wallarm

WHAT'S WRONG WITH DAST

How useful are scanners for discovering software vulnerabilities? It all depends on what is scanned, how it is scanned, and how the results are interpreted.

According to **Technopedia**, "Dynamic application security testing (DAST) is a process of testing an application or software product in an operating state. This kind of testing is helpful for industry-standard compliance and general security protections for evolving projects." Examples of popular scanners include **OWASP ZAP** and **burp**.

Typically, DAST involves running a scanner to test how an application behaves in response to aggressive stimuli. The scanner is capable of sending attack payloads and checking the server responses. This is how a scanner detects vulnerabilities: if the application's response to a request with a payload matches with a signature rule, the scanner concludes there is a vulnerability present.

The logic is almost the same as with using success/fail criteria in unit testing, isn't it?

For the scanner to be effective, it needs to address three important areas when conducting scans:

- Which payloads to send *(payloads gallery)*
- Where to send the payloads *(application endpoints discovery)*
- How to analyze the application response *(application behavior discovery)*

These questions are typically asked by all the security experts who evaluate these kind of solutions. These three areas together are known as "coverage."

© 2017 Wallarm, Inc

Most of the security organizations use vulnerability scanners and other DAST tools. However, most of the scanners that are deployed today are limited in their coverage. Consider the summary table below:

	How it is in DAST	**How it should be**
Which payloads are sent *(payloads gallery)*	Common payloads from public sources with no adaptations to a particular application Manually added payloads from post-mortem investigations	Payloads generated specifically for the application under test based on traffic and automated tests All the payloads adapted to parameter manipulation, application capabilities, and formats Number of payloads could be easily increased by using fuzzing techniques for payload generation
Where to send payloads *(application endpoints discovery)*	Crawled endpoints, discovered by browser emulation Manually configured endpoints and data formats	Endpoints are generated automatically based on traffic and automated tests
How to analyze server response *(application behavior discovery)*	Similar behavior across all the applications Manually configured rules	Application responses are profiled automatically based on the traffic and automated tests

Let’s try to explain in details why these questions are so important.

© 2017 Wallarm, Inc

PAYLOADS GALLERY

At first glance, it seems easy to apply all the known payloads to cover all the OWASP Top-10 issues. However, the reality is different. Many payloads are specific to an application platform, framework, and a lot of other parameters. Thus, it is rather inefficient to try to apply all of them. Secondly, some of the payloads could be dangerous for some applications. Even common and well-known vulnerabilities, such as XSS or SQL injections, can go undetected by common checks when the vulnerability is present. Let's consider an example of an application that manipulates the incoming data by truncating the input string to 32 characters. A scanner using a payload of 64 characters would not get the expected results in the response and would assume there are no vulnerabilities, whether or not they are present. Similar behavior will be observed if the application applies input validation but the validation is too weak to block the exploits. Another good example here is a common case of a "magic_ quotes" **PHP configuration option**, deprecated a few years ago. For a long time, most scanners failed to detect vulnerabilities "protected" by this configuration option, but all the security experts and penetration testers knew how to exploit it.

Another important parameter is the time lag before a new payload variant can be delivered to a scanner after a vulnerability has been released. This time lag is critical, but unfortunately, it often takes days to include a new vulnerability in a signature feed because all of them need to be created and validated manually by the vendor's security experts. While we are on the subject, it is worth noting that this lag is the reason why the entire virtual patch market is still alive. It seems faster to apply some signatures to protect from new payloads than to find and fix related vulnerabilities. We have already described why signatures can't protect modern applications properly in our "Evolution of Real Time Detection" white paper.

APPLICATION ENDPOINT DISCOVERY

Most scanners discover application endpoints by crawling. This means browser emulation is used to visit each link and click each button. This approach was reasonable in CGI times last century, but microservices and single-page applications make it less than useful. It is impossible to crawl the modern web because of the sophisticated API data protocols, custom behavior, and a lot of other restrictions like authentication. This issue can be solved by manual DAST scanner configuration, where each application endpoint call is described with all the variants. This is almost as difficult as creating custom unit tests for security.

Moreover, some data encapsulation cases cannot be supported by scanners. For example, try to explain to your DAST that the data in one particular endpoint should be encoded to base64 first and then inserted in a JSON parameter field. Now imagine that this JSON is then encoded again by base64, and you have a common case for enterprise APIs.

As a result, not all the endpoints are covered, which means huge blind spots. This is actually a reason why red teams often detect well-known vulnerabilities during their testing. These teams are just able to discover more application endpoints than the scanners.

© 2017 Wallarm, Inc

APPLICATION BEHAVIOR DISCOVERY

The apps are different. It appears obvious, but this is not so obvious for the DAST scanners, judging from how they operate. Most scanners try to apply some payloads and expect the same response behavior from all of the applications. As a result, a lot of vulnerabilities go undetected and a lot of detected results are, in fact, false positives. For example, when we are posting a message with some SQL-related application errors to StackOverflow.com to ask somebody how to fix it, a DAST scanner will detect that a SQL injection response signature showed up. Now when this DAST scanner tries adding a SQL injection payload in a message field to post in the same thread, it will receive back that SQL-related application error and will decide that the SQL injection vulnerability is present. However, for the StackOverflow.com application to send this kind of response is normal behavior.

A similar situation appears with time-based response checks when a DAST scanner itself creates additional overhead on the testing server, causing additional latency in the server response. As a result, the request with a " '-sleep(2)-' " payload will correlate with an additional two-second response time, but not because of the vulnerability inside the application.

CONCLUSIONS

As we showed above, most DAST solutions have a serious coverage problem.

Legacy DAST solutions are only able to accurately detect:

- The vulnerabilities that are well-known and/or listed in signature feeds.
- Issues in legacy applications with well-known data formats.
- Issues in applications that are "well behaved" and have few content exceptions.

While this approach is OK for 80 percent of applications, such as simple server pages, CGI, and legacy platforms, high value applications and apps using newer architectures and protocols need a different approach that is sensitive to context and capable of adapting to the behavior and data payloads of the individual applications.

© 2017 Wallarm, Inc

ABOUT WALLARM

Wallarm is an innovative AI startup focused on application security. With the help of machine learning, Wallarm reconstructs application context and API logic by looking at application requests and responses and uses this information to automatically create custom security protection rules for each application release, making it a great fit for CI/CD environments with frequent releases. Founded in 2013, Wallarm has already helped hundreds of SaaS and enterprise customers discover and fix critical vulnerabilities, automate web application and API runtime protection and prioritize security risks. Wallarm is a privately held company headquartered in Menlo Park, California and backed by Y-combinator, Partech Ventures, and other investors.

https://wallarm.com
155 Constitution Dr., Menlo Park, CA, 94025
(415) 940-7077