WARATEK

Executive Review and Summary

Runtime Application Security and Threat Remediation

Table of Contents

Waratek Executive Summary

Introduction

The threat landscape has evolved dramatically in recent years. Attackers aiming to steal sensitive data can now develop sophisticated, customized, and zero-day attacks. Criminal organizations or nation-states often support these adversaries, utilizing underground ecosystems that provide technology platforms such as malware development kits and distribution infrastructures, including botnets for hire. Defending against these advanced and persistent threats requires a robust combination of people, processes, and technology that many enterprises struggle to manage. Waratek offers a comprehensive solution through our patented Software-defined RASP platform, designed to provide runtime application protection, virtual patching, and unparalleled security management capabilities. With over a decade of experience protecting some of the world's largest and most critical environments, Waratek's solutions are built to defend against the most sophisticated attacks.

Business Overview

Waratek was founded to address the growing need for real-time application security. Our team comprises security experts with deep experience in defending high-profile applications and environments at enterprise scale from modern-day threats. Waratek's Software-defined RASP platform empowers organizations by allowing them to apply security patches without downtime instantly, prevent vulnerabilities at runtime, and automatically protect their applications from known and unknown (zero-day) exploits. Not only do we develop industry-leading solutions, but we also leverage them on behalf of our customers to ensure they stay ahead of evolving threats. Waratek's solutions are deployed in enterprises across the globe, protecting critical applications and data in real time, ensuring compliance, and mitigating security risks.

Our Mission

Waratek helps you stay ahead of attackers by delivering continuous, real-time application protection. In today's rapidly changing threat landscape, enterprises face attacks from every angle, often lacking the comprehensive visibility and automated defenses necessary to detect and respond effectively. Waratek bridges this gap by providing advanced runtime security with the automation and intelligence needed to detect, respond to, and prevent threats before they can cause damage. By thinking like an adversary and offering complete visibility across your application stack, Waratek ensures your operations stay secure.

Technology Overview

Waratek's Software-defined RASP platform provides holistic, application-layer protection that works seamlessly across environments to secure your applications at runtime. With the power to virtually patch vulnerabilities in real-time, prevent exploits, and enhance overall security without impacting performance, Waratek's solutions are built for today's dynamic threat landscape. Our platform leverages patented runtime tainting to track untrusted data flows throughout all layers of the application stack. Combined with a powerful semantic and syntactic analysis engine, Waratek analyzes at runtime the data, the code structure and the app's runtime behavior to deliver real-time protection against known and unknown security threats with unparalleled accuracy. Waratek ensures that your critical business operations remain safe and compliant, allowing you to focus on growth without the risk of security breaches.

Waratek Solution Overview

Technology Focus
The focus of this project is Waratek's Runtime Application Self-Protection (RASP), a cutting-edge enterprise solution designed to secure Java applications at runtime. Waratek takes a proactive, real-time approach to detecting and remediating vulnerabilities, including zero-day exploits. The platform provides continuous protection for your applications without impacting performance, using a combination of patented data tainting and lexical analysis technologies.

Runtime Security for Java Applications — And More
Waratek's Software-defined RASP goes beyond traditional security approaches by integrating seamlessly into the Java runtime environment. By monitoring and analyzing all inputs to your application, Waratek Waratek provides comprehensive protection against all OWASP Top 10 security risks, including SQL injection, command injection, cross-site scripting (XSS) attacks, insecure deserialization, and more. With real-time data flow tracing, Waratek tracks untrusted data as it moves through the application, distinguishing between trusted and untrusted sources with pinpoint accuracy. This enables Waratek to provide runtime protection that is both proactive and retrospective, ensuring that threats are neutralized before they can cause harm.

Metadata as the DNA of Your Application Security
Signatures and log files have limitations when it comes to comprehensive application security. Waratek takes a different and patented approach by leveraging rich metadata from user inputs. Waratek's platform continually follows untrusted user inputs as they move throughout the application, monitoring for mutated syntax. This approach ensures accurate security controls in real-time without the delays and operational overhead of traditional security approaches. Waratek's patented runtime inspection technology enables full execution tracking, real-time attack detection, and automatic remediation, ensuring your applications stay secure without requiring code changes or downtime.

