Understanding and Selecting Runtime Application Self-Protection

Version 1.3
Released: October 21, 2019

Securosis, L.L.C. 515 E. Carefree Blvd. Suite #766 Phoenix, AZ 85085 T 602-412-3051 info@securosis.com www.securosis.com

Author's Note

The content in this report was developed *independently of any licensees*. It is based solely on our research, material which was originally posted on the Securosis blog but has been enhanced, reviewed, and professionally edited.

Licensed by Contrast Security

Contrast Security is the world's leading provider of security technology that enables software applications to protect themselves against cyberattacks, heralding the new era of self-protecting software. Contrast's patented deep security instrumentation is the breakthrough technology that enables highly accurate assessment and always-on protection of an entire application portfolio, without disruptive scanning or expensive security experts. Only Contrast has sensors that work actively inside applications to uncover vulnerabilities, prevent data breaches, and secure the entire enterprise from development, to operations, to production. More information can be found at www.contrastsecurity.com or by following Contrast on Twitter at @ContrastSec

Copyright

This report is licensed under Creative Commons Attribution-Noncommercial-No Derivative Works 3.0.

http://creativecommons.org/licenses/by-nc-nd/3.0/us/

Table of Contents

Introduction 2

Defining RASP 3

Use Cases 4

Functional Requirements 5

Business Requirements 6

Technology Overview 8

How the Technology Works 8

Integration 11

Buyers Guide 15

About the Author 18

About Securosis 19

Introduction

During our 2015 DevOps research, developers consistently turned the tables on us, asking dozens of questions about embedding security into their development process. We were surprised to discover how much developers and IT teams were taking larger roles in selecting security solutions, working to embed security products into tooling and build processes. Just as they use automation to build and test product functionality, they automate security too.

But the biggest surprise was that every team asked about RASP, Runtime Application Self-Protection. Each team was either considering RASP or already engaged in a proof-of-concept with a RASP vendor. This was typically in response to difficulties with existing Web Application Firewalls (WAF). Most teams still carry significant "technical debt" in the form of security vulnerabilities, which require runtime application protection. Since 2017 we have engaged in *over 200 additional conversations* on what gradually evolved into 'DevSecOps' — with both security and development groups asking about RASP, how it deploys, and benefits it can realistically provide. Our conversations solidified and fleshed out the requirement for developer-centric security tools which offer the agility developers demand, provide metrics prior to deployment, and either monitor or block malicious requests in production.

And we can say unequivocally that there is a strong demand for the features WAF promised a decade ago, but RASP provides and has surpassed.

Research Update

Our previous RASP research was published in the summer of 2016. Since then Continuous Integration for application build processes has become the norm, and DevSecOps is no longer considered a new idea. Developers and IT folks have embraced it as a viable and popular approach for producing more reliable application deployments. But it has raised the bar for security solutions, which now need to be as agile and embeddable as developers' other tools to be taken seriously. The rise of DevSecOps has also raised expectations for integration of security monitoring and metrics. We have witnessed the disruptive innovation of cloud services, with companies pivoting from "We are not going to the cloud." to "We are building out our multi-cloud strategy." in three short years. These disruptive changes have highlighted the deficiencies of WAF platforms — both lack of agility and inability to go "cloud native".

Simultaneously we have been watching advancements in RASP technologies and deployment models. RASP has evolved detection techniques and deployment options to meet broader market demands, and have gotten to the point where they accurately detect attacks that will succeed from those that will not, and then block them. We even see application security teams use RASP in pre-production pen testing and red team exercises to ensure application security is fully operational prior to pushing the application live. With all these changes it has become increasingly difficult to differentiate one RASP platform from another, so we decided to refresh our research. We will cover new approaches, deployment models, and revised selection criteria for buyers.

Defining RASP

Runtime Application Self-Protection (RASP) is an application security technology which embeds into an application or application runtime environment, examining requests at the application layer to detect attacks and misuse in real time. RASP products typically contain the following capabilities:

- Unpack and inspect application requests, typically in the application context
- Monitor and block application requests; products now analyze fully formed requests at multiple enforcement points, sometimes even alter requests to strip out malicious content
- Full functionality accessible through RESTful API
- Protect against all classes of application attacks, and determine if an attack would succeed
- Pinpoint the module, and possibly the specific line of code, where a vulnerability resides
- Instrument application functions and report on usage

For those familiar with Web Application Firewalls, RASP may sound similar in function, but they approach solving similar problems in very different ways! The differences have to do with effectiveness, and RASP offers a number of detection approaches which enable both more accurate attack identification and much less effort to install and maintain than most firewalls. It's a question of security being fully integrated within an application, so it runs anywhere the application does, and scales right along with the application — regardless of whether that's running on bare metal, virtualized, or in the cloud. And RASP offers developers a comfortable medium: security software that behaves as their own code. As one of our readers astutely noted, "If it's not working inside the application, it's not RASP." In this paper we will discuss why development organizations are looking for new solutions, as well as the fundamental advantages RASP provides as an application security platform.

Use Cases

The primary function of RASP is to protect web applications against known and emerging threats. In some cases it is deployed to block attacks at the application layer, before vulnerabilities can be exploited, but in many cases RASP tools process a request until they detect an attack, and then block the forbidden action.

