Prevoty

Whitepaper

A Guide to Runtime Application Self-Protection (RASP)

Prevoty, Inc. HQ
11911 San Vicente Blvd #355
Los Angeles, CA 90049

prevoty.com
info@prevoty.com
310.499.4983
@prevoty

Table of Contents

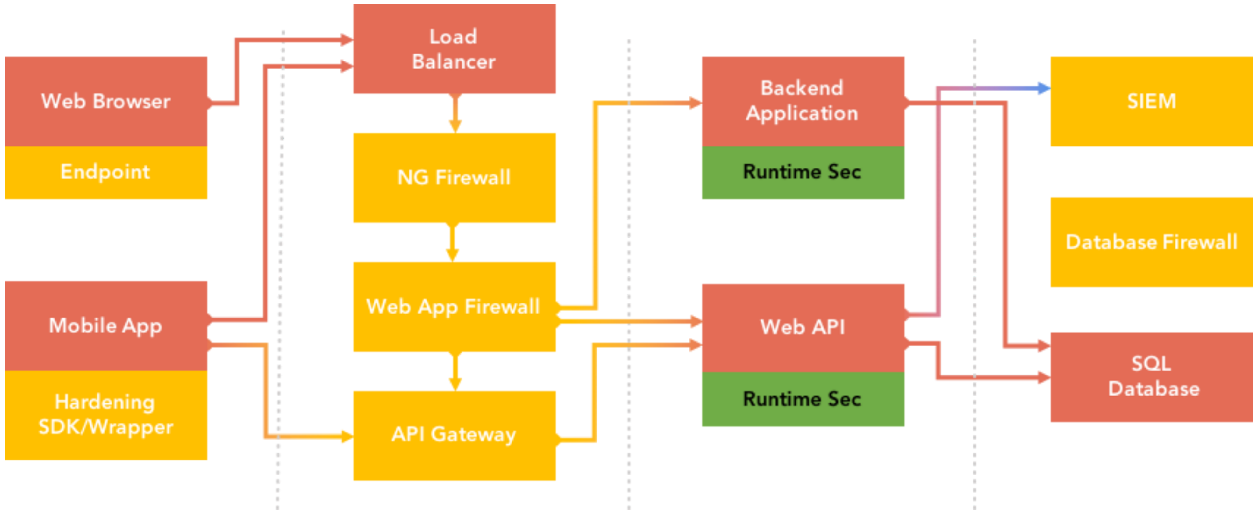
Copyright © 2017 Prevoty. All Rights Reserved.

1.0 The Application Security Ecosystem

Application security has taken a shape of its own in recent times, and there is still no single standard definition for application security. This guide aims to identify the various pieces of application security within an end-to-end ecosystem as well as provide a deeper dive into the newest innovation that focuses on securing applications at runtime: runtime application self-protection (RASP).

The Enterprise Architecture Framework

Figure 1: A typical enterprise using web-based applications as their business



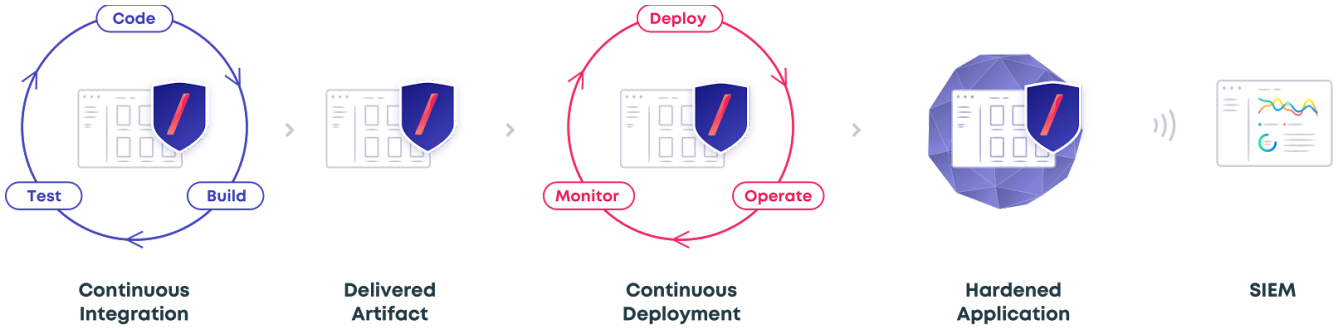
Most enterprises today have both legacy and new software that drive their business, and these applications sit within a complex environment spanning the network, application, database, and operating system. The older the enterprise, the more fragmented its environment. Fragmentation occurs in the Developer or DevOps environment with its multiple languages (JAVA, .NET, Ruby, HTML5 etc) and multiple databases. Fragmentation is also seen in application security, with multiple and often disparate layers of security controls -- ranging from Static, Dynamic, and Interactive security testing to runtime application self-protection (e.g. DAST, SAST, IAST, and RASP), as well as perimeter based protection with network firewalls and web application firewalls (WAFs).

Copyright © 2017 Prevoty. All Rights Reserved.