Identify, Protect, and Respond — All in One Solution
You can't defend what you can't detect. Waratek Secure™ provides unparalleled visibility into your application's runtime environment, automatically identifying, classifying, and neutralizing threats in real-time. With no performance hit, Waratek provide comprehensive protection against advanced attacks, including the OWASP Top 10, injection vulnerabilities, remote code execution, privilege escalation even zero-day vulnerabilities. The platform monitors all phases of the threat lifecycle—blocking infiltration, command and control, lateral movement, and data exfiltration—while providing instant insight into an attack's "who, what, where, when, why, and how," enabling swift remediation.

Waratek's Platform: Real-Time Defense for Modern Applications
Deploying Waratek lets your enterprise identify, mitigate, and report threats in real-time. Waratek's RASP offers deep visibility and control over Java applications, blocking threats such as data exfiltration, insider threats, external attacks, and malware. With zero downtime, Waratek's virtual patching capabilities ensure that your applications remain secure without ever having to stop your system for updates.

Waratek's Real-Time Security Capabilities Include:

1. **Threat Detection**: Waratek provides real-time threat detection by monitoring application behavior at the runtime layer. Using techniques like data tainting and syntax analysis, Waratek identifies and blocks code injection, SQL injection, and other malicious input attacks before they can be executed. This detection is based on the structured analysis of runtime data rather than machine-learning algorithms, ensuring precise and effective protection against known and zero-day vulnerabilities.
2. **Threat Prevention**: Through security rules applied at runtime, Waratek blocks malicious behaviors such as code injections, process forking, and unauthorized file access. These rules are designed to prevent common exploit vectors like command injection or insecure deserialization, ensuring protection without requiring downtime or code changes. This prevention includes real-time mitigation of zero-day vulnerabilities through virtual patching.
3. **Runtime Security Controls**: Waratek offers deep visibility into runtime events by correlating application actions and user inputs to prevent malicious behavior. This includes control over filesystem I/O, network I/O, APIs, and system calls within the Java environment, ensuring comprehensive security for Java applications without complex policy management or the operational inefficiencies of heuristic approaches such as machine learning and signature libraries.
4. **Granular Data Flow Tracking**: Waratek monitors untrusted data flow throughout the application, ensuring that any attempts to manipulate this data are detected and neutralized. This protection is not focused on Data Leakage Prevention (DLP) but rather on runtime protection of critical application operations.
5. **Runtime Event Monitoring**: Waratek continuously monitors and logs runtime events, providing security teams with real-time insights into potential attacks or suspicious activity. Although the platform does not integrate real-time threat intelligence feeds, it does provide granular visibility into application behavior and potential vulnerabilities.

How Waratek Secures Your Java Applications

Waratek's RASP technology goes beyond the limitations of traditional security solutions such as Web Application Firewalls (WAFs) and heuristics-based RASP tools. With Waratek, your applications are shielded from attacks without any performance trade-offs, protecting your critical infrastructure with zero downtime. Waratek provides visibility into every layer of the runtime environment, identifying malicious inputs (payloads), blocking them before execution, and securing applications from known and unknown (zero-day) vulnerabilities.

Detailed Technical Capabilities

Runtime Application Protection and Monitoring
Waratek's Runtime Application Self-Protection provides real-time security for Java applications by monitoring, analyzing, and preventing vulnerabilities at runtime. This is achieved through deep application visibility, allowing Waratek to trace untrusted data, apply security rules, and neutralize threats without affecting application performance or requiring downtime. Waratek captures detailed runtime information essential for proactive defense and analysis, ensuring comprehensive protection.

