Security Research: Program Analysis Meets Security

Padmanabhan Krishnan

Oracle Labs, Brisbane, Australia, QLD 4000
paddy.krishnan@oracle.com

Abstract. In this paper we present the key features of some of the security analysis tools developed at Oracle, Labs. These include Parfait, a static analyser, Affogato a dynamic analysis based on run-time instrumentation of Node.js applications and Gelato a dynamic analysis tool that inspects only the client-side code written in JavaScript. We show the how these tools can be integrated at different phases of the software development life-cycle. This paper is based on the presentation at the ICTAC school in 2021.

1 Motivation

Cloud-based execution of large applications enables consumers to buy services as-and-when it is required without investing in expensive infrastructure. The challenge of ensuring that such services are always secure and available is shifted to the provider of the cloud services. There is no doubt that the attack surface associated with cloud-based services is larger than that for on-premise based execution. Cloud services can be sub-divided into infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS).

From a SaaS or application security perspective, security breaches aim to steal data. That is, the customers' data (e.g., credit card information, health records) is valuable. It is the cloud provider's responsibility to protect customer's data. So information security is one of the main guarantees that the customer is looking for. Secure information flow is one way to ensure that the customer's data-protection requirements are satisfied. Software security provides a variety of mechanisms to ensure proper information flow is enforced. One can detect improper information flows at different levels of the infrastructure and software stack. For instance, firewalls can be used to detect and prevent improper flows at the network level while RASP solutions can provide application specific protection.

The research questions our group addresses are as follows.

- What are the security issues that matter? Can we identify vulnerabilities that when exploited have high impact? Can we detect and prevent 0-day attacks?
- How can one leverage different program analysis concepts to detect security vulnerabilities?

– What are the tradeoffs when handling industrial scale systems? For instance, how can a single application that uses different technologies be analysed?

In this paper we discuss some of the solutions and challenges towards ensuring applications are secure. That is, we describe techniques that check the applications have the correct behaviour. This has to be linked with the desired information-flow security. We focus only on *integrity* (i.e., is data coming from untrusted sources deemed to be safe) and *confidentiality* (i.e., is data released only to the proper entities) [1]. The analysis of other properties such as non-repudiation, and availability is beyond the scope of this paper.

Ideally, we would like to mitigate all risks. Hence a defense-in-depth approach is adopted. We check for a wide variety of potential security vulnerabilities (e.g., OWASP Top 10). We do not restrict our attention to only exploitable vulnerabilities. For instance, SQL injection can be used to violate both integrity (by inserting malicious values) and confidentiality (by exfiltrating sensitive values) requirements. To prevent potential attacks, we check if user-controllable data is sanitised and if any value returned by security-sensitive states is declassified.

In the next section we describe our tools based on a decomposition of the structure of applications and the software development life-cycle.

2 Approach

All modern applications can be abstracted into client-side (i.e., behaviour seen via the browser) and server-side (i.e., behaviour executed by servers which are not directly accessible to the client) sub-systems. Most client-side behaviour is written in JavaScript and executed by the browser. There is diversity in server-side code and such code includes Java, Python, C, and C++. Towards deploying our tool, we adopt a simplified software development process which is shown in Figure 1.

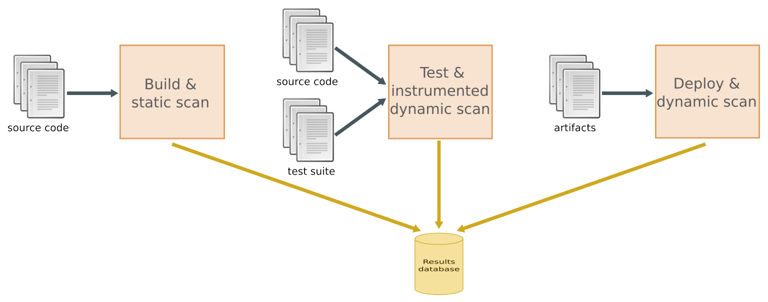

Fig. 1. Simplified Software Development Process

Given the complexity of these systems, we support two main strategies to detect potential security vulnerabilities. These are as follows.