Fragmented Organizations & Control

A successful application security program requires multiple domains and resources to interwork and exchange information. The diagram below shows the various components that come together to make up application security. In additions to the technical components (applications, infrastructure tools, security controls), there are two major organizations involved: Builders and Defenders.

Figure 2: The Secure Software Development Life Cycle (SSDLC)
as it aligns with automated DevOps processes



Builders

Teams typically reside in the business, technology and product units. Their goal is to push applications into production as quickly as possible under the pressure of quickening release cycles. The environment in which they work is undergoing three major disruptions:

1. **IT Changes** - The rise of software defined data centers, virtualization, on-demand virtual machines (versus buy/spin up servers), containerization and cloud applications are resulting in a new set of challenges that distract Builders from the main goal: developing applications fast.

2. **DevOps** - Most application development teams are now joining forces with their IT/CIO organization to move to an agile DevOps environment. While DevOps helps with speed, agility of development, and feedback, it introduces new risks to the traditional software development model.

3. **DevSecOps** - Builders are often forced to work closely with the Defenders or security admins to fix or mitigate their vulnerability backlogs. While processes like SDLC, scanning tools like xAST, and/or penetration testing all point to issues within applications, the pressures of time, budget, performance, and the challenges of remediation often force developers to push code to production in spite of known risks.

Copyright © 2017 Prevoty. All Rights Reserved.

As such, Builders are asserting a few simple requirements for security and control:

1. *A security tool should have zero impact on application performance*
2. *The CISO's office must prioritize vulnerabilities by criticality and provide real attack data from production environments (as opposed to theoretical analyses of code)*
3. *As the CIO's office implements architectural changes in order to move to a software-defined data center, it must make any infrastructure changes transparent to applications*

Defenders

Teams typically reside within the CSO/CISO organizations. Their goal is to holistically address enterprise security and reduce risk. Historically, the scope of these teams have involved management of penetration testing programs and application security testing solutions, as well as overseeing WAF and RASP technologies. While they have access to SIEM technologies, most analytics are still too basic or early in their infancy to provide actionable data or significantly improve defense and response. Information security executives, who oversee the Defenders' efforts, are facing four major obstacles:

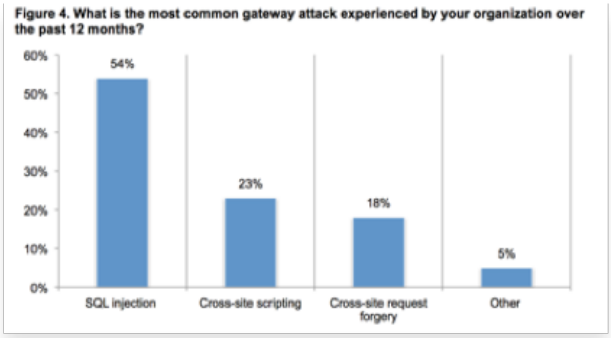
1. **Applications are distributed and complex, and application security approaches are also fragmented.** Virtualization, microservices and the cloud make applications & data ubiquitous -- impossible to monitor consistently and accurately. And security strategies are mainly just a patchwork of marketplace tools and services that are often not as flexible or portable.

2. **Attack volumes are increasing, and the sophistication of common attack vectors are bypassing existing controls.** There is a need for simple, pragmatic, application security controls that work to prevent common, modern attack vectors such as those listed in the OWASP Top 10, which account for the majority of attacks:

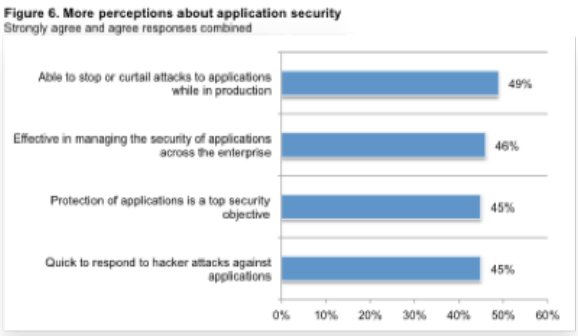
Copyright © 2017 Prevoty. All Rights Reserved.

Figures 3 and 4: The Rising Importance of Application Security

2015: Top 3 cover 95% attacks



Application security now rising as a top business objective



Source: *Ponemon Institute Enterprise Survey. The Increasing Risk to Enterprise Applications. Nov 2015.*

3. **Critical production applications are at risk due to unmitigated vulnerabilities,** Defenders understand the inevitable exposure to risk created when Builders are forced to push applications into production with known vulnerabilities. Security controls & vulnerability mitigation tools must keep up with DevOps, but they are not.

4. **There's no visibility into runtime security events.** Despite analysis at the pre-production stage or intelligence gathered at the network layer, Defenders can't access useful runtime attack data.

What can be done about this deep fragmentation? The rest of this document dives into the specifics of newer runtime technologies along with their potential to confront challenges and increase collaboration across Development, Security and IT teams with one goal in mind: protecting applications in production.