Application Visibility
Waratek offers deep visibility into the application's runtime environment through continuous monitoring. Every interaction, including file operations, network communications, and database queries, is logged and analyzed. By tracking the movement of data through the application, Waratek identifies risky operations and prevents malicious inputs from executing. This detailed runtime metadata allows security teams to detect vulnerabilities that traditional tools like WAFs or firewalls may miss, such as complex injection attacks or process forking.

Threat Detection
Waratek's RASP platform provides comprehensive threat detection for all layers of the Java runtime environment. This includes the detection of known vulnerabilities, such as SQL injection and command injection, as well as emerging threats. Waratek automatically applies security rules that identify and block malicious behaviors, even without prior knowledge of the vulnerability. Detection is based on static and dynamic analysis, protecting against complex attack vectors, including zero-day exploits.

Non-Heuristic Detection
Waratek takes a fundamentally different approach from traditional security methods, which often rely on heuristics like pattern matching, signatures, or blacklists. Instead of using heuristic-based detection, Waratek leverages data tainting and syntax analysis to identify code injection and other attacks in real time. This method eliminates the drawbacks associated with heuristic techniques, such as high false-positive rates and the need for continuous tuning.

Waratek's platform uses **data tainting** to track untrusted input flows through the application. This provides deep insight into where data comes from, its use, and whether it has been manipulated for malicious purposes. Coupled with **syntax analysis**, Waratek examines the structure of inputs against the formal grammar of expected data (such as SQL queries) to identify code injection attempts and other exploits based on syntactic anomalies rather than input patterns.

Waratek offers accurate, real-time protection against zero-day vulnerabilities without the performance degradation or false-positive risks inherent to heuristic-based systems by analyzing the application context and ensuring inputs conform to valid structures. This non-heuristic approach allows for precise threat detection and ensures that applications are protected without the need for code changes or ongoing rule adjustments.

Lexical Analysis Detections
Waratek employs advanced semantic syntactic and lexical analysis techniques to analyze application behaviors in real time. Unlike statistical anomaly detection, which relies on pattern frequency or sequence, semantic syntactic and lexical analysis focuses on the structure and syntax of data inputs and their usage within the application. By breaking down and analyzing the code at a granular level, Waratek can detect malicious inputs as they interact with the application logic.

For instance, lexical analysis allows Waratek to examine the specific syntax of inputs to identify when untrusted data has been manipulated to exploit vulnerabilities, such as concatenated SQL queries or HTML and Javascript code that result to XSS attacks. This method goes beyond pattern matching by analyzing the intent and structure of the input, providing real-time, precise identification of security threats. Waratek's lexical analysis engine ensures comprehensive detection of both known and unknown threats, offering a more sophisticated layer of runtime protection compared to statistical analysis.

Virtual Patching (RAMPART)
One of Waratek's standout features is its ability to patch vulnerabilities virtually in real-time without requiring application code changes or system restarts. When a new vulnerability is discovered, Waratek can deploy a patch immediately using predefined security rules, providing immediate protection while minimizing disruption to business operations. This capability is especially valuable in defending against zero-day vulnerabilities, ensuring applications remain secure before official patches are available. Waratek can address any CVE, anywhere in the application stack even in cases where there is no public zero-day exploit available.

Insider Threat Detection
In addition to external threats, Waratek also monitors for insider threats. By analyzing user behavior and access patterns within the application, Waratek can detect anomalies that indicate privilege escalation or data exfiltration attempts. This proactive approach helps organizations prevent both accidental and malicious insider threats.

Reporting and User Administration
Waratek provides built-in reporting features, including executive-level summaries, detailed threat detection reports, and performance analytics. Custom reports can also be generated to meet specific business needs. Waratek supports role-based access control (RBAC), ensuring that sensitive data is accessible only to authorized personnel. All user actions within the platform are audited, with detailed logs available for review or export to a Security Information and Event Management (SIEM) system.

Alerting Mechanisms
Waratek uses a sophisticated policy engine to generate real-time alerts based on runtime events. Alerts can be triggered by specific behaviors, such as attempts to exploit known vulnerabilities or access restricted files. In addition, Waratek supports integration with external monitoring systems, enabling security teams to stay informed of potential threats as they arise. Waratek also plans to integrate machine-learning-based anomaly detection into its alerting mechanisms for even greater accuracy in threat detection.

