White Paper:

Zero Downtime for Zero Day Vulnerabilities

July 2015
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EXECUTIVE SUMMARY</td>
<td>3</td>
</tr>
<tr>
<td>2. BUSINESS BENEFITS</td>
<td>4</td>
</tr>
<tr>
<td>3. INTRODUCTION</td>
<td>4</td>
</tr>
<tr>
<td>4. PROBLEM DEFINITION</td>
<td>5</td>
</tr>
<tr>
<td>5. SOLUTION DEFINITION</td>
<td>7</td>
</tr>
<tr>
<td>6. TAINT DETECTION</td>
<td>8</td>
</tr>
<tr>
<td>7. RULES FILE</td>
<td>9</td>
</tr>
<tr>
<td>RULE FORMAT</td>
<td>10</td>
</tr>
<tr>
<td>Category</td>
<td>10</td>
</tr>
<tr>
<td>Sub-Category</td>
<td>10</td>
</tr>
<tr>
<td>Parameter</td>
<td>10</td>
</tr>
<tr>
<td>Action</td>
<td>11</td>
</tr>
<tr>
<td>Loglevel</td>
<td>11</td>
</tr>
<tr>
<td>CASE STUDY APACHE STRUTS VULNERABILITY CVE-2013-2251</td>
<td>12</td>
</tr>
<tr>
<td>8. SUMMARY</td>
<td>13</td>
</tr>
<tr>
<td>9. APPENDIX</td>
<td>14</td>
</tr>
<tr>
<td>ABOUT WARATEK</td>
<td>19</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>20</td>
</tr>
</tbody>
</table>

©Waratek Ltd. This document is distributed for information purposes only and does not form part of, or constitute an agreement with, Waratek. Current and prospective Waratek products, services and processes are liable to change from time to time without notice and accordingly no liability of any nature (in contract, tort or otherwise, including for misrepresentation) can be accepted by Waratek in relation to this document. Unless otherwise agreed with Waratek, the contents of this document must be kept confidential and may not be copied or distributed.

No employee or other representative of Waratek has authority to incur liability on behalf of Waratek in relation to this document.

The following are trademarks of Waratek Ltd: Waratek, Waratek AppSecurity for Java, Waratek Locker, Waratek Security Rules Editor, Java Virtual Container and BYOS.
1. Executive Summary

In this paper we will describe how Waratek AppSecurity for Java reduces organisational risk by protecting applications from unidentified security vulnerabilities (Zero Day Vulnerabilities).

It will also outline how Waratek dramatically reduces the operational costs incurred by existing patching processes.

We will describe how Waratek's implementation of Runtime Application Self Protection (RASP) gives organisations unparalleled visibility into their applications, providing them real-time monitoring of the runtime environment.

Furthermore, given this full visibility of the runtime environment, we will demonstrate how the unique Taint Detection Engine blocks the most critical attacks vectors, such as SQL Injection, Command Line Injection and Cross Site Scripting.

We will also illustrate how simple rules defined in the AppSecurity rules engine can disable the fundamental attack vectors that are most commonly exploited by Zero Day Vulnerabilities, rendering them harmless.

Finally we show how the dynamic nature of Waratek AppSecurity alters the mechanics of security patching, giving enterprises the ability to maintain Zero Downtime on their mission critical systems by enabling a “Virtual Patching,” capability. Given the widespread difficulties and costs of current security patching practices, it is our intent to demonstrate the operational benefits of Waratek's innovative solution to the reader.
2. Business Benefits

Cybersecurity is a principle concern for the enterprise and this is reflected in the increasing budgets allocated to cybersecurity departments. The costs of a cybersecurity breach include reputational, operational and ultimately financial loss, in some cases the losses being so severe that they effectively result in the termination of business activities. The principle benefit of Waratek AppSecurity for the business is protection from some of the most potent and insidious weapons in Cyber crime, namely the exploitation of zero day vulnerabilities; zero day attacks.