How RASP Complements the Security Ecosystem

Gartner first defined runtime application self-protection (RASP) as a security technology built or linked into an application runtime environment to control execution and prevent real time attacks.[1] Before RASP entered the security market, the industry's offerings provided protections on the network layer and on the host, but lacked active protection at the application layer. With the exception of a WAF, there were no production environment protections to provide controls at runtime. With 28 technologies and growing, the application security space isn't trivial -- nor is it well defined.

RASP is an emerging tool that typically falls under the "Runtime Testing and Protection" or "Application Self Protection" category. As a newcomer, RASP is important not only in its own function but also in how it differs from and/or interacts with the other technologies in the ecosystem. Most of the time, RASP

[1] *Gartner IT Glossary: Runtime Application Self-Protection (RASP). http://www.gartner.com/it-glossary/runtime-application-self-protection-rasp*

Copyright © 2017 Prevoty. All Rights Reserved.

supplements and even improves the effectiveness of other tools. The following table aims to define the various technologies in the application security ecosystem that potentially interact with RASP, and how:

Technology	Interaction
Web application firewall (WAF) or next-generation web application firewall	Since most firewalls are inserted in the data and control path as passive elements to ensure compliance, RASP offers a critical layer of attack prevention behind a WAF or next-generation WAF. WAFs monitor and block traffic by applying rules. Several next-generation network firewall solutions (Dell Sonicwall, Palo Alto Networks) include some basic WAF functions and/or application awareness as they sit as a bump in the wire on the architecture. Since most firewalls use patterns and heuristics, they offer limited assurance against application attacks. Rather, they are useful as the first layer of defense. RASP is not a proxy nor does it block traffic; instead, it neutralizes malicious or malformed payloads and specific inputs to serve as a last line of defense. Because it sits within the application, it has access to attack details, including unsanctioned database activity. Leading solutions from vendors like F5, Imperva enable this today and view RASP as a complementary technology. *See more in "WAF vs. RASP" on Page 8.*
Dynamic application security testing (DAST)	Leading DAST solutions like that of Whitehat Security provide visibility into vulnerabilities. The interaction between DAST and RASP is simple. RASP can be used to prioritize vulnerabilities before running any testing tool, guiding developers on how best to minimize risk and ensuring effective secure coding practices. RASP can be also be installed on an application in production and turned on in protection mode. Attacks and abnormal inputs are cleaned and mitigated in real time, and proof of runtime threat activity is reported to a dashboard like WhiteHat Security's Sentinel.
Security incident and event management (SIEM)	RASP sits inside the application and enriches attack data with critical insights into the where any transformation or exfiltration attempt took place, where, and by what bad actor, greatly improving a SIEM's security analytics with runtime intelligence. Most leading vendors like Splunk allow for logs and files to be visually displayed as well as run through data analytics engines to determine patterns. All RASP solutions should have the ability to generate monitoring and protection logs in the following formats: CEF, LEEF and JSON. The first two target SIEMs like ArcSight, QRadar and Nitro. By generating JSON logs, a RASP's output can be ingested by more modern/flexible SIEMs such as Splunk or even Elasticsearch.

Copyright © 2017 Prevoty. All Rights Reserved.

WAF vs. RASP: Isn't a WAF enough?

A web application firewall (WAF) normally sits in front of web applications, inspecting incoming HTTP request traffic for known attack payloads and abnormal usage patterns. When a suspicious payload or usage is detected, the WAF can either report the violation or report and block the request. However, because the WAF is unaware of how the application will actually process the payload or input data, it is quite common for the WAF to identify suspicious payloads or usage patterns incorrectly, resulting in false positives. The remedy for this is often extensive "tuning" of input data filters and the burden of lengthy "learning" modes wherein the application remains unprotected. To further complicate the situation, modern application development has strongly shifted to a continuous deployment model, in turn creating a constantly shifting attack surface. It is nearly impossible for traditional WAFs to keep up, using a filter and usage pattern approach.

Modern RASP technologies are easier to deploy, provide a more uniform set of controls regardless of programming language, and perform with higher accuracy.

The RASP approach differs from traditional WAFs because it is tightly coupled with the application code traditionally susceptible to malicious exploit. Unlike a WAF, RASP can automatically adapt to any language or environment and uses contextual awareness -- not blacklists and whitelists -- to detect threats. Instead of blindly guessing that a particular payload will (or will not) be able to exploit an unknown part of the application code before the data is sent to the application, RASP technology inspects the complete (and often-times transformed data) in the context of how the application will use it -- if and only if the application will attempt to use the data.

RASP technologies, because of their proximity to vulnerable code constructs inside the running application, typically have significantly fewer false positives than WAFs. Whereas WAFs simply erect walls in front of the application, RASP protects the application from the inside out with its unique capability to perform context-sensitive detection. Its instrumentation of the runtime environment enables vulnerability mitigation without access to the source code. This reduces false positives and improves visibility into vulnerabilities — including weaknesses previously unknown to the organization.

2.0 RASP Value & Use Cases

RASP entered the market as a progressive alternative to the application security status quo. So what problem(s) does RASP solve, and how? Below, we outline RASP's main values to the enterprise:

Copyright © 2017 Prevoty. All Rights Reserved.