Session Termination
Waratek can automatically terminate suspicious sessions or block specific actions to prevent further damage when a threat is detected. This includes terminating malicious processes, preventing file operations, or blocking unauthorized network communications in real-time, ensuring that threats are neutralized before they can exploit vulnerabilities.

Security Subsystems (Stores and Trampolines):
The Waratek platform uses an innovative system of hooks and trampolines to enforce security rules. Hooks are injected into the application bytecode to redirect control flow to Waratek's trampolines, which implement specific security checks. The security subsystems (called "Stores") store and manage these rules, which are applied dynamically to block unsafe operations like unauthorized file access or unsafe method invocations.

Runtime Micro-compartmentalization and privilege de-escalation
Waratek creates fine-grained, micro-compartments around application operations by defining specific boundaries around these operations. Waratek can lower the privileges of specific micro-compartments to avoid privilege escalation and API abuse attacks.

Portal:
The Portal Dedicated (on-premise) or Portal (cloud-based) is the interface for security engineers to configure and manage security policies. These policies are written in Waratek's RAMPART declarative language and are used to define the security rules enforced at runtime. The console also handles security event reporting, configuration management, and communication with the deployed agents. Both the on-premise and cloud solutions provide similar functionality, though they differ in deployment environments.

Privileged Actions:
Waratek supports privileged operations for actions that user-defined security rules might otherwise restrict. This feature allows the agent to perform necessary tasks, such as sending security events or interacting with system resources, while bypassing the applied security constraints for the agent itself. Privileged contexts are tightly controlled and only used when necessary, ensuring that normal application processes remain subject to the defined security policies.

Deployment and Scalability:
Waratek is designed for easy deployment in both small and large-scale environments. The agent is lightweight and integrates seamlessly into existing Java environments, requiring no changes to the application's source code. The scalable platform allows the Management Console to manage hundreds or thousands of Java agents across various environments. Communication between agents and the console is secured using TLS encryption.

Differentiators and Cooperative Collaboration

For the purposes of this comparison, we'll focus mainly on the approach adopted several vendors with regards protection for live application workloads. The primary approach here is commonly termed Heuristics. Heuristics is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution, or when classic methods fail to find any exact solution in a search space. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. Under each metrics we'll generalize the approach or call out the specific technology.

1. Deployment and Integration

- **Waratek**: Integrates directly into the JVM layer, offering runtime protection without modifying the application source code or requiring external devices. This means applications are inherently protected once they run within Waratek's secure container, adhering to **NoDev** and **NoOps** principles. Waratek modifies runtime behavior without requiring application redeployment or restarts, making it a seamless and unobtrusive solution.
- **Heuristic based RASP**: Typically, a perimeter defense mechanism, but an approach also used within the runtime by several other RASP vendors. This is essentially a WAF into the runtime, with little to no additional benefits. It acts as a barrier between the client and application by inspecting and filtering incoming HTTP requests. While they can block attacks before they reach the application, they are detached from the application itself, providing no deep integration with the runtime environment.

2. Granular, Runtime-Level Protection

- **Waratek**: Provides **deep runtime protection** by operating inside the JVM itself. It monitors Java applications' behavior at the bytecode level, allowing for fine-grained control over application processes such as **file I/O**, **network activity**, and **SQL injections**. Waratek can apply security rules dynamically without restarting the JVM, enabling **virtual patching** in real time without application downtime.
- **Heuristic based RASP**: Similarly to all perimeter defense solutions such as WAFs, heuristic-based RASP solutions rely on **pattern matching** and **signature-based filtering** at the HTTP request level, meaning they can only block known attack patterns with questionable false positive rates. They do not have insight into the application's runtime environment or how data flows through the application once the request has been processed. They also often rely on predefined attack signatures, which makes them less effective against novel or zero-day attacks.

