Secure Coding.
Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

Interested in learning more?
Check out the list of upcoming events offering "Defending Web Applications Security Essentials (DEV522)"
at http://software-security.sans.orghttp://software-security.sans.org/events/
Testing Web Apps with Dynamic Scanning in Development and Operations

A SANS Whitepaper
Written by Barbara Filkins
June 2017

Sponsored by
Veracode
The shift to web-based applications in the past decade, combined with rapid growth in the mobile-cloud ecosystem, has required fundamental changes in the way corporate applications are designed, built and maintained. From the outside, architectures have become simplified with independent presentation, application and persistence tiers based on open standards. From an application security perspective, however, the view has become much more complex.

Secure applications require more than simply securing their components: code, libraries and subroutines. Maintaining secure web applications is a more complex equation that takes into account dependence on the underlying infrastructure—operating systems, database and business logic—interconnected by web services, network protocols and virtualization. Components can be written in different languages, and they may connect to diverse databases and run on different operating systems or web servers. In addition to software flaws, system misconfiguration may put your data at risk.

The rise of web applications has spawned another complication: growing demand for rapid application development and deployment. Development and operations first came together in 2008 as DevOps to support the continuous delivery of software at the speed of business. A similar trend is replacing the traditional view of application security as a testing activity reserved solely for the latter stages in the software development life cycle (SDLC), conducted by dedicated security staff.

DevSecOps methodologies prescribe the collaboration between developers, infrastructure designers, operations staff, business decision makers and security experts to continuously deliver secure software. Security must move into all phases of the application life cycle. Processes and tools must provide for the discovery of flaws, support continuous testing and provide metrics that can be used to make decisions within the enterprise.

Dynamic application security testing is one means to achieve a holistic approach to the verification and validation of web application security. This paper explores the use of Dynamic Application Security Testing (DAST), establishing principles and guidelines for black box testing from application design to application retirement as organizations prepare to procure DAST services and tools.
Application Security Testing

Application security testing (AST) is the “process of evaluating the security of a computer system or network by methodically validating and verifying the effectiveness of application security controls.” When discussing testing, it is important to note the subtle difference between verification and validation. A tester will verify that the software product meets the stated requirements; this can be done by inspection as well as by automated tests. A tester must validate that the product actually meets those requirements by exercising the product, including negative testing, to identify security vulnerabilities that may be present but are not accounted for in the defined security requirements.

The two major flavors of AST used to evaluate the security of web applications are Static Application Security Testing (SAST) and Dynamic Application Security testing (DAST). A comparison is provided in Table 1. We will be concentrating on those methods that evaluate the security of a web application.

Table 1. Application Security Testing: DAST vs. SAST

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAST, also known as white box testing or developer viewpoint</td>
<td>DAST, also known as black box testing or hacker viewpoint</td>
<td></td>
</tr>
<tr>
<td>• Perform code analysis, including source, binary and intermediate builds</td>
<td>• Test application components or full applications when the internal working of the component or app is not required</td>
<td></td>
</tr>
<tr>
<td>• Verifies the application from an inside viewpoint</td>
<td>• Validates the application from an outside viewpoint</td>
<td></td>
</tr>
<tr>
<td>• Exposes security flaws in code, rather than exploitable vulnerabilities</td>
<td>• Exposes actual exploits and behavior of applications responding to those exploits</td>
<td></td>
</tr>
<tr>
<td>• Finds vulnerabilities early in the SDLC, allowing remediation earlier than in the QA cycle</td>
<td>• Easily automated, efficient, flexible and scalable</td>
<td></td>
</tr>
<tr>
<td>• Supports all software methods and architectures</td>
<td>• Allows continuity of application security from QA to deployment to production</td>
<td></td>
</tr>
<tr>
<td>• Finds runtime vulnerabilities</td>
<td>• Finds runtime vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>• Integrates easily into corporate security strategy; findings map directly to risk and prioritization</td>
<td>• Integrates easily into corporate security strategy; findings map directly to risk and prioritization</td>
<td></td>
</tr>
<tr>
<td>• Can quickly be deployed by the security team</td>
<td>• Can quickly be deployed by the security team</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No evaluation of runtime vulnerabilities, as application is not tested under production conditions, where insecure server configuration may introduce serious vulnerabilities</td>
<td>• Prone to false positives and negatives, especially for fully automated solutions</td>
</tr>
<tr>
<td>• No evaluation of how components interact</td>
<td>• May not catch all vulnerabilities</td>
</tr>
<tr>
<td>• Effectiveness limited to web applications and web services</td>
<td>• Oriented toward latter phases of the SDLC</td>
</tr>
</tbody>
</table>

Dynamic Testing Across the Life Cycle

As testing methods, SAST and DAST complement each other across the SDLC. Ideally, both methods should be used during the pre-production phases. This should identify most flaws prior to deployment—provided the static reviews are conducted and completed by knowledgeable analysts, not just the developer, and the range of dynamic analysis tests is sufficiently wide. See Figure 1.