SMARTER RESPONSES

Visibility into Hidden Runtime Attacks

Response teams do not have insight into application security events in production and thus cannot accurately correlate pre-production vulnerability findings with runtime attack data. Furthermore, there is even more limited visibility into access or exfiltration attempts for applications and databases moving to the cloud. There is also a significant amount of noise generated by testing tool results, application firewall activity, and vulnerability reports. RASP can help security operations and application development teams filter through the noise and better allocate resources using runtime intelligence.

RASP delivers correlated network, application and database security logs for smarter, faster responses and powerful visibility into an organization's actual runtime exposure to risk. Application security monitoring using RASP is a new capability that has been designed to give enterprises the ability to determine which applications are actually under attack in real time (and how) -- effectively improving risk management and remediation efforts. In short, application monitoring via RASP answers the following questions:

Specifically, new application-level insights and forensics can also catch authentication, authorization and transactional fraud. Detailed information on all database queries issued by specific applications allow for detailed audit trails and support root cause analysis for data breaches.

BETTER DEFENSES

Real-Time Vulnerability Mitigation

Remediation efforts are unable to verify and mitigate 100% of application security vulnerabilities found in the secure software development life cycle. Nevertheless, enterprises are often pushing applications into production with known vulnerabilities that cannot be remediated due to a lack of access to the code base, legacy frameworks, and other roadblocks. These vulnerability exception procedures can be costly and extremely risky.

RASP implementations are uniquely positioned to help enterprises protect applications at runtime, neutralizing known vulnerabilities and protecting against previously unknown threats and zero-day

Copyright © 2017 Prevoty. All Rights Reserved.

attacks. Depending on the nature of the deployment, RASP can also transform or block content and database queries so that everything the application processes is safe.

Many organizations use RASP to embed a last line of defense that travels with the application, whether in the cloud, on-premise, in pre-production or in production. As an automated, technical control for compliance requirements, RASP takes the pressure off of development by performing real-time vulnerability mitigation.

FASTER RELEASES

Scalable, DevOps-Friendly Application Security

Until RASP, application security and DevOps were frequently at odds. The increasingly distributed, agile nature of application development and deployment has made preventing a breach complex and challenging. Virtualization, containerization, microservices and the cloud make applications and data ubiquitous -- impossible to monitor consistently and accurately across different platforms and environments. Worse yet, pre-production testing requirements create bottlenecks in the software development process. Some vulnerabilities can even block release, which can be problematic in a rapid-release, Continuous Integration (CI) or Continuous Deployment (CD) DevOps cycle.

Some RASPs can be woven directly into the DevOps build/deployment process so that applications can safely and be deployed into production without any delays. RASP's detection and prevention features can be embedded directly into every application release as part of the automated CI/CD pipeline, which means applications can automatically self-protect no matter where it sits in the SSDLC. Security tests are no longer tied to release schedules and remediation is prioritized using production attack data.

Attack Coverage: OWASP Top 10 and more

RASP can protect against sophisticated threats and zero-days, which are included but not limited to the examples listed in the table below:

Cross-Site Scripting	Weak Authentication (Basic Auth)	HTTP Response Splitting
Cross-Site Request Forgery	Broken Session Management	HTTP Method Tampering
DOM Cross-Site Scripting	Weak Browser Cache Management	Unvalidated Redirects
SQL Injection	Logging Sensitive Information (credit card numbers and email addresses	Path Traversal
XML Injectlon	Insecure Transport Protocol	Unauthorized Markup (trying to inject prohibited HTML tags)
JSON Injection	Uncaught Exception	Unauthorized Media (trying to inject links to prohibited media sites)
Database Access Violation (advanced SQL Injection)	Insecure Direct Object References	
Command Injection	Security Misconfiguration	

Copyright © 2017 Prevoty. All Rights Reserved.

Link Spam XML External Entity Injection	Sensitive Data Exposure Unvalidated Redirects and Forwards	

Common Use Cases for RASP

These following scenarios are highlighted to showcase more specifically how security admins, DevOps teams and developers use RASP to address some of the challenges facing application security today:

USE CASES	
1 - Reduce vulnerability backlog	Application security testing (AST) tools help uncover vulnerabilities in pre-production. But in most enterprises, the backlog keeps increasing over time. Applications perform critical business and transactional functions and are frequently pushed into production with known vulnerabilities. With RASP, more than 95% of the backlog may not ultimately need to be remediated or fixed by developers as RASP can neutralize the threat in case of attack in production. Results in significant efficiency gains as well as resource and cost savings.
2 - Real-time visibility into production attacks	Implementing a RASP in monitoring mode allows for full visibility into real-time attacks (as opposed to potential known vulnerabilities). This enriched runtime threat data can be sent to SIEMs and logging tools to inform both developers and other ecosystem products like WAFs or next-generation firewalls. This also cuts through the noise from other pre-production tools by exposing only critical runtime security events -- answering questions like, “What are the most attacked production apps/assets? What were the data exfiltration events that originated from outside the firewall?” With improved forensics and post-mortem threat analysis, security operations can more accurately correlate vulnerabilities and direct remediation efforts for improved development and compliance.
3 - Faster application releases with scalable DevSecOps	Application security and the software development lifecycle are often at odds. If installed directly into the automated DevOps funnel via Continuous Integration and Continuous Development tools, RASP delivers security more seamlessly so that controls are always “baked” into every release by default. Organizations can push applications into production faster without worrying about security vulnerabilities. This capability reduces operational friction and fosters more trust and collaboration across these teams, helping achieve SecDevOps / DevSecOps alignment and fluidity.