3. Zero-Day Attack Mitigation

- **Waratek**: Excels at handling **zero-day vulnerabilities** by embedding security into the application runtime itself. By blocking application attacks such as **SQL injections**, **command injections**, deserialization and **cross-site scripting (XSS)** at runtime, Waratek can neutralize unknown vulnerabilities and exploits even before they are officially patched by the software vendor.
- **Heuristic based RASP**: Can struggle with zero-day vulnerabilities since it depends on predefined rules and attack signatures. Unless rules are updated to detect new types of threats, it may fail to block sophisticated or previously unknown attacks.

4. Visibility and Customization

- **Waratek**: Offers unparalleled **visibility** into the application's internal workings. Administrators can define **simple, customizable security rules** at the runtime level using the Waratek platform's domain-specific language (DSL). This flexibility allows organizations to tailor the security model to their specific needs.
- **WAF**: Provides a more **generic layer of protection**, focusing on filtering malicious HTTP requests. While WAF rules can be adjusted, they don't offer the same level of integration or customization based on the internal behavior of the application code itself.

5. Overhead and Performance

- **Waratek**: Due to its deep integration with the JVM, Waratek has **minimal performance impact**, especially when compared to the overhead typically associated with network-based security tools. Since Waratek operates at the runtime level, it can enforce security rules more efficiently and dynamically.
- **Heuristic based RASP**: Can introduce **latency** as it inspects all incoming requests and can become a bottleneck, particularly under high traffic conditions. Its need to inspect every HTTP request can result in noticeable delays in response times, especially when using more complex filtering rules. It's widely understood that with this type of approach, Security and Performance are mutually exclusive.

6. Security Maintenance and Operations

- **Waratek**: Enables **virtual patching**, which allows organizations to apply security fixes in real-time without restarting the JVM or taking the application offline. This approach significantly reduces the burden on operations teams and eliminates the need for traditional patching cycles that often lead to downtime.
- **Heuristic based RASP**: Requires frequent updates to its rules and attack signatures, which can require human intervention. It does not provide the ability to apply security patches to the application itself, meaning that any new vulnerabilities discovered will still need to be patched through traditional means.

Conclusion:

- **Waratek** offers **in-depth, runtime-level protection** for Java applications that a heuristic based technology cannot provide. It integrates security directly into the runtime, offering fine-grained control, zero-day protection, and virtual patching with **no downtime**.
- **Heuristic based RASP**, on the other hand, blocks known attack patterns at the network level but because these only consider partial http requests they lack any real understanding of how an application intended to use this request and as such they are very prone to false positives. We can say that the heuristic approach offers protection by analyzing the request or data sent to the application whereby Waratek will analyze the code and how it interacts with each and every request, it's this context which offers significantly more accuracy and a significantly lower total cost of ownership.

Technical Appendix

Application Runtime Protection - Vendors Comparative Matrix

Metrics	Waratek	WAF	Contrast	Imperva	Signal
Low TCO	✔	✘	✘	✘	✘
Production Ready	✔	✘	✘	✘	✘
Performant and Accurate	✔	✘	✘	✘	✘
Low FP's	✔	✘	✘	✘	✘
Full Stack protection	✔	✔	✔	✔	✔
Application Code Changes Needed	✘	✘	✔	✔	✔
Heuristics Based	✘	✔	✔	✔	✔

Waratek Security Controls & Features Matrix

The Common Weakness Enumeration (CWE) list from MITRE contains several entries that overlap with each other. The matrix below only covers the most important CWEs and is not a comprehensive list. Waratek's features can be used to remediate more vulnerabilities not listed in the matrix below.