Astute readers will notice that these are basically the classic use cases for Intrusion Detection Systems (IDS) and Web Application Firewalls (WAFs). So why look for something new if other tools already provide the same benefits? The answer is not in what RASP does, but in *how* it works in context of the application, where it can determine if an attack is relevant, making it more effective in a wide range of scenarios.

Let's delve into about what clients are asking for so we can bring this into focus.

Primary Market Drivers

RASP is a relatively new technology, so current market drivers are tightly focused on addressing security needs of two distinct "buying centers" which have been largely unaddressed by existing security applications. We discovered this important change since 2017 through hundreds of conversations with buyers, who expressed remarkably consistent requirements. The two buying centers are security and application development teams. Security teams are typically looking for a reliable WAF replacement without burdensome management requirements, while development teams asked for a security technology to protect applications within the framework of existing development processes.

The security team requirement is controversial, so let's start with some background on WAF functions and usability, which is essential for understanding the problems driving firms toward RASP.

Web Application Firewalls typically employ two methods of threat detection; **blacklisting** and **whitelisting**. Blacklisting is detection — and often blocking — of *known attack patterns* spotted within incoming application requests. SQL injection is a prime example. Blacklisting is useful for screening out many basic attacks against applications, but new attack variations keep showing up, so blacklists cannot stay current, and attackers keep finding ways to bypass them. The many variants of SQL injection are the obvious illustration.

But whitelisting is where WAFs provide their real value. A whitelist is created by watching and *learning* acceptable application behaviors, recording legitimate behaviors over time, and preventing any requests which do not match the approved behavior list. This approach offers substantial advantages over blacklisting: the list is *specific* to the application monitored, which makes it feasible to enumerate good functions — instead of trying to catalog every possible malicious request — and therefore easier (and faster) to spot undesirable behavior.

Unfortunately, developers complain that in the normal course of application deployment, a WAF can never *complete* whitelist creation — 'learning' — before the next version of the application is ready for deployment. Their argument is that WAFs are inherently too slow to keep up with modern software development, so they devolve to blacklist enforcement. Developers and IT teams alike complain that WAF is not fully API-enabled, and that setup requires major manual effort. Security teams complain they need full-time personnel to manage and tweak rules. And both groups complain that when they try to deploy into Infrastructure as a Service (IaaS) public clouds, the lack of API support is a deal-breaker.

Customers also complain of deficient vendor support beyond basic "virtual appliance" scenarios — including a lack of support for cloud-native constructs like application auto-scaling, ephemeral application stacks, templating, and scripting/deployment support for the cloud. As application teams become more agile, and as firms expand their cloud footprints, traditional WAF becomes less useful.

To be clear, WAF can provide real value — especially commercial WAF "Security as a Service" offerings, which focus on blacklisting and some additional protections like DDoS mitigation. These typically run in the cloud as a proxy service, often filtering requests "in the cloud" before they pass on to the application or RASP solution. But this WAF deployment model is limited to a 'Half-a-WAF' status — lacking the whitelisting capability. Traditional WAF platforms continue to work for less agile on-premise applications, where the WAF has time to build and leverage a whitelist. So existing WAF is largely not being "ripped and replaced", but it is largely unused in the cloud or more agile development teams.

Security teams are looking for an *effective* and *easier to manage* application security tool to replace WAF. They need to cover application defects and technical debt, because not every defect can be fixed quickly in code.

Developer requirements are more nuanced: they cite the same end goals, but tend to ask which solutions can be fully embedded into existing application build and certification processes. To work with development pipelines, security tools need to go the extra mile, protecting against attacks *and* accommodating the disruption underway in the developer community. Solutions need to be as agile as modern application development, which generally starts with compatible automation capabilities. It needs to scale with the application, typically by being bundled into the application stack at build time. It should 'understand' the application and tailor its protection to the application runtime. Security tools should not require that developers be security experts. Development teams working to "shift left" to get security metrics and instrumentation earlier in their process want tools which work in pre-production as well as production.

RASP offers a distinct blend of capabilities and usability options which make it a good fit for these use cases. This is why, over the last three years, we have been fielding several calls *each week* to discuss it.

Functional Requirements

The market drivers mentioned above change traditional functional requirements — the features buyers are looking for.

- **Effectiveness:** This seems like an odd buyer requirement. Why buy a product which does not actually work? The short answer is 'false positives' that waste time and effort. The longer answer is that **many** security tools don't work well, produce too many false positives to be usable, or require so much maintenance that building your own bespoke tool can seem like a better investment. RASP typically provides full capabilities without the need for run-time learning of application functions, offers broader coverage of application threats by running in the application context, and can run in blocking mode. This last is especially important in light of current application threats, such as the Capital One cloud hack via SSRF (Server Side Request Forgery).

- **Application Awareness:** As attackers continue to move up the stack, from networks to servers to applications, attacks tailored to application frameworks and languages are increasingly the norm. RASP differentiates on its ability to include application context in security policies. Many WAFs offer 'positive' security capabilities (whitelisting valid application requests), but embedding within applications provides additional application access and instrumentation to RASP. Further, some RASP platforms assist developers by identifying suspect modules or lines of code. For many development teams, better detection capabilities are less important than having RASP pinpoint vulnerable code.