Figure 1. AST Across the Life Cycle

Ideally, the test environments should be an exact replica of production. In reality, they almost never reflect the real world. A test environment is controlled. It is usually constrained in terms of available processing power and network bandwidth and, most important, it lacks the ability to interact with other deployed applications.

DAST should be used first to determine the final secure state of the application in the test environment and then used again to automatically scan once the application is released into production. While an application may scan as secure in the test environment, an unpatched or misconfigured production web server may render the application vulnerable. These issues need to be remediated before an application is considered formally released.

DAST also plays an important role in maintenance and operations. Compliance testing, for example, will still need to occur. Don’t forget that DAST is also a powerful tool for security regression testing of updated applications.
Addressing Strategy

By exposing exploitable vulnerabilities, DAST also identifies the degree of risk to the business that vulnerabilities may pose, and does so in terms that allow stakeholders throughout the organization to make informed decisions on how to respond. It provides a unique capability that enhances an enterprise’s application security strategy, whether the organization is newly establishing a program or trying to make mature AppSec processes more efficient and effective.

How DAST can address and help communicate DevSecOps stakeholder concerns is shown in Table 2.

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Concerns</th>
<th>DAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>• Can we make informed decisions as to the state of application security?</td>
<td>• Immediately provides visibility into how identified vulnerabilities, backed by a proof of exploit, map directly to business risk.</td>
</tr>
<tr>
<td>Developer</td>
<td>• Is what I am coding secure?</td>
<td>• Promotes understanding of the importance of each finding and allows prioritization of remediation, as each finding is provided in context.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Eventually leads to improved use of SAST and improved secure coding standards.</td>
</tr>
<tr>
<td>Security</td>
<td>• Will we be able to detect vulnerabilities in production?</td>
<td>• Establishes an initial secure baseline for an application after it is deployed into production.</td>
</tr>
<tr>
<td></td>
<td>• How do I ensure application teams are aware of their security vulnerabilities?</td>
<td>• Provides an effective method to maintain the security of an application after it is deployed into production.</td>
</tr>
<tr>
<td></td>
<td>• How do I ensure previous vulnerabilities are not reintroduced?</td>
<td>• Provides operations, security and application owners with a single tool for assessing risk in pre- and post-production.</td>
</tr>
<tr>
<td>Operations</td>
<td>• What will be the impact on my network or system performance?</td>
<td>• Scans for systems I may have on the Internet, but which I am not aware of and may not have secured.</td>
</tr>
<tr>
<td></td>
<td>• How will remediation affect the application?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• What do any configuration changes do to my application?</td>
<td></td>
</tr>
</tbody>
</table>
In designing applications that remain secure after deployment as well as before, designers must consider infrastructure design and how they can repeat patterns from web apps that have already proved to be secure. Through automation, DAST can help an organization establish patterns that can be applied across the entire life cycle of the web app.

**Development**

Integration or system-level testing is the stage in the development process at which the application is tested along with configuration and other elements. DAST validates a) whether the security functionality is implemented in accordance with requirements, b) how it will behave when exposed to common application-level exploitation attempts, and c) the secure design decisions that shape the architecture.

DAST allows testing to be automated through APIs. A mandatory set of tests can be designed and automated to address the following root causes, which account for the Open Web Application Security Project (OWASP) Top 10 vulnerabilities:

- Input validation and sanitization
- Error checking and handling
- Robust authentication and session management
- Complete mediation of the application, which means there is only one way in or out of a given architecture. If the mediation point was an index.php page, for example, a user could access site content only by making requests through this page. Keep in mind, however, that DAST may not be the best approach for this purpose. SAST, and even penetration testing, are much better.
- Secure configuration of the application, including all components of the tiered architecture

---

2 The latest version of the OWASP Top 10 can be found at [www.owasp.org/index.php/Category:OWASP_Top_Ten_Project](http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)
**Deployment**

Deployment is potentially the moment an application will be the most secure, with the most complete configuration baseline being established, or it may be the most vulnerable, as some weaknesses in the underlying infrastructure may not have not been accounted for. Consider the following two steps as part of the formal fielding of a system, including updates to any of its components:

1. **Test the deployed application using DAST tools.** Conduct a final penetration test in the test (QA) environment to ensure that the application is as designed and all outstanding bugs have been identified and remediated. Then, move the application to production and conduct the same test(s) to ensure that the production environment has not introduced any new vulnerabilities. After successful testing, formally capture the application baseline.