Waratek AppSecurity provides a simple and powerful implementation of Runtime Application Self Protection (RASP) affording an unprecedented level of security to the enterprise. Through the creation of a small number of general rules, many of the attack vectors that zero day attacks rely upon are effectively disabled. What is more the whole process of maintaining security is optimised in line with NoDev and NoOps principles by the introduction of Virtual Patching. The disruption to services caused by existing security patching processes is eliminated. Enterprises can respond to security threats far more quickly than has been possible in the past and the response comes at a greatly reduced cost, both financial and temporal.

3. Introduction

A zero-day attack or threat is an attack that exploits a previously unknown vulnerability in a computer application or operating system, one that developers have not had time to address and patch. It is called a "zero-day" because the programmer has had zero days to fix the flaw. (Source Wikipedia)

By their nature zero day vulnerabilities pose a great threat to enterprise systems. In recent years both legitimate and black markets have been established that facilitate the exchange of information pertaining to zero day exploits allowing the criminal community to quickly exploit such vulnerabilities before a software vendor becomes aware of them and acts to publish a mitigating patch. Even when a patch is published many clients of the software are slow to implement the patch leaving them even more exposed as knowledge of the vulnerability is made public, giving a larger number of criminals access to the information they need to comprise unprotected systems. Thus the need to test and deploy a patch for a zero day vulnerability as soon as the vendor releases the patch is an imperative for any enterprise.
4. Problem Definition

For many enterprises that utilise Java, promoting a security patch from development and test environments into production (Figure 1) requires either managed down time of the production system as depicted in Figure 2 or some operational choreography in order to maintain service during the deployment, depicted in Figure 3 and Figure 4. This is because Java security patching invariably requires a restart of the Java Virtual Machine (JVM).

“Java Security Patching invariably requires restarting the JVM”

When a patch is released for a zero day vulnerability, clients of the software that require the patch rush to test and deploy the patch into production. This process can be extremely disruptive both to development projects, which must reallocate resources to review the patch, integrate and test it and to operations teams that have to schedule the deployment of the patch to production systems.

For small organisations the overall level of disruption may be of little consequence, but for large institutions with hundreds or thousands of applications the disruption caused by such unplanned events is costly and failing to act in time renders the enterprise extremely vulnerable to criminal exploitation.

Existing mechanisms in Java for applying security patches essentially violate NoDev\(^2\) and NoOps\(^3\) principles. They divert development effort away from delivering core business logic towards integrating and testing the patch and they demand operational resources to deploy the patch into production.

“Security patching can result in downtime or complex operational choreography”
Beyond the financial cost of carrying out unplanned security patch deployments to mitigate zero day vulnerabilities there is also a temporal cost. Project lead times can be adversely impacted and, as we will see, the security patching process can result in managed downtime of the service or complex operational choreography to maintain the service while the application is patched.

A typical n+1 Highly Available architecture employs a minimum of two of every infrastructure component within a data centre to increase the availability of applications deployed to the infrastructure. For example in Figure 1 we depict two active load balancers within a region routing network traffic to a pair of JVMs deployed to a pair production servers residing in separate data centres.

Step 1 shows all of the active components serving customers requests during normal processing. Where Service Level Agreements (SLA) permit managed downtime of the service; Java security patching will result in the JVMs being shutdown before the patches are applied as show in step 2. The final step in this choreography is to resume service returning to the initial state depicted in Figure 1.

Our contrived depiction hides the potential complexity of this security patching operation which, for many businesses, still requires manual intervention and possibly co-ordination across multiple teams (data base, networks, middleware, apps, etc) depending on the nature of the application.

Figure 3 and Figure 4 depict an alternative step 2 and step 3 where the SLA demands make it necessary to maintain a service while security patching is taking place. The choreography requires network traffic to be diverted from a JVM, while it is patched and restarted, to other JVMs that remain active and continue to service client requests. There are many different permutations of this choreography, for example network traffic may be diverted from individual JVMs, groups of JVMs deployed on specific servers or between different data centres while security patching takes place.

