A Dose of Reality on Automated Static-Dynamic Hybrid Analysis

Chris Eng, Senior Director of Research

Executive Summary

As application inventories have become larger, more diverse, and increasingly complex, organizations have struggled to build application security testing programs that are effective and scalable. New technologies and methodologies promise to help streamline the Secure Development Lifecycle (SDLC), making processes more efficient and easing the burden of information overload.

In the realm of automated web application testing, today's technologies fall into one of two categories, Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). SAST analyzes application binaries or source code, detecting vulnerabilities by identifying insecure code paths without actually executing the program. In contrast, DAST detects vulnerabilities by conducting attacks against a running instance of the application, simulating the behavior of a live attacker. Most enterprises have incorporated at least one SAST or DAST technology; those with mature SDLCs may even use more than one of each.

In the past year or so, industry analysts and product vendors have become enamored with so-called "hybrid analysis" technologies. Hybrid techniques aim to correlate the results of SAST and DAST to dramatically expand dynamic coverage, prioritize the combined set of results, and reduce both false positives and false negatives. This whitepaper will examine each of these claims to give consumers technical insight into whether hybrid technologies can realistically live up to the hype.

Several observations will be described in the following sections:

- Hybrid analysis may expand dynamic coverage, but the lack of application context limits its effectiveness.
- The challenge of reliably generating URL-to-source mappings, coupled with the existence of URL rewriting, undermines the accuracy and usefulness of vulnerability correlation.
- Hybrid analysis does not reduce false positive rates; rather, it lulls users into a false sense of security by suggesting that non-correlated vulnerabilities are false positives.
- Correlation should not be equated with exploitability. Vulnerabilities should be prioritized based on severity and business impact, not based on how many scanners are capable of detecting it.

Expansion of Dynamic Coverage

Hybrid approaches claim to improve the coverage of dynamic analysis by leveraging static analysis to identify additional entry points and parameters that might otherwise be missed. Enumerating entry points and attack parameters may expand dynamic coverage in a purely literal sense, but blindly fuzzing more pages and more parameters without the proper context will dramatically increase testing time without providing proportional value. Hybrid analysis may expand dynamic coverage, but the lack of application context limits its effectiveness.

The idea to statically enumerate entry points and parameters stems from the fact that certain development frameworks, such as Apache Struts, rely on configuration files to assign URLs to specific entry points in the

Java code. Other configuration files may contain lists of parameters associated with each entry point. Parsing these configuration files to harvest this information helps drive the dynamic scan, filling some of inevitable coverage gaps in traditional dynamic analysis technologies.

One limitation to this approach is that not all languages or frameworks define the application surface in a declarative fashion. In fact, Struts is somewhat of an anomaly, with clean, one-to-one mappings between URLs and backend Java classes. In contrast, a Servlet-based J2EE application may have a single servlet that's responsible for directing traffic programmatically. The effectiveness of using static analysis to generate an application map will vary widely with different application architectures and framework choices. In many cases, generating a map won't even be possible.

Enumerating entry points through static analysis might improve dynamic coverage in certain situations, but in some sense this is a misguided way of approaching the problem. After all, the real problem with dynamic scan coverage isn't breadth, it's depth. One of the most common failures of dynamic crawling is multi-stage processes, such as a workflow that requires one to fill in a form properly before moving on to the next. The web application configuration files may reveal many downstream targets of a particular workflow but without capturing the sequence of actions required to legitimately reach that point. Jumping directly to the downstream URLs will result in an error in most cases, though it may still be worth checking. In general, recreating the context in which a page is requested is just as important as knowing the page exists. Static analysis may help compensate for shortcomings of the DAST product, but finding new ways to improve coverage within the dynamic scanner itself is a more logical approach.

Enumerating attack parameters has similar challenges. Simply knowing the names of parameters doesn't guarantee that the dynamic scanner can figure out how to use them in the proper context. Perhaps the 'secret' parameter is only acted upon when other parameters contain a specific combination of values. Understanding the business logic behind complex control flow decisions is a weak spot for any SAST technology. Static analysis can produce a parameter list, but it can't tell the dynamic scanner how to use them intelligently.

On the other hand, having a complete list of attack parameters could reveal parameter mismatches that DAST would never detect on its own. For example, as a web application evolves, developers may remove functionality from the user interface without disabling all of the server-side code. Or a malicious developer might add a backdoor that grants them access to the application when a secret parameter is appended to the request. A dynamic scanner can't detect either of these situations because they are not exposed through the user interface. Comparing the list of parameters found statically against those found dynamically has the potential to uncover attack vectors that would otherwise be missed entirely, though manual analysis would be required to determine the impact.

Correlation of Static and Dynamic Findings

Perhaps the flashiest claim of hybrid analysis is the ability to correlate static findings with dynamic ones – that is, to declare with certainty that the vulnerability in parameter X on this externally facing URL is the same

vulnerability found in Y.java on line 100. The advertised value of this information is that it provides more detailed remediation guidance for dynamic issues while also helping prioritize static issues.