Vulnerability	Waratek Security Control Description	Waratek Security Control Name
SQL Injection (CWE-89)	Data Tracing is used to identify unsafe data entering into the JVM. Tainted Syntactic Analysis for the SQL syntax is used to detect SQL injection attacks.	Data Tracing & Tainted Syntactic Analysis
Command Injection (CWE-78)	Process control by restricting OS commands using black and white lists. The file:exec rule intercepts all process forking operations and restricts processes that are allowed or not allowed to be forked.	Process Forking Interception
Process Control (CWE-114)	Process control by restricting OS commands using black and white lists. The file:exec rule intercepts all process forking operations and restricts processes that are allowed or are denied to be forked.	Process Forking Interception
Cross-site Scripting (CWE-79), (CWE-80)	Data Tracing is used to identify unsafe data entering into the JVM. Tainted Syntactic Analysis for the HTML syntax is used to detect XSS attacks.	Data Tracing & Tainted Syntactic Analysis
HTTP Header Injection / Response Splitting (CWE-113)	Data Tracing is used to identify unsafe data entering into the JVM. Tainted Syntactic Analysis for the HTTP headers is used to detect HTTP Header Injection attacks and Response	Data Tracing & Tainted Syntactic Analysis

	Splitting.	
HTTP Verb/Method Validation (CAPEC-274)	Waratek hooks into the Servlet API and intercepts all HTTP requests. Using the RAMPART HTTP Verb validation rule users can restrict HTTP resources (URIs) to specific HTTP verbs.	HTTP Servlet Operations Interception
Missing Encryption of Sensitive Data (CWE-311)	Waratek's virtualization platform transparently upgrades ServerSocket instances (which are not encrypted) to SSLServerSocket instances (which are encrypted). The upgrade is transparent to the guest and the updated SSLServerSocket uses the latest TLS algorithms and cipher suites that are available in the host JVM.	Runtime Virtualization
Cleartext Transmission of Sensitive Information (CWE-319)	Waratek's virtualization platform transparently upgrades ServerSocket instances (which are not encrypted) to SSLServerSocket instances (which are encrypted). Because of the fact that the upgrade occurs on the host JVM, the upgrade is transparent to the guest and the updated SSLServerSocket uses the latest TLS algorithms and cipher suites.	Runtime Virtualization
Unrestricted Upload of File with Dangerous Type (CWE-434)	Using the file:write rule, Waratek performs file-system write operation control by restricting file extensions and paths using black and white lists.	File System Operations Interception
Direct Use of Unsafe JNI (CWE-111)	Waratek can intercept direct use of JNI libraries. Using the file:native security rule JNI libraries can be restricted based on white or black listed JNI files. Also, RAMPART can perform input validation to the JNI method calls to	JNI (native) Library Loading Interception RAMPART for input validation

	safeguard their correct API usage.	
Cross-Site Request Forgery (CSRF) (CWE-352)	Waratek's solution is based on the best practices for CSRF. The CSRF:STP rule is based on the Synchronizer Token Pattern. Waratek generates a cryptographically strong random token (challenge), binds it to the user's HTTP session and injects it on every HTTP response using a very efficient streaming tainted HTML lexer. Finally, Waratek validates the token on every received HTTP request. The CSRF Same-Origins HTTP RAMPART rule is based on validating the HTTP request's origin via standard standard HTTP request headers.	Synchronizer Token Pattern Verifying Origin With Standard Headers
Path Traversal (CWE-22)	Waratek identifies and tracks all external input using the tainting/tracing engine. If an IO operation is attempted by the application using paths that are externally controlled (i.e. tainted) then the Waratek filesystem lexer lexes the path for tainted path sequences that traverse the file system. In other words, Waretek identifies all IO operations where the user has control over the absolute or the relative location of a filesystem path.	Data Tracing & Tainted Syntactic Analysis
Relative Path Traversal (CWE-23)	Waratek identifies and tracks all external input using the tainting/tracing engine. If an IO operation is attempted by the application using paths that are externally controlled (i.e. tainted) then the Waratek filesystem lexer lexes the path for tainted sequences such as "..", "../", etc. In other words, Waretek	Data Tracing & Tainted Syntactic Analysis