- **API Support & Automation:** Most of our readers know what Application Programming Interfaces (API) are and how they are used. Less well-known is the rapidly expanding need for programatic interfaces in security products, thanks to application delivery disruptions brought by cloud services and DevOps. API are how we orchestrate building, testing,

and deployment of applications. Security products like RASP offer full platform functionality via API — sometimes as build server plug-ins or even as cloud services — enabling software engineers to work with RASP in their native metaphor. And they provide agents, containers, or plug-ins which work within the application stack.

- **Coverage & Language Support:** This is still RASP's biggest issue, one which hampered adoption for its first few years of existence. RASP platforms, to provide security for different languages or platforms, require custom rules and logic to detect attacks. Most provide full support for core platforms like Java and .NET; beyond that support still gets a bit spotty for Python, PHP, Node.js, Ruby, etc. Another part of the problem is the complexity of environments — not just the server side, but the client side as well. Over the last few years we have watched an expanding universe of frameworks, client-side utilities, web-facing API, and changing fashions for data encoding. RASP needs to parse user supplied inputs, handle diverse clients running JavaScript and Angular.js, micro-service architectures, and possibly multiple versions of API, all at the same time. The diversity of application environments makes it challenging for RASP vendors to provide full support. We encourage you to use the breadth of platform support as a sign of maturity of the RASP product you are considering as newer additions tend to offer limited support.

- **Pre-Deployment Validation:** The earlier in the production cycle errors are discovered, the easier — and cheaper — they are to fix. This is especially important for expensive or dangerous technologies such as cars and pacemakers. So testing in general, and security testing in particular, works better earlier in the development process. Rather than relying on vulnerability scanners and penetration testers *after* application deployment, more and more application security testing is performed pre-deployment. Of course this is possible with other application-centric tools, but RASP is easier to build into automated testing, can often determine which parts of an application have vulnerabilities, and is commonly used during red-team exercises and pre-production 'blue/green' deployment scenarios.

Business Requirements

We want to add some additional nuance to the use cases above, and look at some of business drivers for this type of technology. These topics lead directly into selection criteria, below.

- **Integration with Trouble Ticketing:** Pretty much every security platform must now integrate with internal systems for tracking security issues as they are discovered, investigated, fixed, tested, and then re-deployed. Trouble ticketing systems form the backbone of these efforts, controlling task assignment, so this type of integration is essential. We still get user requests for integration with Syslog, SIEM, and Splunk — usually in addition to ticketing software. Fortunately these features are provided by most vendors.

- **Compliance:** WAF was widely adopted for PCI-DSS — essentially contractually mandated security for systems which process credit card data — because it was specifically listed as an appropriate control for web-facing applications. As WAF's popularity wanes while the PCI requirement remains, RASP is being used as a compensating control. In fact some RASP vendors have performed third party verification with the major QSA organizations and are vetted as a suitable control for application security.

- **Virtual Patching:** Most firms have a ton of "technical debt" because applications and their platforms contain more security flaws than the development team can fix in short order. Keeping development, QA, and production versions of underlying software up to date with basic security patches is a challenge, much less fixing all the vulnerable code. RASP helps by detecting which versions of application libraries are out of date, if vulnerable sub-modules are used to understand if patching is necessary, and also by offering "virtual patches" which block attacks against remnant vulnerabilities. Most development and operations teams do not track the never-ending stream of security patches, so having a tool automate discovery and reporting is helpful. Seldom is this function in firms' top 5 requirements, but it is still common on RFPs (Requests For Proposals).

- **DevOps & Metrics:** In DevOps they say that if you do not have metrics, you're just another guy/gal with an opinion. Metrics are key to determining if you have a problem, where you should direct your resources, and whether your security investments are actually working. RASP can catalog application functions and open source usage, understand the correct number and type of parameters for each API, and then apply policies within the code of the running application. But it also can understand runtime code paths, server interaction, framework nuances, library usage, and custom code. This helps security teams visualize security issues in code, and tailor how they wish to respond. Deployed in pre-production it enables discovery of defects prior to production rollout. Finally, several RASP platforms also offer IAST (Interactive Application Security Testing), or deploy at the developer's desktop, to function *before* code gets checked in, finding issues earlier in the development process.

Technology Overview

It is time to discuss technical facets of RASP products — including how the technology works, how it integrates into an application environment, and advantages of different integration options. We will outline important considerations, such as platform support, which impact selection. We will also consider a couple aspects of RASP technology which we expect to evolve over the next couple years.

How the Technology Works

Over the last couple years the RASP market has settled on a couple basic approaches — with a few variations to enhance detection, reliability, or performance. Understanding the technology is important for understanding the strengths and weaknesses of different offerings.