– Static analysis (also called SAST or static application software testing) where the source code is analysed. Here we do not need the entire working system as the source code is not executed. Our approach also supports the analysis of partial code; i.e., the entire codebase is not required.
– Dynamic analysis (also called DAST or dynamic application software testing). If the source code is available, we can instrument it at a high-level while if the source code is not available where we can perform low-level instrumentation. In both cases we can have monitors that can observe the behaviour.

Various tools have been developed at Oracle Labs to improve automated detection of security vulnerabilities. These include Parfait, Affogato and Gelato and the key aspects of each of these tools is described below.

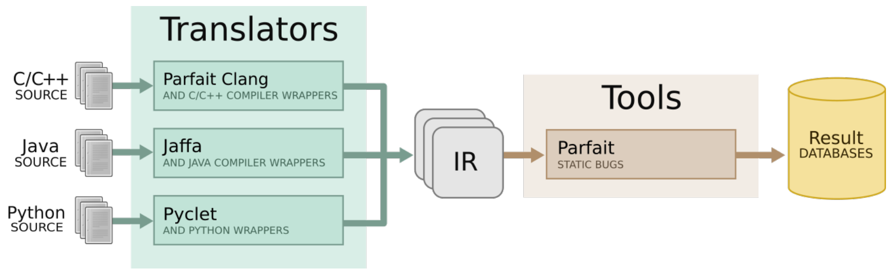
Translators
C/C++ SOURCE
Java SOURCE
Python SOURCE
Parfait Clang AND C/C++ COMPILER WRAPPERS
Jaffa AND JAVA COMPILER WRAPPERS
Pyclet AND PYTHON WRAPPERS
IR
Tools
Parfait STATIC BUGS
Result DATABASES

Fig. 2. High-level Architecture of Parfait

3 Parfait

Parfait [2, 3] is a static analysis tool that focuses on high precision (90% true positive reports) and scalable (approximately 10 minutes per million lines of code on a standard developer laptop). This is possible because it focuses only on vulnerabilities that matter (e.g., SQL injection, cross-site scripting, XXE). These vulnerabilities are identified by the owner of the codebase. Because they know the functionality of the application, they can assess the impact if a vulnerability is exploited. Parfait supports the analysis of programs written in C, C++, Java and Python. It translates the source code into an LLVM IR [4] which is then analysed using customisations of classic program analysis techniques [5]. Figure 2 shows Parfait's architecture. Parfait has been deployed in the *Build and static scan* phase as shown in Figure 1.

In summary, Parfait has a very high true positive rate and thus identifies actionable items. Parfait thus minimises developer effort in fixing issues reported by Parfait. Being a static analysis tool it can handle incomplete code. The usual challenges of static analysis (e.g., how to summarise the use of frameworks and

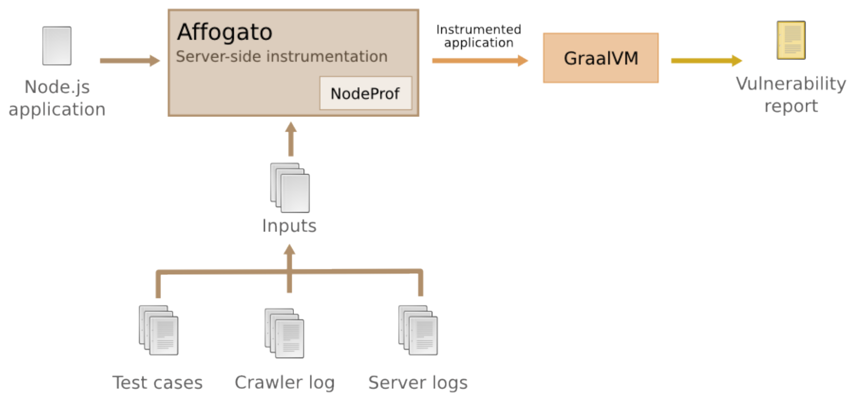

Fig. 3. High-level Architecture of Affogato

reflection) remain. We are also working on incremental (e.g., commit time) analysis which will improve the scalability [6].

4 Affogato

Affogato [7] is an instrumentation-based dynamic analysis tool for Node.js. Affogato uses taint tracking and inference mechanisms to detect vulnerabilities. The precision and overheads can be tuned by controlling the exact behaviour of the taint analysis. As with any instrumentation-based analysis the overhead amortises for long runs but is high for short running tests. The high-level architecture of Affogato is shown in Figure 3. Affogato can detect the usual injection attacks (such as SQL injection) and other specific vulnerabilities such information leakage. Affogato has been used in the *Test and instrumented dynamic scan* phase as shown in Figure 1.