While all of this sounds nice in theory, the reality is that equality-based correlation is highly dependent on application architecture. This is nearly identical to the challenge of generating an application map via static analysis covered in the previous section. To be able to take a URL and map that to a source file and line number is a heterogeneous problem. What works in some situations is virtually impossible in others.

The other major impediment to equality-based correlation is that it fails to account for external factors that may bring the accuracy of URL-to-source mappings into question. A simple example of this would be a web server filter such as Apache's mod_rewrite, which modifies URLs on the fly. Similar techniques are commonly used to create URLs that are optimized for SEO or simply more human readable, as shown here:

External URL	Rewritten URL
http://example.com/widgets?id=403	http://example.com/store/ViewCategory?catId=5&prodId=403
http://example.com/buy/50/10/30	http://example.com/store/AddItem?itemId=50&y=10&z=30

Because rewritten URLs are not taken into account, vulnerabilities found in these pages will not be properly correlated. The hybrid analysis correlation logic has no idea that the externally-facing URL /widgets is rewritten as /store/ViewCategory?categoryId=5, nor does it know that the 'id' parameter is rewritten as the 'prodId' parameter.

It's also worth noting that many web application firewalls, load balancers, reverse proxies, and other network devices can also be configured to perform URL rewriting. Ultimately, the challenge of reliably generating URL-to-source mappings coupled with the existence of URL rewriting undermines the accuracy and usefulness of vulnerability correlation.

Reduction of False Positives and False Negatives

Hybrid analysis claims to reduce false positives (FPs) and false negatives (FNs). Reducing false positives means reporting fewer vulnerabilities that end up not being real; reducing FNs means detecting a greater percentage of all the vulnerabilities that exist in an application.

Let's start with FPs. How could vulnerability correlation possibly reduce FPs? The only way to truly reduce FPs is by improving the accuracy of the scan logic and/or heuristics. Perhaps the claim is implying that correlated findings are real vulnerabilities, and others are FPs. That would certainly be one way to position correlation as an FP reduction mechanism, but do we really want to suppress findings that couldn't be correlated?

Consider the following scenario: if 100 cross-site scripting (XSS) vulnerabilities are detected statically but only 10 of them are correlated with XSS vulnerabilities detected dynamically, does that imply the remaining 90 are FPs? This would be a dangerous assumption, particularly if developers use the correlation data to justify not

fixing – or at least closely examining – all 100 items. In reality, many of the 90 uncorrelated items may be exploitable vulnerabilities that the dynamic scan simply failed to detect. Hybrid analysis does not reduce false positive rates; rather, it lulls users into a false sense of security by suggesting that non-correlated vulnerabilities are false positives.

Moving on to FNs, there is undeniable technical merit to the claim that a hybrid approach can uncover more vulnerabilities than the sum of the two approaches in isolation. As discussed earlier, enumerating additional entry points provides marginal benefit, but identifying and attacking additional parameters may expose latent vulnerabilities in stale code as well as potential backdoors.

Better Prioritization of Remediation Efforts

Proponents of hybrid analysis claim that correlated results present greater risk to the organization because there is proof that the vulnerability can be exploited. In other words, developers should remediate correlated issues before focusing on the others. At first glance, this approach may sound reasonable, but when considering the ramifications, its merits are questionable. The theory being put forth is that correlated vulnerabilities are inherently more important to fix, so developers should address those items first. But is this a wise approach?

The notion that a vulnerability shouldn't be taken seriously without proof of exploit is short-sighted, inefficient, and borderline irresponsible. An experienced developer can fix dozens of SQL injection vulnerabilities in the time it takes to exploit one in a convincing fashion. Using correlation as a prioritization mechanism reinforces the "prove it before I fix it" mentality that is detrimental to the overarching goal of producing secure software.

Exploitabiilty can be one component of a risk-based prioritization process, but it's important not to equate correlation status with exploitability. They are not the same thing. "Correlated" may imply "exploitable", but "not correlated" doesn't imply "not exploitable." Whether a flaw is exploited by DAST is merely one factor of risk prioritization and may even be indicative only of that particular scanner's capabilities.

Raising the priority of correlated issues will effectively de-emphasize legitimate vulnerabilities found by only one analysis method. For example, SAST and DAST are more likely to report the same reflected XSS vulnerability than the same blind SQL injection vulnerability, simply because the latter is more difficult for DAST to detect. The impact of the SQL injection vulnerability is greater, and it may very well be exploitable, but if one were to prioritize based on correlation, the XSS vulnerability would appear more important. Vulnerabilities should be prioritized based on severity and business impact, not based on how many scanners are capable of detecting it. Exploitability and discoverability are not the same thing.

What about vulnerability categories that are only detectable through one analysis method or the other? A hard-coded password will never be detected dynamically and therefore will never be correlated. Does this mean it deserves a lower remediation priority? Similarly, a deployment issue such as directory indexing will never be detected statically and therefore will never be correlated. That doesn't mean it's undeserving of an immediate fix.