- **Native API Callbacks:** In this deployment model the system inserts sensors or callbacks at key junctions within the application stack to observe application behavior between your application code, libraries, frameworks, and the underlying operating system. This approach is typically implemented using native application profiler/instrumentation API to monitor runtime application behavior. When a sensor is hit RASP gets a callback, and evaluates the request against policies relevant to the request and application context. For example database queries are examined for SQL Injection (SQLi). But they also provide request deserialization 'sandboxing' to detect malicious payloads, along with what I call 'checkpointing': a request which hits checkpoint A but bypasses checkpoint B can be considered hostile with a high degree of confidence. These approaches provide far more advanced application monitoring than WAF, with nuanced detection of attacks and misuse. But full visibility require monitoring of all relevant interfaces, with a performance and scalability cost. Customers need to balance thorough coverage against performance.

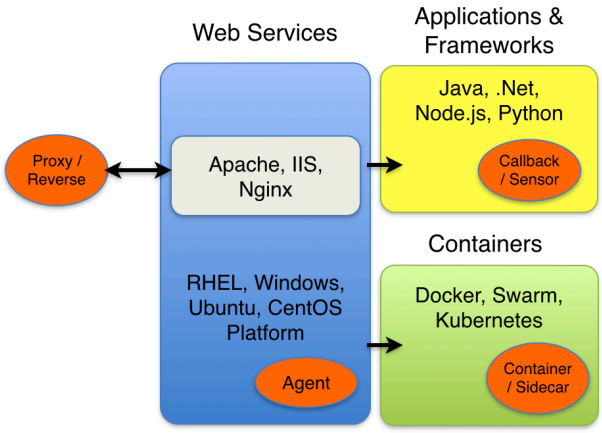
- **Servlet Filters & Plugins:** Some RASP platforms are implemented as web server plugins or Java Servlets, typically installed in Apache Tomcat, JBoss, or Microsoft .NET to process requests. Plugins filter requests before they execute functions such as payment transactions, applying detection rules to each request on receipt. Requests which match known attack signatures are blocked. This is effectively the same functionality as a WAF blacklist, with added capabilities such as lexical analysis of inbound request structures. This is a simple way to retrofit protection into an application environment; it is effective at blocking malicious requests without the deep application understanding possible using other integration approaches.

- **Library or JVM Replacement:** Some RASP products are installed by replacing standard application libraries and/or JAR files, and at least one vendor offers a full replacement Java Virtual Machine. This method basically hijacks calls to the underlying platform into a custom application. The RASP platform passively observes application calls to supporting functions, applying rules as requests are intercepted. For example in the case of JVM replacement, RASP can alter classes as they are loaded into memory, augmenting or patching the application and its stack. Like instrumentation integration, this approach provides complete visibility into application behaviors and user requests.

Some customers prefer this option as effectively automated platform patching, but most customers we speak with find replacement of the application runtime in production unacceptable.

- **Static Hybrid:** Like many firewalls, several RASP platforms can deploy as a reverse proxy. In one case a novel variant couples a proxy, an instrumentation module, and a limited static analysis scanner. Essentially it generates a Code-Property-Graph — like a static analysis tool — to build custom security controls for all application and open source functions. This approach requires full integration into the application build pipeline to scan all source code. It then bundles the scan result into the RASP engine as the application is deployed, effectively providing an application-specific functionality whitelist. The security controls are tailored to the application with excellent code coverage — at the expense of full build integration, the need to regularly rebuild the CPG profile, and some added latency for security checks.

Several small companies have come and gone over the last couple years, offering a mixture of application logic crawlers (i.e: Dynamic Application Security Testing, or DAST) rule sets, application virtualization to mimic the replacement model listed above, and runtime mirroring in a cloud service. The full virtualization approach was interesting, but being too early to market and being dead wrong in approach are virtually indistinguishable. Still, over time I expect to see new RASP detection variations, possibly in the area of AI, and new cloud services for additional support layers.



Detection

RASP attack detection is complicated, with multiple techniques employed depending on request type. Most products examine both the request and it's parameters, inspecting each component multiple ways. The good news is that RASP is far more effective at detecting application attacks than its predecessors. Unlike other technologies which use signature-based detection, RASP watches application behavior and external references, maps application functions and third-party code usage, maps execution sequences, deserializes payloads, and applies polices accordingly. Things like sandboxing requests to see how they play out, use of semantic analysis to detect misuse, input tracing to detect artificial introduction of code and behavioral analysis to go along with traditional tools like signatures and IP address reputation. This not only enables more accurate detection, but also improves performance by optimizing which checks are performed based on request context and code execution path. Enforcing rules at the point of use makes it much easier to both understand proper usage and detect misuse.

Most RASP platforms employ structural analysis as well. They understand what framework is in use, and the common exploits of the framework. As RASP has access to the entire application stack, it can detect variations in third-party code libraries — roughly comparable to a vulnerability scan of an open source library — to detect use of outdated code. RASP can also quickly vet incoming requests and detect injection attacks. There are several approaches — one uses a form of tokenization (replacing parameters with tokens) to verify that a request matches its intended structure. For example tokenizing clauses and parameters in a SQL query can quickly detect a ‘FROM’ or ‘WHERE’ clause with more tokens than it should have, indicating the query has been altered.