In summary, Affogato works on an running instance of the application. It relies on test cases to exercise the behaviour. The runtime overheads for long-running tests are acceptable. The main challenge is related to the effect of instrumentation. We need to ensure that the original semantics are preserved after instrumentation. As with any dynamic analysis, we have to reduce the runtime overheads introduced by extra code that has been introduced.

5 Gelato

Gelato [8] is a client-side analysis tool that does not rely on the availability of the server-side source code. The client-side JavaScript code is analysed and used to detect end-points exposed by the server. It also uses taint tracking and hence can detect DOM-XSS and reflected XSS. Figure 4 shows the high-level architecture of Gelato. Gelato is a state-aware crawler that uses a man-in-the-middle proxy to intercept traffic between the browser and server. This traffic is both instrumented and analysed to detect security issues. Gelato has been used in the *Deploy and dynamic scan* phase as shown in Figure 1.

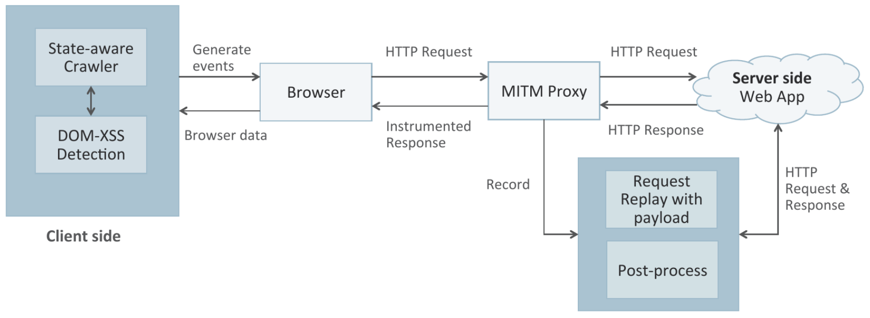

Fig. 4. High-level Architecture of Gelato

Gelato, like Affogato, works on a running instance. But unlike Affogato, Gelato does not require test-cases. By analysing the client-side code, Gelato mimics an external, real-world attacker. A key challenge is how to detect if Gelato is making progress without access to the server code. Without such knowledge, it is hard to estimate the coverage, say the percentage of endpoints that are actually detected, of the analysis. While integrating fuzzers might help, it is still difficult to get precise information.

6 Safe-PDF

Safe-PDF [9] uses the SAFE abstract-interpretation framework [10] to detect malicious JavaScript code in PDF documents. While abstract-interpretation can, in general, be expensive, it takes less than four seconds per document which is acceptable, say when scanning e-mail attachments. While Safe-PDF uses a static analysis technique, it can also be used when the application (say the mail system) is executing. The delay introduced by the analysis is not noticeable in a mail-delivery system.

Using abstract-interpretation allows Safe-PDF to go beyond detecting syntactic patterns. By using the semantics of the JavaScript code, Safe-PDF has very low false-positives. The main challenge is how to generalise this approach. For instance, what techniques are needed for other types of documents, e.g., MS-Office documents.

7 Future Work

Thus far our research has focused on preventing security vulnerabilities being introduced (at the coding level), check for potential violations of information-flow policies at the testing phase, and preventing attacks at run time (at the deployment and operations phases). Currently, our tools are run independent of each other.

But security analysis is like a game where the attacker has to find only one exploit while the defenders (e.g., the cloud service providers) have to protect against all possible attacks.

Real systems consist of different technologies including the use of many third party libraries. It is unrealistic to expect that a single tool that can handle all these technologies. Tools that focus on specific aspects are more effective in practice. Therefore, different tools are required to cover all security aspects. These tools generate different signals and it is important to combine them to have a single security view. This is what we call the Intelligent Application Security (IAS) [11]. It is our vision for the future. We envisage IAS as providing the necessary infrastructure that enables different security-analysis tools to share information and refine their analyses based on the input received from other tools. An initial step in this vision is described in [12] which combines the features of Affogato, Gelato and a fuzzing tool.