	identifies all IO operations where the user has control over the relative location of a filesystem path.	
Absolute Path Traversal (CWE-36)	Waratek identifies and tracks all external input using the tainting/tracing engine. If an IO operation is attempted by the application using paths that are externally controlled (i.e. tainted) then the Waratek filesystem lexer lexes the path for tainted absolute path sequences such as "/abs/path", "C:\Windows\", etc. In other words, Waratek identifies all IO operations where the user has control over the absolute location of a filesystem path.	Data Tracing & Tainted Syntactic Analysis
Open Redirect (CWE-601)	Data Tracing is used to identify unsafe data entering into the JVM. Tainted Syntactic Analysis for the HTTP location syntax is used to detect Open Redirect attacks.	Data Tracing & Tainted Syntactic Analysis
Improper Input Validation (CWE-20)	For vulnerabilities such as SQLi, XSS, Path Traversal, Open Redirect, etc, Data Tracing and Tainted Syntactic Analysis is used. For custom vulnerabilities, RAMPART rules can be used.	Data Tracing & Tainted Syntactic Analysis RAMPART rules
Dynamic Code Evaluation: Code Injection (CWE-94)	Data Tracing and Tainted Syntactic Analysis for application injection vulnerabilities mitigation. NSLR for bytecode injection mitigation.	Data Tracing & Tainted Syntactic Analysis NSLR
Use of a Broken or Risky Cryptographic Algorithm (CWE-327)	The network:TLSUpgrade rule transparently upgrades legacy and broken SSL cipher suites and algorithms.	Method Invocation Interception Field Access Interception
OGNL Expression Injection (CWE-917)	OGNL uses Java reflection and introspection to address the Object Graph of the runtime application. Using	RAMPART rules

	simple RAMPART rules users can intercept reflective operations and control the ones that should be allowed or not.	
Unsafe Reflection (CWE-470)	Using simple RAMPART rules users can intercept reflective operations and control the ones that should be allowed or not.	RAMPART rules
Session Fixation (CWE-384)	Waratek identifies successful user logins, intercepts the login process and regenerates the user's HTTP Session ID.	HTTP Session ID Regeneration
External Control of File Name or Path (CWE-73)	Path Traversal and File rules restrict the file name and path that can be externally controlled.	Data Tracing & Tainted Syntactic Analysis File System Operations Interception
Legacy Java Protection (CWE-937)	Runtime Virtualization allows the usage of the latest JVM version as host JVM and a legacy JVM version as the guest JRE. Additionally, the Waratek Security Platform protects the guest JRE against vulnerabilities and zero-days via the enabled security rules.	Runtime Virtualization
Clickjacking (CAPEC-103)	Waratek can intercept HTTP responses and add any desired HTTP header. The header:servlet rule allows a user to add the X-FRAME-OPTIONS HTTP header to the HTTP response that instructs the web browser not to allow framing.	HTTP Response Header Addition
Deserialization of Untrusted Data (CWE-502)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the	Runtime Virtualization Runtime Micro-compartmentalization Runtime Privilege De-escalation

	deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service.	
Incomplete Blacklist (CWE-184)	The file, classlink, reflect and network rules allow to define new white/black lists that override any other blacklist defined in the application or external firewall. Additionally, the Runtime Micro-compartmentalization performed for the deserialization mitigation does not depend on any type of white/black lists; therefore a blacklist can never be incomplete for the case of deserialization since there is no blacklist.	Waratek file, network, classlink, reflect rules. Runtime Micro-compartmentalization Runtime Privilege De-escalation
Insufficient Compartmentalization (CWE-653)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Neither the JVM or any application framework compartmentalizes the system during Java deserialization.	Runtime Micro-compartmentalization Runtime Privilege De-escalation
Improper Fulfilment of API Contract ('API Abuse') (CWE-227)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service.	Runtime Micro-compartmentalization Runtime Privilege De-escalation
Execution with Unnecessary Privileges (CWE-250)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the	Runtime Micro-compartmentalization Runtime Privilege De-escalation