Blocking

Blocking is one area where the new generation of RASP platforms excel. WAF typically makes a determination to block based upon incoming HTTP requests. It relies upon signatures — as discussed earlier — to detect know malicious or known good patterns, blocking based upon this prior knowledge. This type of inspection lacks both application context — visibility of all functions a given request will touch — and the ability to let a malicious request 'play out' to the point where there is no doubt about intent. This new generation of RASP tools offers both, with multiple detection techniques like sandboxing, semantic analysis, input tracing and behavioral analysis to go along with signatures. When you couple multiple detection techniques — each being appropriate for specific classes of attacks — with multiple enforcement points, you have a quantum leap in accuracy over traditional WAF. It is safe to say that RASP is not simply WAF in the application, rather an entirely improved method of blocking application attacks. Our interviews have shown a vastly higher percentage of RASP customers run in full blocking mode, and this jump has accelerated over the last two years.

Performance and Scalability

RASP embeds within the application or its stack, so it scales with the application. For example if the application scales out copies on multiple server instances, RASP will scale along with it. If deployed on virtual or cloud servers, RASP benefits from added CPU and memory resources right alongside the application.

RASP enforcement rules — both how they operate and the number of checks — may impact latency and performance. More and deeper request analyses strengthen security but increase latency. If third-party threat intelligence is not cached locally, or external lookups are used, latency increases. If sensors or integration points only collect events and pass them to an external server for analysis, added services are likely to increases latency. As with all security products, don't trust vendor numbers — run your own tests, in your environment, with representative traffic. Fortunately vendors have applied considerable engineering resources to performance over the last couple years, significantly reducing latency issues.

Instrumentation

Security teams often want visibility into application security, and it is increasingly common for them to use scans in the build pipeline, pre-production, and production to develop metrics and gain visibility into application security posture. This shows where they need more resources and helps gauge the effectiveness of deployed resources.

One huge advantage of RASP is that it can instrument application usage and defect rates continuously, during the runtime. Part of this capability is derived from its ability to catalog application functions, understand the correct number and type of parameters, and apply policies within running application code. But RASP also can understand runtime code paths, server interactions, open source libraries, framework nuances, library utilization, and custom code. This offers advantages when tailoring detection rules, such as enabling specific policies to detect attacks against the Spring framework. RASP can be configured to block specific attacks against older versions of libraries, providing a type of virtual patching. This also provides non-security benefits — helping Quality Assurance and Operations teams see how code is used, providing a runtime map which can identify performance bottlenecks and unused code.

Integration

This section will outline how RASP integrates into a technology stack, into both production deployment and application build processes. We will show what it looks like, and why it's important for newer application security technologies to fit into these steps. We will close with a discussion of how RASP differs from other security technologies, and some advantages and tradeoffs of differing approaches.

As we mentioned in the introduction, our research into DevOps unearthed many questions on RASP. They came from non-traditional buyers of security products: application developers and product managers. Their teams, by and large, were running Agile development processes. They wanted to know whether RASP could effectively block attacks and fit within their existing processes.

I analyzed hundreds of customer call notes over the last couple years to come up with the top 7 customer RASP questions, roughly in order of how often they came up.

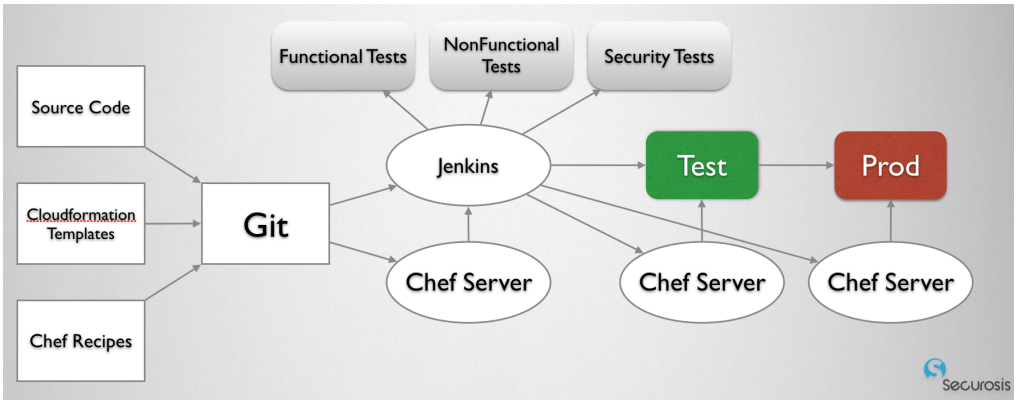
- *"We presently use static analysis in our build process, but we are looking for solutions that scan code more quickly and would like a 'preventative' option. Can RASP help?"*
- *"Development releases code twice daily, which is a little scary, because we only scan with static analysis once a week [or month]. Is RASP suitable for providing protection between scans?"*
- *"We would like a solution that provides some 0-day protection at runtime, and sees application calls."*
- *"Development is moving to a microservices architecture, but WAF only provides visibility at the edge. Can we embed monitoring and blocking into microservices?"*
- *"We have many applications with substantial security technical debt, our in-house and third-party code is not fully scanned, and we need XSS/CSRF/Injection protection. Should we look at WAF or RASP?"*
- *"We are looking at a 'defense in depth' approach to application security, and want to know if we can run WAF alongside RASP."*
- *"We want to 'shift left': move security as early as possible, and embed it into our application development process. Can RASP help?"*

These questions clearly illustrate how changes in application deployment, its increasing speed, and the declining applicability of WAF, are driving interest in RASP.