Copyright © 2017 Prevoty. All Rights Reserved.

4 - Provide Runtime Intelligence for DevOps	In addition to providing insights on which applications are the most secure/insecure, RASP can provide critical intelligence for DevOps teams. Similar to other tools developers use during the design/test phase (e.g. New Relic for performance and DAST/SAST for code scans, RASP may be used to provide visibility into what the application will do at runtime (e.g. database calls, file read/writes, login/logout, failed logins, lateral calls from production applications, versions and frameworks used, etc.)
5 - Protect legacy applications	In enterprises where applications are the business, protecting legacy apps that drive revenue is a critical requirement. Most of these legacy apps are written in older languages and do not have active development or support to fix vulnerability. RASP allows for protecting these without the need for developers or support.
6 - Last line of defense in a layered security model	If a WAF or next-generation firewall is the first line of defense, a RASP is the last line of defense. Today, applications mostly rely on external protection like WAF or IPS (Intrusion Prevention Systems), and there is a need to build security features into the application so that it can protect itself at runtime. An application instrumented with RASP would be more powerful than external devices which have only limited context on the logic, behavior and execution of the application. RASP is an integral part of an application run time environment and can be implemented as an extension of the Java debugger interface. It can detect an attempt to write high volume data in the application run time memory or detect unauthorized database access. It has real-time capability to take actions like terminate sessions, raise alerts etc. WAF and RASP can work together in a complementary way. WAF can detect potential attacks and RASP can actually verify it by studying the actual responses in the internal applications.
7 - Optimize the Secure Software Development Lifecycle (SSDLC)	Enterprises should continue to use dynamic and static testing technologies, and complete the secure software development life cycle (SSDLC) by protecting their applications with RASP while in production. RASP-enabled attack intelligence will improve and optimize the effectiveness of the SSDLC to ensure tight resource assignments and clear milestones for remediation. For instance, RASP plug-ins and SDKs can be an effective part of a proactive Secure Coding Training program. Developer training programs can make a positive impact on the number of vulnerabilities introduced during coding, but it is often a challenge to provide a simple answer to the question: “Which ones should I fix? Should a whitelisting or blacklisting approach be used? Which characters should be eliminated or allowed in the input data? What about encoding? Obfuscated data? Performance?” A simple and uniform answer would be to fix critical vulnerabilities that are prone to (or have shown) exploit in production, and to snap the RASP plug-in into the application’s deployment package or insert RASP API calls at key points in the code base to mitigate risk for remaining lower-tier vulnerabilities. Either approach is easy

Copyright © 2017 Prevoty. All Rights Reserved.

	to do during code construction without significant changes to how the application is written or tested.
8 - Improve Security Operations & Response	RASP empowers security operations center (SOC) teams with application-layer insights to make faster, easier, smarter decisions, shortening the investigation lifecycle and improving perimeter controls. RASP can unify network, application and database security intelligence into a pre-correlated report, enabling action based on actual (not theoretical) risk, such as proactively blocking IP addresses of “bad actors” without the risk of false positives. Traditionally, when there is an event, perimeter controls provide data on the source IP, destination IP, and the payload the triggered the signature. Then, the security team would then have to spend significant amounts of time validating and testing to answer “Is the attack real? How should we respond?” RASP can build reports and visualizations for real-time events with extremely low false positive rates, feeding that live attack data into a SIEM or other logging tool, revealing when the database is returning abnormal data sets so security managers can visualize production environment application threats and correlate with other data sources. This process eliminating time wasted on lower-tier investigation and analysis and allowing to jump directly to the appropriate response teams. Until RASP, SOC managers could not correlate pre-production vulnerability findings with runtime attack data. Understanding traversals and movement of exploits across applications and databases can help security teams make more informed responses for faster forensics to catch authentication, authorization and transactional fraud.
9 - Protection for applications anywhere & everywhere	Application architectures are evolving. Modern on-demand and infrastructure-as-a-service (IaaS) providers are gaining popularity because they enable business gains for better collaboration and security.. For instance, virtualization and migration to cloud services increase infrastructure security efficiencies by creating high-performance productivity clusters and user-friendly experiences. As such, applications and their data are now ubiquitous, impossible to monitor consistently and accurately. However, this means that security must be flexible and portable. It must be compatible not only with old and new programming languages, but also web application frameworks and microservices, support for on-premise, cloud and containerized deployments, as well as a direct integration with a wide array of code scanners, data logging tools, and SIEMs. RASP can provide visibility into access or exfiltration attempts for applications & databases moving to the cloud and across microservices. RASP lives and travels within the application and logging all runtime security events. Database activity is monitored from within the application for complete insights into app-level behavior. Applications stay protected no matter where they are, and production intelligence can be analyzed in SIEMs and used for enhanced database activity monitoring.