	system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. The system is protected regardless of the application user/role privileges and regardless of the presence of a Java Security Manager.	
J2EE Bad Practices: Use of System.exit() (CWE-382)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. System.exit() is a privileged operation that is not accessible during deserialization.	Runtime Micro-compartmentalization Runtime Privilege De-escalation
J2EE Bad Practices: Direct Use of Threads (CWE-383)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. Direct use of Threads are	Runtime Micro-compartmentalization Runtime Privilege De-escalation

	privileged operations that are not accessible during deserialization.	
Trust Boundary Violation (CWE-501)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service.	Runtime Micro-compartmentalization Runtime Privilege De-escalation
Use of Dynamic Class Loading (CWE-545)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. Dynamic class definition using a deserialized stream is a privileged operation that is not accessible during deserialization.	Runtime Micro-compartmentalization Runtime Privilege De-escalation
EJB Bad Practices: Use of Class Loader (CWE-578)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This	Runtime Micro-compartmentalization Runtime Privilege De-escalation

	safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. Dynamic class definition using a deserialized stream is a privileged operation that is not accessible during deserialization.	
Uncontrolled Recursion (CWE-674)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. During Java deserialization of Java collections, uncontrolled recursion is not allowed. Also, using RAMPART, any method or function can be patched to avoid uncontrolled recursion.	Runtime Micro-compartmentalization Runtime Privilege De-escalation RAMPART
Use of Potentially Dangerous Function (CWE-676)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged and non-dangerous operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. Also, using RAMPART dangerous functions can be replaced by non-dangerous code.	Runtime Micro-compartmentalization Runtime Privilege De-escalation RAMPART
Allocation of Resources Without Limits or Throttling (CWE-770)	During an object deserialization (Java or	Runtime Micro-compartmentalization

	XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. During Java object deserialization, specific limits are put in place to avoid unbounded allocation of resources (DoS). Also, using RAMPART specific resource limits or throttling can be added to methods and functions.	Runtime Privilege De-escalation RAMPART
Uncontrolled Memory Allocation (CWE-789)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. During Java object deserialization, specific limits are put in place to avoid unbounded allocation of memory (DoS). Also, using RAMPART specific limits can be added avoid unbounded memory allocation from methods.	Runtime Micro-compartmentalization Runtime Privilege De-escalation RAMPART
Improper Control of Interaction Frequency (CWE-799)	During an object deserialization (Java or XML) operation, Waratek compartmentalizes the system at runtime and	Runtime Micro-compartmentalization Runtime Privilege De-escalation RAMPART

	creates boundaries around operations. Data Tracing is used to trace the deserialized stream and allows only non-privileged operation to be performed via the deserialized stream. This safeguard's the JVM's state, protects against API abuse case and Denial-of-Service. During Java deserialization of Java collections, the total number of invocations of specific methods is limited to avoid unbounded invocation frequency (DoS). Also, using RAMPART the interaction frequency can be controlled for any method or function.	
Use of Obsolete Functions (CWE-477)	Using RAMPART, code that uses obsolete functions can be patched to use supported and non-obsolete functions.	RAMPART
Function Call with Incorrectly Specified Arguments (CWE-628)	Using RAMPART, methods that are invoked with arguments that are not correctly specified can be patched to invoke the arguments correctly.	RAMPART
Unchecked Return Value (CWE-252)	Using RAMPART, code that does not check a method's return value can be patched to check its validity.	RAMPART
Improper Initialization (CWE-665)	Using RAMPART, code that does not initialize resources correctly can be patched to initialize the resources correctly.	RAMPART
Comparison of Object References Instead of Object Contents (CWE-595)	Using RAMPART, methods that incorrectly compare object references instead of the contents of the objects themselves can be patched and replaced with methods that correctly compare object values instead of object references.	RAMPART
Use of Wrong Operator in	Using RAMPART, methods	RAMPART

String Comparison (CWE-597)	that use the wrong operator when comparing a string, such as using "==", can be replaced with methods that use the equals() method which is the proper way to perform String comparison.	
Improper Resource Shutdown or Release (CWE-404)	Using RAMPART, methods that incorrectly release a resource can be patched to properly release a resource and make available for re-use.	RAMPART