Security tools that fit this model are actively being sought by development teams. They need granular API access to functions, quick production of test results, and delivery of status back to supporting services.

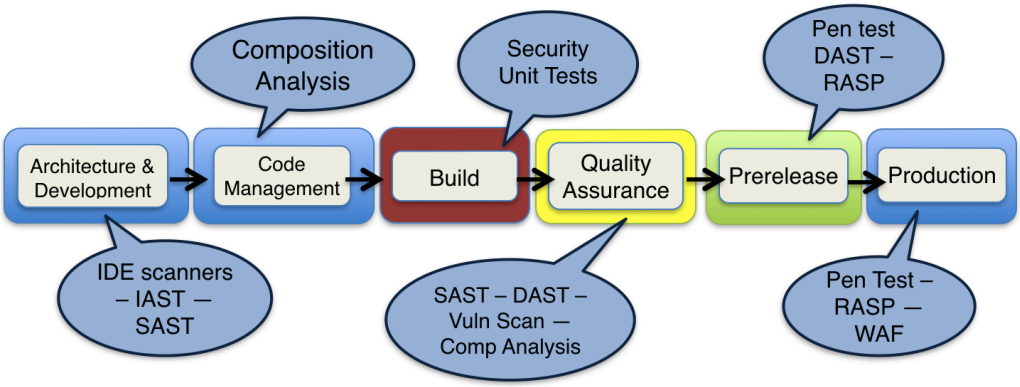
Build Integration

The majority of firms we spoke with are leveraging automation to provide Continuous Integration — essentially automated building and testing of applications as new code is checked in. Some are farther down the DevOps path, and have reached Continuous Deployment (CD). To address this development-centric perspective, the diagram below illustrates a modern Continuous Deployment / DevOps application build environment. Each arrow could be a script automating some portion of source code control, building, packaging, testing, or deployment.



Process Integration

The build pipeline offers a mechanical view of development, but a process-centric view offers a different perspective on where security technologies can fit. The following diagram shows different logical phases in the process of code development, each staffed by people performing a different role (*e.g.* architects, developers, build managers, QA, release management, IT, and IT Security). The diagram's step-by-step nature may imply waterfall development, but do not be misled — these phases apply to *any* development process, including spiral, waterfall, and agile.



The above graphic illustrates major phases of application development. The callouts map common types of security tests at specific phases within Waterfall, Agile, CI, and DevOps frameworks. Keep in mind that automated deployment and DevOps are still relatively young. Many current security tools were built before rapid and automated deployment existed or were well established. Older products are typically too slow, some cannot focus tests on new code, and others lack API support. So orchestration of security tools — basically what works where — is still maturing. The time each type of test takes to run, and the type of result it returns, drives where it fits into the phases above.

RASP is designed to be bundled into application stack and is part of the application & application delivery process. RASP components can be included as part of an application — typically installed and configured under a configuration management script so RASP starts up as part of the application stack. RASP is commonly used in both pre-release / pre-deployment phase, as well as the traditional use case in production to block attacks. In pre-release testing it is used to instrument an application to ensure penetration tests, red team tests, and other synthetic attacks are accounted for prior to 'going live'. In production RASP performs monitoring and blocking.

- **Pre-release Testing:** This is exactly what it sounds like: RASP is used when the application is fully constructed and going through final tests prior to launch. In this role RASP can be deployed several ways. It can be configured to monitor only, using application tests and instrumenting runtime behavior to learn how to protect the application. Alternatively RASP can monitor security tests attempting to break the application, with the tool performing security analysis and reporting on its results. Development and Testing teams can see whether RASP detects tested attacks. Finally, RASP can be deployed in full blocking mode to see whether security tests are detected and blocked, and how they impact user experience. This provides opportunity to change application code or augment RASP rules before the application goes into production.

- **Production Testing:** Once an application is placed in a production environment, either before actual customers are using it (using Blue-Green deployment) or under customer load, RASP can be configured to block malicious application requests. Regardless of how the RASP tool designed (whether based on embedded runtime libraries, servlet filters, in-memory execution monitoring, application instrumentation, or virtualized code paths), it protects applications by detecting attacks at runtime. This model essentially provides execution path scanning, monitoring all user requests and parameters. Unlike technologies which block requests at the network or web proxy layer, RASP inspects requests at the application layer so it has full access to the application's inner workings. Working at the API layer provides better visibility to determine whether a request is malicious, and blocking capabilities focused on specific attack variants.

- **Runtime Protection:** RASP is not just for testing, but for full runtime protection and blocking of attacks. Not just the typical Cross-Site Scripting (CSS), SQL Injection (SQLi), Cross-Site Request Forgery (CSRF), or other drive-by attacks — but malicious code execution, weak authentication, improper session management, use of vulnerable third-party software, and misuse of custom code as well. And it can detect platform-specific attacks (*e.g.* .NET and Java) as well as attacks on application frameworks (*e.g.* Spring, Struts, and Play). RASP is uniquely positioned to protect applications from a broad range of attacks. Again, monitoring and protecting at the application layer includes subtle contextual and behavioral clues which can provide substantial improvement in detection with few or no false positives.