Even within a single business one often finds a high degree of variance in the way services are maintained for different applications as they are being patched.

While complex patching choreographies can be standardised and automated they can still be time consuming to execute and difficult to recover from should a failure occur. Ultimately neither approach for applying security patches (managed downtime or operational choreography) is satisfactory. What is need is a mechanism that obviates the need to deploy a code patch to mitigate security vulnerabilities, thereby eliminating the need to restart the JVM and removing the dependencies created on development and operations teams. Such an approach; a NoDev, NoOps compliant approach, is exactly what Waratek AppSecurity for Java provides.
5. Solution Definition

Waratek has developed the industry’s first secure JVM, building upon Oracle’s HotSpot JVM. As a 100% compatible TCK compliant JVM, any standard Java application can run without modification and thus immediately benefit from Runtime Application Self Protection. Think of it as a Secure Container that applications run in, which can be moved around as required.

“AppSecurity for Java protects applications from sophisticated attacks without having to alter application code”

AppSecurity for Java protects applications from sophisticated attacks without having to alter application code. It embeds security into the Java run-time layer, not the application itself. Pre-defined security rules immediately identify and optionally block compromised application behaviour with minimal performance impact.

“Waratek can load and enable security rules without requiring a restart”

The security rules engine is configured via an easy to use text file and can load and enable security rules without requiring a restart of the either the JVM or any applications deployed to it as depicted in Figure 4. The rules file can contain application rules governing a range of application behaviour and specific security vulnerability classes for example:

- File I/O
- Network I/O
- Static Linking
- Reflection
- Errors and Exceptions
- Java Native Interface
- SQL Injection
- Cross Site Scripting
The rules are designed to be as simple as possible, no more than a single line long and allowing a high degree of generality minimising the size of the rules file in marked contrast to Web Application Firewalls.

Figure 5 demonstrates how Java applications deployed to Waratek AppSecurity are dynamically protected at runtime. When a new security patch is released by either third party application providers or by an internal application team a new security rule based on the patch can be defined. The rule will typically be deployed and tested on non production infrastructure and upon successful testing it will be deployed into production.

AppSecurity can load and enable the new rule set without requiring a restart of the secure container or any applications deployed to it. Patching is effectively carried out in a single step.

The process is simple and can be carried out by the team that controls the application or middleware infrastructure without any operational dependency on other teams.

6. Taint Detection

For simple security vulnerabilities and exploit scenarios, the traditional notion that “tainted data is unsafe” may be sufficient. However as application business logic complexity has grown – driven in large part by the growth of web applications – this naïve notion has revealed itself as too crude for many real-world applications. As a result, a perception has grown in some communities that data tainting has no value.

Java, the most popular programming language in financial services and many other industries, and most commonly attacked, has never possessed native support for data tainting functionality. While multiple attempts have been made to provide data tainting on the Java Platform, most have introduced unacceptable performance overheads, effectively ruling out their practicality for most applications.

However, the recent emergence of Runtime Application Self-Protection (RASP) technologies has renewed interest and development in using data tainting on Java.

The reason is simple. RASP technologies are built on the principal of analyzing the runtime behavior of an application as it executes and correlating application events in real-time to identify attacks and vulnerability exploits. This ‘real-time’ correlation gathers as much information as possible about application behavior (what Gartner calls ‘application context’), including:

- what values have been input to the application, and from where,
- where those input values are flowing through the application, and
- what those input values are being used for by the application
When combined with RASP, data tainting fills a significant security void, namely the ability to identify ‘data origin’. Only data tainting provides the intelligence to distinguish, at runtime, between data which originated from outside and data which originated from inside the application. For RASP, this intelligence unlocks a higher level of runtime event correlation, which provides new, more accurate and reliable application security capabilities.

RASP is changing the negative perception of data tainting as a tool for application security. As new low-overhead data tainting systems become integrated with RASP the two technologies are taking application security to the next level.