2. **Continue monitoring the deployed application for a predefined period.** Establish automated tests to confirm that the deployed application will remain secure. Expand the boundaries of the scan to include other applications in production. What if the application was deployed before network upgrades?

For an enterprise, these steps serve as an important set of criteria to fulfill before the app is put into production. This is the point at which the organization could identify a deployed web service as insecure, for example, in addition to spotting potential vulnerabilities due to incompatibilities or conflicts with other production applications.

**Maintenance and Operations**

Web application maintenance does not include a “love it and leave it” option; applications must be monitored and maintained even after deployment. Web apps become, unavoidably, part of a larger environment that exposes them to dangers that may never have been fully modeled or tested in development environments. To remain secure, web applications must be tested in action in the environments in which they are actually deployed. DAST is one way to accomplish that testing.
Patterns for the use of DAST need to be established, including:

- **Monthly or quarterly health checks for applications.** These should be structured around the verification and validation of specific applications to ensure no new security risks have been introduced since the last check and the desired level of security remains intact. DAST tools can support automated testing across multiple applications in the environment.

- **Change management requests and audits.** Automation can establish a series of standard tests when approved changes in the configuration of the production environment occur, whether for an individual application or the entire infrastructure.

- **Discovery.** Unknown applications can introduce the risk of a data breach or failed compliance audits. This translates into damage to the brand, lost customers, greater audit costs and fines.

Finding public-facing web applications should be as simple as scanning a company’s IP range and seeing which web applications pop up; however, this is not always so simple.

Many corporate websites—significant numbers of which are run by business units rather than the IT group—may escape normal detection for that reason. In addition, the IT department may also host some services in the public cloud, such as Amazon Web Services. These would also be outside of the company’s IP range and difficult to detect.

Just because you didn’t know an application exists doesn’t mean it can’t do the type of damage you are trying to prevent. The data security officer of a state university received a report of a data breach from applications she had never authorized or evaluated, for example. Upon investigation she found an unauthorized SharePoint-based records-management application had been installed by the university president’s office. Because the installation had never been reported to her or her team, nor had it been listed among the department’s applications, there was no indication there was an application that needed to be monitored to prevent the leak of sensitive information until after the leak had already occurred.
Monitoring and assessment. Large organizations with hundreds of web apps usually can identify only specific time windows each month to perform in-depth vulnerability scans, scans that may not actually target the most critical assets. Enterprises need scalable tools that can perform lightweight scanning of these resources in parallel, to identify the most critical vulnerabilities (i.e., most easily exploitable). This ability to perform parallel assessments gives the organization visibility into its overall risk and allows it to evaluate which critical assets should be scanned in depth.

The discovery process may also provide quick wins. You may find assets that can be retired or shut down, such as those that may have been enabled for a short-lived but now defunct marketing campaign.

Scenario Recap

Let’s recap a fictitious but common scenario showing how DAST might help prevent an attack. See Table 3.

<table>
<thead>
<tr>
<th>Attack Stage</th>
<th>Prevention</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>An attacker identifies a web server with an information leak issue: Error messages are not suppressed in the public-facing website.</td>
<td>Configure the web server to not expose information leaks in production that give the attacker feedback on attacks.</td>
<td>Vulnerabilities would be patched, eliminating the web server version as an attack vector.</td>
</tr>
<tr>
<td>The attacker attempts to break into the web app via a SQL injection against the login form.</td>
<td>Use DAST during development (including updates to the web app) to identify the SQL injection flaw prior to fielding the app, allowing corrective actions to be taken regarding input validation and sanitation. Corrective action by the developer would be validated with DAST prior to deployment.</td>
<td>SQL injection flaws would be corrected during development and validated as corrected prior to deployment, eliminating SQL injection as a method of compromise.</td>
</tr>
<tr>
<td>The attacker is able to access sensitive user data as an administrator.</td>
<td>Use DAST during maintenance and operations to continually assess deployed web apps to identify this as an easily exploitable vulnerability.</td>
<td>Assessment would be followed by an in-depth scan of the web app and corrective action would be taken to harden the app, eliminating the vulnerability as an attack vector.</td>
</tr>
</tbody>
</table>
Measures to Communicate