- **Development & IAST:** Some RASP platforms also double as IAST tools. This means they can be "shifted left" to the point of code creation, providing testing benefits even before code is checked in or built. Testing in development is less common, because few developers are in the habit of testing prior to code check-in, or incentivized to do so. But such testing tools have advantages for finding more security flaws earlier, and can help teach developers how to build secure code. Coupling RASP with IAST offers runtime contextual detection with early discovery, providing development and exposure analysis teams an option on how to handle remediation.

To WAF or not to WAF

Most organizations we spoke with were primarily seeking something to work within existing development pipelines. WAF's lack of API for automatic setup, the time needed to learn application behavior, and most importantly WAF's inability to pinpoint vulnerable code modules, were all cited as reasons WAF failed to satisfy developers. These requests

came from more ‘Agile’ teams, more often building new applications than maintaining existing platforms. But we consistently heard that RASP meets a market demand unsatisfied by other application security technologies.

Note that WAF was already in place at almost every client we spoke with, and none planned to remove their existing WAF. Some cited PCI-DSS as their reason for leaving it, some mentioned “defense in depth”, and others were not yet comfortable enough with RASP to “rip & replace”. So most current RASP users run it alongside WAF.

It is important to recognize that these technologies are complementary rather than incompatible. There is absolutely no reason you can’t run RASP behind your existing WAF. Some organizations use cloud-based WAF for front-line protection, while embedding RASP into applications. Some use WAF to provide “threat intelligence”, DDoS protection, and network security, while using RASP to fine-tune application security — often supplanting WAF’s whitelisting capability. Still others double down with overlapping security functions, much the way organizations use layered anti-spam filters, accepting redundancy in exchange for broader coverage or unique benefits from each product. WAF platforms have a good ten-year head start, with broader coverage and very mature platforms, so firms are loath to throw away WAF until RASP is fully proven, or more accepted as a compensating control under regulations such as PCI-DSS.

Buyers Guide

It's time to take a look at RASP selection. For our 2016 version of this paper, the market was young enough that a simple list if features was enough to differentiate one platform from another. But current platform maturity makes top-tier products more difficult to differentiate.

The last section discussed principal use cases, and then technical and business requirements. Depending upon who is driving your evaluation, your list of requirements may be based on either. With those driving factors in mind — and we encourage you to refer back as you go through this list — here is our recommended process for evaluating RASP. We believe it will help you identify which products and vendors fit your requirements, and avoid some pitfalls.

Define Needs

- **Create a Selection Committee:** We hate committees too, but the simple fact is that when RASP effectively replaces WAF (whether or not WAF is actually going away), RASP affects not only the security team, but also development, compliance, and operations. So it's important to include someone from each of those teams (to the degree they exist in your organization) on the committee. Ensure that anyone who could say no, or subvert the selection at the 11th hour, is on board from the beginning.

- **Define Systems and Platforms to Monitor:** Is your goal to monitor select business applications or all web-facing applications? Are you looking to block application security threats, or only for monitoring and instrumentation to find security issues in your code? These questions can help refine and prioritize your functional needs. Most firms start small, figure out how best to deploy and manage RASP, then grow over time. Legacy apps, Struts-based applications, and applications which process highly sensitive data might be your immediate priorities — you can monitor other applications later.

- **Determine Security Requirements:** The committee can be incredibly beneficial for understanding your *real* requirements. Sitting down with the entire selection team is likely to adjust your perception of what a platform needs to deliver and the priorities of each function. Everyone may agree that blocking threats is a top priority, but developers might feel that platform integration is next highest, while IT wants trouble-ticket system integration, but security wants support for all languages and platforms in use. All this while marketing and sales teams demand better performance and customer experience. The list of competitions priorities will be deep, and some stakeholders will have inordinate political cloud, so create lists of "must have", "should have", and "nice to have".

- **Define:** Here you basic needs — determined earlier — are translated into specific technical features, and any additional requirements are considered. With this information in hand you can document requirements to produce a coherent RFI.

Evaluate and Test Products

- **Issue the RFI:** Larger organizations should issue an RFI though established channels and contact a few leading RASP vendors directly. If you are in a smaller organization start by sending your RFI to a trusted VAR and email a few RASP vendors which look appropriate. A Google search or brief contact with an industry analyst can help understand who the relevant vendors are.

- **Define the Short List:** Before bringing anyone in, match any materials from vendors and other sources against your RFI and draft RFP. Your goal is to build a short list of 3 products which can satisfy most of your needs. Also use outside research sources (such as Securosis) and product comparisons. Understand that you'll likely need to compromise at some point in this process, as it's unlikely any vendor can meet every requirement.

- **The Dog & Pony Show:** Bring vendors in, but instead of generic presentations and demonstrations, ask them to walk you through specific use cases which match your expected needs. Provide them a vulnerable application or two — either one of yours or open source project code like WebGoat — on a private network and let them demonstrate capabilities. Have your internal pen testers or red teasers engage the application and show what is discovered. This is critical because they are very good at showing eye candy and presenting the depth of their capabilities, but having them attempt to deploy and solve *your* specific problems will help narrow down the field and finalize your requirements.