Copyright © 2017 Prevoty. All Rights Reserved.

10 - Reduce AppSec risk & increase compliance	RASP can serve as a compensating control for unremediated (or impossible to remediate) vulnerabilities that would otherwise undergo a costly compliance exception process. This is often the case due to a lack of access to the code base, legacy frameworks, and other roadblocks and time pressures. .With appropriate runtime protection and depending on the configuration, PCI compliance can be achieved in a way that is fast, accurate, and simple to maintain. *See the following sections for more reading on fulfilling compliance requirements and the value of RASP for PCI.*

Fulfilling Compliance Requirements

Depending on the implementation, RASP can provide critical technical controls for organizations attempting to meet stringent regulatory security and data privacy compliance standards -- thanks to a few key functions:

- RASP provides data validation mechanisms to prevent the exploitation of potentially vulnerable coding constructs in software. Data that flows into and through an application can be inspected by a RASP to protect from known, common application layer attacks and -- depending on the methodology -- zero-day threats
- Incorporating a RASP during application development is a simple and consistent way to implement the security and visibility capabilities needed in a Secure Development Lifecycle for both custom-built and off-the-shelf software applications. Whether deployed during the SDLC or DevOps process, RASP security controls can travel with the application and always remain “on”
- RASP also logs security incidents and user actions (user identity, type of event, date and time, success/failure, origination, and name of affected system component) for improved forensics and post-mortem correlations

A Closer Look: PCI and RASP

In the case of the Payment Card Industry Data Security Standard (PCI DSS), RASP can supply an automated control for a number of requirements (see table below):

6.3	Develop internal and external software applications securely
6.5	Address common coding vulnerabilities in software development processes

Copyright © 2017 Prevoty. All Rights Reserved.

6.6	Ensure applications are protected against known attacks
10.2	Implement automated audit trails
10.3	Record audit trail entries
11.4	Use intrusion detection / prevention techniques

3.0 RASP Technology Implementation

RASP in Action: Passive and Active

All RASP technologies should be able function in two different yet complementary modes: monitoring and protection.

1. When in **passive monitoring** mode, a RASP solution should utilize very limited application resources such as CPU and memory (RAM). It should also add minimal latency. While monitoring, a RASP should be able to generate similar logging events as if it were in **passive** mode. This allows organizations to build or access a security analytics report or “heatmap” of where real-world attacks are hitting the application.

2. When in **active protection** mode, a RASP solution should still utilize limited application resources to detect threats while automatically mitigating attacks in real-time and preventing database exfiltration. It should not require significant resources to tune or configure, or require cumbersome rule sets or definition lists. It should add minimal latency to an application. While in **active** mode, a RASP should generate actionable intelligence about real-world attacks, as well as what action was performed to neutralize the malicious or malformed payload.

Technical Components: Analysis and Implementation

RASP has two unique technical components: the security **analysis** plus the **implementation** of the application-level analytic processor. A technical evaluation of any RASP must discuss the merits and the weaknesses of each component both in isolation and in conjunction.

1. Application Threat Analysis

How security attacks are detected, computed, and subsequently mitigated is one of the most important attributes of RASP because it impacts accuracy and performance as well as implementation. For

Copyright © 2017 Prevoty. All Rights Reserved.

instance, a common issue with application security controls is the prevalence of false positives and false negatives, which drain resources and create an unwieldy amount of noise.

The four main methodologies for attack computation are pattern matching, heuristics, data flow analysis and language-theoretic security (LANGSEC). Every RASP solution performs threat analysis using a different approach (and sometimes a blend). A brief overview of each of the four methodologies and their sensitivity of its respective alerts is outlined the following page:

METHOD	OVERVIEW	FALSE POSITIVES	FALSE NEGATIVES
Pattern Matching	Pattern matching uses string literals and regular expressions to determine if a payload is safe. Example: checking if a SQL query contains comment characters like '#'.	**High** Blindly adding attack patterns can cause applications to accidentally break and identify false positives. Attack patterns must be evaluated by developers and QA teams by hand and using automated tests that exercise the entire application.	**High** If attack patterns do not exist in the corpus, then the result will be a false negative. Diligent pattern maintenance is important.
Heuristics	Heuristics uses multi-criteria analysis to statistically identify problems without defining them. Example: detecting anomalies in HTTP request/response lifecycles looking for abnormal events.	**High** Anomaly detection has a high propensity to stop known good traffic. As an example, periodic scheduled internal jobs (i.e. cron) that access web services is outside the "normal" bounds and requires specific IP whitelisting.	**High** Anomaly detection relying on statistical methods can be subverted by increasing the samples of bad behavior. Since attack traffic is not labeled safe or malicious via supervised machine learning, payloads with high frequencies will get through (normal behavior).
Data Flow Analysis	Data flow analysis uses language provided instrumentation APIs to track how variables and data flows through an application. Example: watching for HTTP variables that make their way into SQL queries.	**High/Moderate** Flow analysis has the ability to create a high number of false positives due to the variability of how applications are developed. As an example, an internal development practice for interpolating variables may not be a real security issue.	**Moderate** Flow analysis has the potential to generate modern false negatives. As an example, if an application is constructed where information flows differently (e.g. out of band producer/consumer queues) then flow analysis will miss an attack.

