships$ and $fromRelationships$ contain the connections of an application to others. The $softwareTechnologies$ attribute shows the application's technologies like C# and Java.

```
[{
  "hostname": "jshop.juiceshop.svc",
  "toRelationships": {
    "calls": [
      "APPLICATION-C46735D64G092C8H"
    ]
  },
  "softwareTechnologies": [{
      "type": "NODE_JS (14.18)"
  }],
  "fromRelationships": {
    "runsOn": [
      "PROCESS_GROUP-FQVDC6732B412233"
    ]
  }
}]
```

Fig. 2. Data structure for the topology and runtime information.

3) Ingest: The core part of this work, the ingest, is split into three parts: Middleware, Wrangler, and Datastore. The first part, the Ingest Middleware, transforms the result of the orchestration component and potential external vulnerability scanners to the DAST report format of Gitlab [17], partially seen in Figure 3. The Wrangler then uses this report to correlate the results with the data of the observability platform. The correlation is performed based on the hostname of the application, which is present both in the topology data and the scan result. The aggregated data can then be used to filter the scan results with pre- and user-defined rules, which are saved in the Datastore. Security experts manually create these rules, but the architecture could be used to enable problem-specific rule sets automatically based on specific characteristics of applications in further work.

```
"vulnerabilities": [{
  "name": "SQL Injection - SQLite",
  "identifiers": [{ "name": "CWE-89" }],
  "severity": "HIGH",
  "location": {
    "hostname": "http://jshop.juiceshop.svc",
  }
}]
```

Fig. 3. Data structure for the found vulnerabilities of a scan result.

4) Visualization: A penetration tester can view the ingested data on a web application. Furthermore, it has a rule editor for custom user rules defined in Rego [18], a query language inspired by Datalog. A sample rule is shown in Listing 4 that filters the false-positive SQL injections, mentioned in Section I, where the detected technology does not match the actual technology. For example, possible injection of MySQL, while the application itself does not use this technology.

```
deny[msg] {
  input.vulnerability.type == "SQLi"
  not input.vulnerability.technology == \
      topology.database.type
  msg: sprintf("SQLi vulnerability %s does \
      not match the underlying database.")
}
```

Fig. 4. A user-defined rule to avoid false positives of SQL injection alerts.

IV. DISCUSSION

Previous research from Seng, Ithnin, and Said [10] and Doupé, Cova, and Vigna [9] shows that quantifying the results of web application security scanners is complex and common practice is to run the scanner against multiple testbeds. One important metric is the TPR of discovered vulnerabilities of such tests. As this paper does not improve the internals of web application security scanners but aims to improve the produced results by reducing false positives, spurious vulnerabilities are the most important metric. This section outlines the used methods for evaluating the proposed technique.

The proposed approach is evaluated by running one open-source (OWASP ZAP) and one commercial scanner (Tenable Web App Scanning), against two web applications. One of these testbeds is the monolith application OWASP JuiceShop, which includes real-world applications' security flaws and vulnerabilities from the OWASP TopTen. The second insecure

 ISBN: 978-1-61208-951-5

application is a custom testbed called Unguard from the company Dynatrace that contains hand-inserted vulnerabilities with proven attack patterns. These two applications have different application architectures to analyse which contextual information is helpful in which architecture.

False Positives: Whether the false positive rate improved with the proposed approach still needs to be evaluated. Nevertheless, it is assumed that the results with Unguard will be better than the ones with OWASP JuiceShop. Because the latter application is a monolith, the topology does not provide much value here. Only the information of the application's technology will therefore be beneficial for the results, which will result in a worse filter result.

Visualization: The graph-based visualization of the context seems to provide value in the mitigation process. Still, this result is hard to measure as it is subjective based on the application's user. Nevertheless, it is shown that vulnerability and topology information can also be merged and presented graphically.

V. Conclusion

In this paper, we presented a system that tries to reduce false positives of scanners in a cloud environment. Although there is much research on improving the functionality of web application scanners, utilizing contextual information is not much examined to the authors' knowledge. To this end, an architecture to combine scanner results and topology information has been proposed, and a prototype called Themis with a rule-based filtering approach and graph-based visualization of found vulnerabilities was shown. The effectiveness will be thoroughly evaluated in the near future with the two projects, Unguard and OWASP JuiceShop, using the FPR as a metric. It is assumed that results with OWASP JuiceShop will be restricted due to the monolithic architecture of the application. Better results are expected for microservice-based applications where topology information is beneficial. Additionally, the effectiveness of the rule-based filtering and the graph-based visualization's helpfulness for security experts has to be further evaluated in a production environment.