Finally, tying priority to correlation status also requires the correlation algorithm itself to be credible. In practice, the correlation algorithm will contain both FPs (calling two flaws correlated when they aren't) and FNs (failing to correlate two flaws that are in fact the same). Stated differently, developers are being encouraged to build action plans around unreliable data. To illustrate this problem, consider the several possible outcomes when comparing a static finding and a dynamic finding. The following table demonstrates how priorities could be misaligned due to FPs and FNs in the correlation algorithm.

Static	Dynamic	Actual Relationship	Reported Relationship	Why?	Prioritization Suggested by Hybrid Analysis
Item A	Item B	Item A and Item B are identical	Correlated	Correlation algorithm worked as designed	These are "real" vulnerabilities, fix them first! (**Reality**: They're no more important than the other vulnerabilities detected)
Item A	Item B	Item A and Item B are unrelated	Correlated (FP)	Bugs in correlation algorithm, or perhaps URL rewriting making two items appear to be correlated when they really aren't	These are "real" vulnerabilities, fix them first! (**Reality**: This will lead to confusion seeing as there are actually two separate vulnerabilities)
Item A	Item B	Item A and Item B are identical	Not correlated (FN)	Incomplete URL-to-source mapping, URL rewriting, or other technical impediments to correlation	Not correlated, must be FPs? (**Reality**: Many of those uncorrelated vulnerabilities are not only real but also exploitable)

Conclusion

Hybrid analysis has received a great deal of attention from industry analysts and product vendors, who applaud its potential benefits while ignoring drawbacks to the approach. Hybrid solves some problems but creates others, which makes it a feature that is nice-to-have but not critical – or even important – to a security testing program. However, it can be useful, provided one understands its limitations.

Using hybrid techniques to expand attack coverage may be beneficial in certain scenarios, when the application is simplistic enough that the lack of context will not matter. Most of the time, it will just extend the duration of the dynamic scan while URLs and parameters are attacked in meaningless ways.

URL-to-source mappings will certainly help less skilled developers quickly locate the code responsible for issues discovered by DAST. Experienced developers, on the other hand, already know where to find the source code associated with a certain URL; they don't need hybrid analysis to tell them. The more complex the application, the less likely it is that reliable mappings will be generated.

Vulnerability correlation is a double-edged sword. It gives users greater confidence that certain vulnerabilities are exploitable, but incorrectly implies that all of the remaining non-correlated vulnerabilities are less important. Therefore, one must be especially careful about using vulnerability correlation data for prioritization purposes. Correlation will never be perfect, or even close to perfect, due to the complex, heterogeneous nature of modern software and the challenges of static modeling. Prioritization should be a risk-based exercise that takes many factors into account, including severity, remediation effort, exploitability, and business criticality.

In conclusion, hybrid analysis is unlikely to be a passing fad, but it will take years for it to reach a maturity level of where organizations can trust it to be accurate and reliable enough to incorporate into tried and true SDLC methodologies. If you choose to use it, be sure to understand its liabilities, and don't be fooled into thinking it's a silver bullet.

COMPANY BACKGROUND

Veracode SecurityReview's patented and proven cloud-based capabilities allow customers to govern and mitigate software security risk across a single application or an enterprise portfolio with unmatched simplicity. Veracode was founded with one simple mission in mind: to make it simple and cost-effective for organizations to accurately identify and manage application security risk.

Veracode has its roots in core technology that was first developed at the leading security consulting firm @stake in 2002 to automate application security assessments. @stake was acquired by Symantec in 2004 to bolster its application security expertise. In 2006, the technology and founders left Symantec to create Veracode with key patents on breakthrough binary code analysis, securing $19.5 million in venture capital from lead investors .406 Ventures and Atlas Ventures as well as other strategic investors.

The company was formally launched in February 2007 at the RSA Conference. The Veracode executive team has deep security and industry expertise from industry-leading security and services companies such as @stake, Symantec, Guardent, VeriSign and Salesforce.com. The management team is comprised of industry veterans representing the application security space from both the code security perspective as well as the threat space — providing a full understanding of what it takes to secure today's software.

In 2010 Veracode announced its revenues grew the previous year almost 300% from 2008 to 2009 and that it secured new capital for additional expansion in a round led by New York-based StarVest Partners. The unprecedented growth comes at the same time as other company achievements such as surpassing the 100 customer milestone, being the first cloud-based security company to reach the 50 billion lines of code scanned milestone, expanding its executive team and board of directors to include Dr. James Cash, Emeritus James E. Robison Professor of Business Administration at Harvard University Business School , and being recognized as a finalist in SC Magazine's "Best Security Software Development Solution" category. StarVest, an early investor in NetSuite and other innovative technology businesses, joins current investors, Atlas Venture and .406 Ventures in the expansion funding.

Veracode has revolutionized the way in which software is being secured. Veracode's automated, SaaS service – SecurityReview – is a fundamentally better way to manage application security risk.