Copyright © 2017 Prevoty. All Rights Reserved.

Language Security (LANGSEC)	LANGSEC is the process of formally understanding how data such as content payloads, database queries, operating system commands, etc. will execute in an environment. Example: understanding if a database query contains a tautology, contradiction or attempting to access an invalid column.	**Low** Since LANGSEC relies on building formal grammars of languages, the number of false positives is significantly reduced.	**Low** Since LANGSEC relies on building formal grammars of languages, the number of false negatives is significantly reduced.

2. Analytic Processor Implementation

First, let's outline the three different approaches to performing runtime application security analysis and mitigating risk in production:

A.	B.	C.
Use a WAF or proxy to analyze all traffic for known security threats	**Instrument an application via agents/modules to inspect data in production**	**Replace the virtual machine itself with one that performs security functions**

RASP services are typically implemented using method **B** or **C** depending on several influencing factors such as the provider, performance requirements, language support, available network and service resources, and the intended result.

Instrumentation of an application to perform security functions (in this case, runtime self-protection) involves modifying the application itself by adding code, either manually via a Software Development Kit (SDK) or by a frameworks-based plug-in. In addition to providing Java and .NET plugins in the form of agents and modules, RASP solutions should allow for manual invocation via SDKs or software libraries. This enables modern enterprises that are actively practicing secure software development to use secure coding libraries to target specific vulnerabilities.

Whether using a plugin-based agent or module for direct framework integration or SDKs that can be called from a variety of coding languages to invoke self-protection functions, a RASP-based product should be easy to deploy, maintain and control. From a DevSecOps perspective, RASP solutions should be able to be controlled and automated from a centralized location or on an ad-hoc basis. Once deployed, RASP should be a fast, distributed system comprised of a number of modular services that parse and validate all incoming data without any dependencies on definitions, patterns, regular expressions, taint analysis or behavioral learning.

Copyright © 2017 Prevoty. All Rights Reserved.

RASP solutions should not require a command and control server to operate. However, should a server exist, it should provide the ability to enable and disable RASP features and switch between monitor and protect modes without requiring an application server to restart. RASP servers should be able to deploy on premise, in virtual environments and in public/private clouds. A RASP solution should be easy to package into an automated build/deployment process.

Finally, RASP agents should be able to attach to both legacy and modern applications. This attachment is work that can be done by both developers and operations team. No knowledge of application behavior should be required for attachment to be successful.

A Closer Look: Instrumentation via Agents/Modules vs. VM Replacements

All RASP solutions that target applications executing on the Java Virtual Machine (JVM) or the .NET Common Language Runtime (CLR) must go through fully published instrumentation and profiling APIs that have existed for years. As a concrete example, the Java agent specification, specifically used for instrumenting applications, has been available since Java 1.5 (released in the early 2000's).

An alternative approach for performing instrumentation is to replace the underlying virtual machine (VM) with one that can perform security monitoring and protection. While valid from a technical perspective, this approach has significant challenges:

- *How can you prove that a VM replacement does not introduce new bugs for your existing applications? In order to prove this, an organization has to run significant regression tests across all their applications that will now be targeting this replaced VM.*

- *What if a bug or vulnerability is detected in the replaced VM? Today, organizations rely on companies like Microsoft and Oracle to perform bug qualification and patching, which undergoes a significant battery of tests. An organization looking for patches is bottlenecked by the company providing VM support.*

Copyright © 2017 Prevoty. All Rights Reserved.

4.0 Evaluation Criteria for RASP

Things to look for in a RASP Solution

The following table describes a core set of requirements that should be considered when evaluating a RASP solution. Keep in mind that RASP originated as a solution not simply to test for application security risks, but to mitigate real-time threats to production applications. It has also evolved to provide powerful capabilities for database monitoring and application attack visibility for improved forensics and faster remediation. No two RASP solutions are the same; it's important to carefully consider each solution's capabilities to ensure applications are protected with no impact on operations or performance and plenty of vendor support.