DevSecOps must communicate throughout the life cycle of the application, realizing that security is a collaborative process. Some issues can be fixed only by developers (code flaws) while others must be handled by network operations (misconfigurations). Proper tools and processes are required to achieve this level of communication. DAST tools must integrate with existing development tools and environments (especially ticketing systems) and workflow processes. The following set of questions can help shape what is needed to establish a channel of communication from developers to operational staff, especially among those who are designated as configuration managers:

1. Do your developers, security staff and operators communicate regularly about production realities and the development status? Do you have tools that support such open and transparent exchanges?
2. Do you version deployment configuration and scripts along with your source code?
3. Do you have patterns for platforms and applications, designed jointly by development, operations and security?
4. Are these patterns based on reusable deployment configuration scripts?
5. Can you deploy an environment (platform and application) in one step?
6. Do you deploy your applications daily into production-like (test/QA) environments and verify them?
7. Do you link bugs and work items to changes in the application and configuration?
8. Do you associate tickets for production issues with relevant bugs opened for development to fix?
9. Do you have automated tests to validate your application and platform functions and characteristics?
10. Do you monitor software after deploying your application for anomalous results or performance?
11. Do you have a delivery pipeline exposed through a summary dashboard to assess delivery velocity?
How should organizations go about applying DAST to the SDLC and beyond? SANS suggests the following steps for a plan of action:

1. **Understand the need.** First, talk to members of your enterprise community. Define each major team role based on the approach to web application development your organization has adopted or is moving toward. Then, explore the expectations of each primary member of the team and how this affects buying criteria as well as how DAST, specifically, may or may not help them accomplish their expanded mission. How are developers trying to squeeze flaws out of code as they write it? How is the operations staff trying to harden apps after they’re running as cloud-based workloads? Keep in mind your organization’s goals and objectives. DAST focuses on real-world issues and can offer a faster path to success than SAST alone, especially if the organization’s priority is to identify and address the highest risk application vulnerabilities first or to quickly establish a secure coding culture that enforces best practices.

2. **Determine the rough model for using DAST.** This model doesn’t have to be perfect. You will be refining it, but you need to start somewhere. You should have a preliminary idea of where, when and how you would use the tools, how large the environment is that you are trying to manage, how many apps you are developing per year, and the timeline for fielding each project. A software development shop with many small projects will have very different requirements than those of a large enterprise that may be fielding a handful of very large projects, either for internal stakeholders or paying customers. You also need to understand the change, configuration and communication management processes your organization follows.

   Also, consider how the tools you select and configure may help you leverage your approach to DAST. A centralized, web-based scanner, for example, may make improvements in your security processes as the location of the scanner becomes your “approved scanning host.” This allows you to better use your IPS/IDS, as any scan coming from anywhere other than your centralized server is not approved.

3. **Conduct a preliminary cost-benefit analysis.** Based on this initial, rough model, do a quick cost-benefit analysis as to whether these services should be developed in-house, totally outsourced or adopted in combination. Also examine whether these tools should be purchased outright or implemented through security-as-a-service.
4. **Establish your needs.** Once you have decided on your course of action, it’s time to set up your procurement requirements—the checklist of what your criteria will be for purchase or internal setup.

Requirements normally break down into four categories:

- **Programmatic.** What is it going to take to manage the business activities related to DAST? What are the scope, schedule and resource items that will need to be tracked? Don’t underestimate the time and resources needed to undertake a discovery project if you are going to take this on in-house. At this stage, you will first identify all the public and private applications hosted by your organization, and then classify and prioritize all vulnerabilities.

- **Functional/technical requirements.** In some ways, this is the easiest part of the puzzle. Develop a comprehensive set of functional and technical requirements that meet your specific needs. Take the time to develop an evaluation plan. Ideally, you should assess the potential solution by actually exercising the tool against a known scenario. This will also allow you to compare the different tools you may be considering side by side. Don’t be afraid to talk with your counterparts in the industry to get their firsthand experience with vendors.