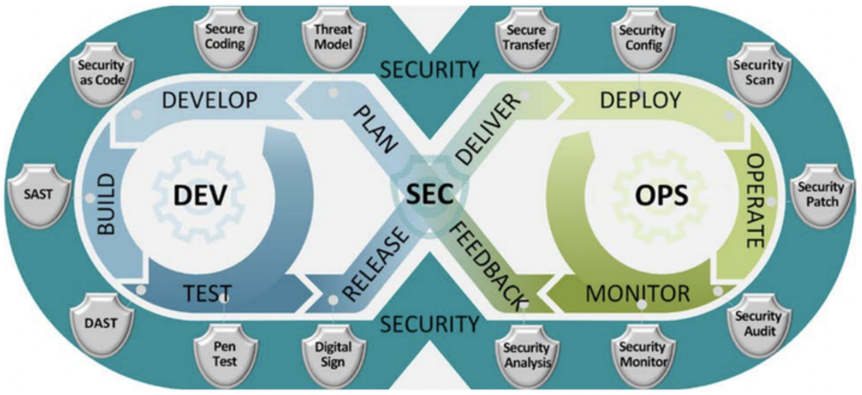

Fig. 5. DevSecOps Cycle

Our future research will address the tools needed to address all aspects within the DevSecOps [13] (see Figure 5) process. DevSecOps is a continuous process designed to overcome silos in the development of large systems [14]. The challenge is that security needs to be considered at every stage. The role of various tools is shown in Figure 5. While the tools presented in this paper address some of these phases (e.g., develop, build, test), we need to develop tools and techniques for other aspects such as security configuration, patching, audit etc.

Acknowledgements

This paper is reporting the work done by the entire team at Oracle Labs, Australia over many years. The author is thankful for their support in developing this paper.

References

1. Krishnan, P., Lu, Y., Raghavendra, K.R.: Detecting unauthorised information flows using points-to analysis. Engineering and Technology Reference (2016)
2. Cifuentes, C., Keynes, N., Li, L., Hawes, N., Valdiviezo, M.: Transitioning Parfait into a development tool. IEEE Security and Privacy **10**(3) (May/June 2012) 16–23
3. Gauthier, F., Keynes, N., Allen, N., Corney, D., Krishnan, P.: Scalable static analysis to detect security vulnerabilities: Challenges and solutions. In: IEEE SecDev. (2018)
4. Lattner, C., Adve, V.: The LLVM compiler framework and infrastructure tutorial. In: LCPC: Mini Workshop on Compiler Research Infrastructures. (2004)
5. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. 2 edn. Springer (2005)
6. Krishnan, P., O'Donoghue, R., Allen, N., Lu, Y.: Commit-time incremental analysis. In: SOAP, ACM (2019)
7. Gauthier, F., Hassanshahi, B., Jordan, A.: AFFOGATO: runtime detection of injection attacks for node.js. In: Companion Proceedings for the ISSTA/ECOOP Workshops, ACM (2018) 94–99
8. Hassanshahi, B., Lee, H., Krishnan, P.: Gelato: Feedback-driven and guided security analysis of client-side web applications. In: SANER. (2022)
9. Jordan, A., Gauthier, F., Hassanshahi, B., Zhao, D.: SAFE-PDF: robust detection of javascript PDF malware using abstract interpretation. Technical report, CoRR (2018)
10. Park, J., Ryou, Y., Park, J., Ryu, S.: Analysis of javascript web applications using SAFE 2.0. In: ICSE, IEEE Computer Society (2017) 59–62
11. Cifuentes, C.: Towards intelligent application security (invited talk). In: SOAP. (2021)
12. Gauthier, F., Hassanshahi, B., Selwyn-Smith, B., Mai, T.N., Schlüter, M., Williams, M.: Experience: Model-Based, Feedback-Driven, Greybox Web Fuzzing with BackREST. In: European Conference on Object-Oriented Programming, ECOOP. Volume 222 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022) 29:1–29:30
13. DoD Chief Information Officer: DoD enterprise devsecops reference design. Technical report, Department of Defense (2019)
14. Rajapakse, R.N., Zahedi, M., Babar, M.A., Shen, H.: Challenges and solutions when adopting devsecops: A systematic review. Information and Software Technology **141** (2022)