Requirement	Capability Yes (5) / Some (3) / No (0)	Priority High (5) / Med (3) / Low (1)	Score = Capability X Priority
Protected application operates as expected RASP solutions must not interfere with expected application behavior.			
functional tests before and after RASP protections have same results *(i.e. the application is not "broken", no false positives)*		**5**	
functional tests before and after RASP monitoring have same results *(i.e. the application is not "broken")*		**5**	
Protected application performs as expected RASP solutions must not significantly impact response timings or require significantly more production machine resources.			
user experience tests before and after RASP protections have similar results *(i.e. very low response latency)*		**5**	
user experience tests before and after RASP monitoring have similar results *(i.e. negligible response latency)*		**5**	
load tests before and after RASP protections have similar results *(i.e. response timings very similar under load)*		**5**	
load tests before and after RASP monitoring have similar results *(i.e. response timings very similar under load)*		**5**	
functional tests before and after RASP protections show similar processor usage (i.e. CPU usage within acceptable range)		**3**	
functional tests before and after RASP monitoring show similar processor usage *(i.e. CPU usage within acceptable range)*		**3**	
functional tests before and after RASP protections show similar memory usage *(i.e. RAM usage within acceptable range)*		**3**	

Copyright © 2017 Prevoty. All Rights Reserved.

functional tests before and after RASP monitoring show similar memory usage *(i.e. RAM usage within acceptable range)*		**3**	
Exploits effectively prevented RASP solutions must prevent harm to application users, organizations and production deployment environments.			
security tests before and after RASP protections show the mitigation of known OWASP Top 10 vulnerabilities such as XSS, SQLi, Command Injection, XML Injection, etc *(i.e. before and after a tool such as IBM AppScan is employed OR an application security assessment is performed by an org such as WhiteHat Security.)*		**15**	
RASP protections not by-passed through fuzzing or obfuscation techniques		**5**	
security tests before and after RASP protections show a significant reduction in the volume of successful attack traffic penetrating the application		**3**	
RASP solution integrates with common dynamic security testing solutions to improve effectiveness *(e.g. IBM AppScan, WhiteHat Security)*		**1**	
RASP solution provides data that can be used to enhance developer awareness / security training programs		**1**	
Blends into operations RASP solutions must fit in with deployment environment and operational procedures and minimize Total Cost of Ownership.			
RASP solution supports applications (custom, open source, third-party) deployed in any environment *(e.g. on-premises data center, private / public cloud, containers, PaaS/IaaS)*		**5**	
RASP solution does not introduce risk to deployment environment *(e.g. not a point of failure due to things like "regular expression denial-of-service")*		**5**	
RASP solution installation is unambiguous and requires minimal external dependencies to operate		**5**	
RASP solution installation can be automated for "at-scale" deployments *(e.g. CI/CD/DevOps - Jenkins, Chef, Puppet, Ansible)*		**5**	
RASP solution is effective immediately with out-of-the-box defaults *(e.g. minimal tuning, no learning-mode required)*		**5**	
RASP solution has pre-built integration capabilities with commonly used monitoring systems *(e.g. Splunk, QRadar, Nitro, etc.)*		**5**	
RASP solution provides comprehensive data output *(i.e. timestamp, payload, URL, port, source / destination IP, violation / event type, session ID, cookies, stack trace)*		**5**	
Upgrades to RASP solution minimize triggers to change management procedures		**3**	

Copyright © 2017 Prevoty. All Rights Reserved.

Upgrades to RASP solution require minimal restarts to production systems		**3**	
Vendor support RASP clients must access to world-class support, the latest software releases, features, optimizations and bug fixes in a timely manner.			
RASP solution vendor's focus is RASP		**5**	
RASP solution vendor has predictable release cycle		**3**	
RASP solution vendor provides 24x7 Severity 1 support		**3**	
RASP solution vendor can provide critical hotfixes upon demand		**3**	
RASP solution vendor provides web/phone/email support		**3**	
RASP product documentation is concise and unambiguous		**1**	
RASP product documentation describes output data format, content and values		**1**	
RASP solution vendor has authorized service providers		**1**	
Total Score			

Copyright © 2017 Prevoty. All Rights Reserved.

5.0 Conclusion

Not all RASPs are Created Equal

This guide was designed to help security decision-makers think critically about the capabilities of RASP solutions, compare and contrast RASP against other tools (e.g. WAFs, penetration testing, application security testing, etc.), and come up with questions to ask vendors. Not all RASP solutions are created equal; each provides different levels of visibility, performance, scalability, accuracy, and ease of implementation / maintenance.

As the category expands, so does the range of effectiveness. Key RASP differentiators include: *methodology, coverage, speed,* and *ease of deployment*. For instance, in today's DevOps-enabled business climate, new fully-automated RASP technology that plays nice with Continuous Integration / Continuous Deployment cycles and reports real-time, actionable security analytics have a significant edge over traditional RASP tools. Therefore, it's important to judge a RASP's capabilities as it relates to each enterprise's unique challenges and use cases while considering the key differentiators. We recommend that readers use this document as a tool to shape their research when evaluating competitive offerings.

RASP Will Continue to Evolve

RASP technology is a powerful and innovative component in making applications more secure given increasing software deployments and distributed application architectures. With RASP, we are finally able to address attacks in production and explore ways to embed security functions into the applications themselves. Once a nascent category undergoing development and expansion, RASP has gained momentum as a viable, enterprise-grade security solution and a popular choice for achieving DevSecOps alignment. Enterprises using RASP are currently unlocking powerful insights and making smarter application development, security operations and vulnerability remediation decisions.

Copyright © 2017 Prevoty. All Rights Reserved.