- **Services.** What does the vendor provide for support and training? If considering security-as-a-service (SecaaS), what are the service-level agreements? From talking with colleagues, does the vendor honor its SLAs? Then, what should be the nature of the services you want from a vendor? Once you know all of your applications, you may still have trouble scanning them all by yourself. You might look for a vendor that makes it easy to scan all of your applications at the same time, or within a short time window, by scanning in parallel. SecaaS vendors tend to have an advantage here, because you won’t have to set up a server farm for sporadic parallel testing.

- **Cost.** The lowest cost solution should not always win. Don’t be misled: Open source tools may be available at little or no cost, but these require extensive in-house resources or expensive consulting services to set up and use. A vendor’s solution may appear cost-viable, but the detailed services (and costs) may be buried in the fine print. If comparing on-premises versus SecaaS vendors, figure hardware, deployment and maintenance costs into the equation.

---

**OWASP Testing Guidance**

OWASP considers all its guidance for testing web apps to treat the application as a black box. Testing is separated into Phase 1 (Passive) and Phase 2 (Active). A DAST solution needs to be evaluated in light of the following elements in Phase 2:

- Information Gathering
- Configuration and Deployment Management Testing
- Identity Management Testing
- Authentication Testing
- Authorization Testing
- Session Management Testing
- Input Validation Testing
- Error Handling
- Cryptography
- Business Logic Testing
- Client-Side Testing
5. **Revisit and revise.** Based on the analysis you have just done, don’t be afraid to revisit your approach by reviewing an updated cost-benefit model.

6. **Structure your procurement around what you decide based on your analysis in the previous steps.** Details of procurement are too complex for the scope of this paper; we will expand upon that discussion and provide more detailed guidelines in a companion paper.
Conclusion

The concepts and requirements around application security testing, including DAST, are changing as application security evolves from traditional linear approaches (i.e., waterfall) toward more continuous approaches as demanded by DevSecOps—whose goal is to develop applications at the speed of business.

Challenges include:

- **Configuration.** As the mobile and cloud computing ecosystem continues to explode, applications become more and more dependent on the same web-based services, and loosely coupled system architectures continue to expand, the security model becomes more diverse and complex to maintain. Automation is not a luxury; it’s a necessity.

- **Change management.** Effective processes are key to reducing risk and driving remediation: How can we tell that things are getting more secure? Are we actually reducing the risk through using DAST?

- **Communication.** Clear lines of communication are needed. More people are doing similar kinds of work in parallel, scanning apps as well as servers for vulnerabilities. The feedback loop now includes non-development types—business owners, consumers and executive management—who understand neither development nor security. Remember that a lack of communication inhibits delivery, especially the accelerated delivery demanded by DevSecOps.

Ultimately, you will need to quantify the differential between the risk due to the exposure of vulnerabilities and the effectiveness of the security tests in mitigating the security risk. Understanding both will make it easier to make decisions affected by variations in the cost of security testing activity and the testing tools adopted.
Barbara Filkins, a senior SANS analyst who holds the CISSP and SANS GSEC (Gold), GCIH (Gold), GSLC (Gold), GCCC (Gold), GCPM (Silver) and GLEG (Gold) certifications, has done extensive work in system procurement, vendor selection and vendor negotiations as a systems engineering and infrastructure design consultant. She is deeply involved with HIPAA security issues in the health and human services industry, with clients ranging from federal agencies (Department of Defense and Department of Veterans Affairs) to municipalities and commercial businesses. Barbara focuses on issues related to automation—privacy, identity theft and exposure to fraud, as well as the legal aspects of enforcing information security in today’s mobile and cloud environments.

SANS would like to thank this paper’s sponsor:
## Upcoming SANS App Sec Training

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS 2020</td>
<td>Orlando, FL</td>
<td>Apr 03, 2020 - Apr 10, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Mentor Session - DEV522</td>
<td>Novi, MI</td>
<td>Apr 21, 2020 - May 21, 2020</td>
<td>Mentor</td>
</tr>
<tr>
<td>SANS Amsterdam May 2020</td>
<td>Amsterdam, Netherlands</td>
<td>May 11, 2020 - May 18, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Silicon Valley - Cupertino 2020</td>
<td>Cupertino, CA</td>
<td>Jun 22, 2020 - Jun 27, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Copenhagen August 2020</td>
<td>Copenhagen, Denmark</td>
<td>Aug 24, 2020 - Aug 29, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Network Security 2020</td>
<td>Las Vegas, NV</td>
<td>Sep 20, 2020 - Sep 27, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Online</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s Only</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>