- **Finalize RFP:** At this point you should completely understand your specific requirements, so you can issue a formal RFP.

- **In-house Deployment Testing:** Set up several test applications if possible; we find public and private cloud resources effective for setting up private test environments to put tools through their paces. This exercise will very quickly show how easy or hard each product is to use. Try embedding the product into a build tool and see how much of the heavy lifting the vendor has done for you. Since this reflects day-to-day effort required to manage a RASP solution, deployment testing is key.

- **In-house Effectiveness Testing:** You'll want to replicate key capabilities in house. Build a few basic policies to match your use cases, then violate them. This is both to test the effectiveness at finding real vulnerabilities, but it will also give you an idea if the tool is 'noisy' or produces false positives. Many firms replay known attacks, or use penetration testers or red teams to hammer test applications to ensure RASP detects and blocks the malicious requests they are most worried about. Many firms leverage OWASP testing tools to exercise all major attack vectors and verify that RASP provides broad coverage. Make sure to tailor some of their features to your environment to ensure customization, UI, and alerts work as you need. Are you getting too many alerts? Are some indicated attacks not actually processed by the application, thus pose no risk? Do alerts contain actionable information so a developer can do something with them? Put the product through its paces to make sure it meets your needs.

Selection and Deployment

- **Select, Negotiate, and Purchase:** Once testing is complete, take the results to your full selection committee and begin negotiations with your top two choices. — assuming more than one meets your needs. This takes time, but it is *very* useful to know you can walk away from a vendor if they won't play ball on pricing, terms, or conditions. Pay close attention to pricing models — are they per-application, per-instance, per-server, or some hybrid? When you expand your RASP footprint you should already know it will not cause your bill to skyrocket.

- **Implementation Planning:** Congratulations, you've selected a product, navigated the procurement process, and made a sales rep happy. Now the next stage begins: you need to plan deployment. That means making sure of

details: lining up resources, getting access/credentials to devices, locking in an install schedule, and even the logistics of getting devices to the right locations. No matter how well you execute on selection, unless you implement flawlessly and focus on quick wins and getting immediate value from the RASP platform, your project will fail.

- **Professional Services:** In some cases initial setup is where the majority of the work takes place. Unlike WAF, day-to-day maintenance of RASP tends to be minor. So some vendors, either directly or through partners, can help with integration and templating initial deployment.

I can hear the groans from small to medium-sized business looking at this process and thinking this is a ridiculous amount of detail. We created a granular selection process for you to pare down to your organization's requirements. We wanted to make sure to capture all the gory details organizations need for successful procurement. Our process is appropriate for a large enterprise, but a little pruning can easily make it a good fit for a small group. That's the great thing about process: you can change it however you see fit, at no expense.

For many organizations, who simply cannot fix every application security issue before release, implementing a (RASP) platform is a requirement. The sheer volume of existing application security defects, the rate of discovery of new attacks, and the velocity of modern development, there is often no other viable option. Regardless of whether your driver is compliance, operational security, or something else, we need to react to threats — without weeks to fix, test, and deploy. Quick and efficient handling of attacks, and useful instrumentation to determine which parts of an application are vulnerable, are critical for Security and Deployment to tackle application security. RASP provides an effective tool to bridge short-term requirements with long-term application security goals.

About the Author

Adrian Lane, Analyst and CTO

Adrian Lane is a Senior Security Strategist with Securosis, and VP of Development at DisruptOps. With over 25 years of industry experience, he brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services. Our services include:

- **Primary research publishing**: We publish the vast majority of our research for free through our blog, and package the research as papers that can be licensed for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements, and follow our Totally Transparent Research policy.
- **Cloud Security Project Accelerators**: Securosis Project Accelerators (SPA) are packaged consulting offerings to bring our applied research and battle-tested field experiences to your cloud deployments. These in-depth programs combine assessment, tailored workshops, and ongoing support to ensure you can secure your cloud projects better and faster. They are designed to cut months or years off your projects while integrating leading-edge cloud security practices into your existing operations.
- **Cloud Security Training:** We are the team that built the Cloud Security Alliance CCSK training class and our own Advanced Cloud Security and Applied SecDevOps program. Attend one of our public classes or bring us in for a private, customized experience.
- **Advisory services for vendors**: We offer a number of advisory services to help our vendor clients bring the right product/service to market in the right way to hit on critical market requirements. Securosis is known for telling our clients what they NEED to hear, not what they want to hear. Clients typically start with a strategy day engagement, and then can engage with us on a retainer basis for ongoing support. Services available as part of our advisory services include market and product analysis and strategy, technology roadmap guidance, competitive strategies, etc. Though keep in mind, we maintain our strict objectivity and confidentiality requirements on all engagements.
- **Custom Research, Speaking and Advisory**: Need a custom research report on a new technology or security issue? A highly rated speaker for an internal or public security event? An outside expert for a merger or acquisition due diligence? An expert to evaluate your security strategy, identify gaps, and build a roadmap forward? These defined projects bridge the gap when you need more than a strategy day but less than a long-term consulting engagement.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors. For more information about Securosis, visit our website: <http://securosis.com/>.