Acknowledgment

This research was partially supported by Dynatrace. We thank our colleagues from the Cloud Application Security Protection department.

References

[1] N. Taleb and E. A. Mohamed, "Cloud Computing Trends: A Literature Review," Jan. 16, 2020. DOI: 10.36941/ajis-2020-0008.

[2] A. Behl, "Emerging Security Challenges in Cloud Computing: An Insight to Cloud Security Challenges and their Mitigation," in *2011 World Congress on Information and Communication Technologies*, Dec. 2011, pp. 217–222. DOI: 10.1109/WICT.2011.6141247.

[3] Amazon. (Apr. 2022). Shared Responsibility Model, Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/ (visited on 04/04/2022).

[4] Microsoft. (Mar. 1, 2022). Shared Responsibility in the Cloud, [Online]. Available: https://docs.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility (visited on 04/04/2022).

[5] S. Manepalli. (Jun. 25, 2021). Building an End-to-end Kubernetes-based DevSecOps Software Factory on AWS, Amazon Web Services, [Online]. Available: https://aws.amazon.com/blogs/devops/building-an-end-to-end-kubernetes-based-devsecops-software-factory-on-aws/ (visited on 04/04/2022).

[6] R. Chandramouli, "Implementation of DevSecOps for A Microservices-based Application with Service Mesh," Sep. 29, 2021. DOI: 10.6028/NIST.SP.800-204C.

[7] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in *2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA: IEEE, May 2010, pp. 332–345. DOI: 10.1109/SP.2010.27.

[8] M. Alsaleh, N. Alomar, M. Alshreef, A. Alarifi, and A. Al-Salman, "Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners," *Security and Communication Networks*, vol. 2017, pp. 1–14, May 24, 2017. DOI: 10.1155/2017/6158107.

[9] A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Kreibich and M. Jahnke, Eds., vol. 6201, Bonn, Germany: Springer, Jul. 8, 2010, pp. 111–131. DOI: 10.1007/978-3-642-14215-4_7.

[10] L. Seng, N. Ithnin, and S. Said, "The Approaches to quantify Web Application Security Scanners Quality: A Review," *International Journal of Advanced Computer Research*, vol. 8, pp. 285–312, Sep. 28, 2018. DOI: 10.19101/IJACR.2018.838012.

[11] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," in *2018 26th International Conference on Systems Engineering (ICSEng)*, Sydney, NSW, Australia: IEEE, Dec. 2018, pp. 1–6. DOI: 10.1109/ICSENG.2018.8638176.

[12] A. A. Anhar and Y. Suryanto, "Evaluation of Web Application Vulnerability Scanner for Modern Web Application," presented at the International Conference on Artificial Intelligence and Computer Science Technology, Yogyakarta, Indonesia: IEEE, 2021, pp. 200–204. DOI: 10.1109/ICAICST53116.2021.9497831.

[13] Gartner. (Apr. 2022). Definition of Context-aware Security, Gartner, [Online]. Available: https://www.gartner.com/en/information-technology/glossary/context-aware-security (visited on 04/04/2022).

[14] N. Chergui and N. Boustia, "Contextual-based Approach to reduce False Positives," *IET Information Security*, vol. 14, no. 1, pp. 89–98, 2020. DOI: 10.1049/iet-ifs.2018.5479.

[15] G. Eschelbeck and M. Krieger, "Eliminating Noise from Intrusion Detection Systems," *Information Security Technical Report*, vol. 8, no. 4, pp. 26–33, Apr. 1, 2003. DOI: 10.1016/S1363-4127(03)00004-9.

[16] secureCodeBox. (Apr. 2022). Finding Format, Finding, [Online]. Available: https://docs.securecodebox.io/docs/api/finding (visited on 04/04/2022).

[17] Gitlab. (Mar. 29, 2022). DAST Report Format, GitLab, [Online]. Available: https://gitlab.com/gitlab-org/security-products/security-report-schemas/-/blob/master/dist/dast-report-format.json (visited on 04/04/2022).

[18] Open Policy Agent. (Apr. 2022). Policy Language - Rego, Open Policy Agent, [Online]. Available: https://openpolicyagent.org/docs/latest/policy-language/ (visited on 04/04/2022).