### 7. Rules File

To demonstrate the power of AppSecurity both to proactively mitigate zero day vulnerabilities and to quickly remediate applications once a patch is released we will examine an example rules file.

```
VERSION 1.0
# Comments are allowed
# This rule will disable the execution of any binary initialted the within the JVM
file:exec:*:deny:warn

# Network I/O Rules

# Static Linking Rules
classlink:class:java.lang.System:shutdown:debug

# Reflection Rules
reflect:class:com.blah.MyClass:shutdown:error

# Error & Exception Rules
throwable:throw:java.sql.SQLIntegrityConstraintViolationException:allow:info

# Deny all SQL injection attacks
sql:database:oracle:deny:trace
```

In this simple rule file we have effectively disabled two key attack vendors that are often exploited by zero day attacks namely, file execution and SQL Injection (the first and last rules respectively). In this instance even if a zero day vulnerability is exploited that depends on these attack vectors, it will not have any material effect on our application and the security breach will be logged giving security personnel the opportunity investigate the issue.
Rule Format

Each rule is contained within a single line and consists of 5 components. No white space must be present within the rule. The components are separated using the ‘:’ character and must appear in the following order:

<category>:<subcategory>:<parameter>:<action>:<loglevel>

We will briefly review each component here however a detailed specification is beyond the scope of this document and reader is advised to refer to the latest Waratek Specification.

Category

Category refers to the group of classes to which a security rule can be applied. Table 1 below describes the valid categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Rule Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File I/O</td>
<td>file</td>
<td>Intercept attempts to access a file or directory, or execute processes.</td>
</tr>
<tr>
<td>Network I/O</td>
<td>network</td>
<td>Intercept attempts to connect to address or port.</td>
</tr>
<tr>
<td>Static Linking</td>
<td>classlink</td>
<td>Intercept invocation of certain methods, or writes to fields.</td>
</tr>
<tr>
<td>Reflection</td>
<td>reflect</td>
<td>Intercept reflected access to methods, fields, or classes.</td>
</tr>
<tr>
<td>Errors &amp; Exceptions</td>
<td>throwable</td>
<td>Intercept and act on specific Throwable objects being created.</td>
</tr>
<tr>
<td>Native Library</td>
<td>native</td>
<td>Disable native library loading, or specify library white list and/or grey list.</td>
</tr>
<tr>
<td>SQL</td>
<td>sql</td>
<td>Intercept SQL injection attacks.</td>
</tr>
</tbody>
</table>

Table 1 Description of available Categories in a rule file

Sub-Category

Sub-Categories generally correspond to an operation that may be performed by an application, for example ‘read’ or ‘write’ a file.

Parameter

Parameters can be thought of as arguments and can identify resources within sub-categories. For example, in the case of the ‘file’ category, if the sub-category indicates that ‘write’ must be intercepted, then the parameter can be used to specify the path to a particular file for which writes should be intercepted.
Action

Action describes the action taken by AppSecurity when the rule is triggered. Actions that are common to all categories are described in Table 2 below:

<table>
<thead>
<tr>
<th>Rule Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destroy</td>
<td>Shutdown Java Virtual Container (JVC) immediately.</td>
</tr>
<tr>
<td>shutdown</td>
<td>Shutdown JVC gracefully.</td>
</tr>
<tr>
<td>suspend</td>
<td>Suspend the threads of the JVC.</td>
</tr>
<tr>
<td>deny</td>
<td>Return false or throw exception.</td>
</tr>
<tr>
<td>allow</td>
<td>Return true. If no action is specified in the rule, allow is assumed.</td>
</tr>
</tbody>
</table>

Table 2 Common actions

Loglevel

The Log Level component indicates the verbosity of logging. Omitting the Log Level will result in no log messages being generated when the rule is triggered. Table 3 below describes the various Log Level options:

<table>
<thead>
<tr>
<th>Log Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>The highest possible rank and is intended to turn off logging. This level is assumed if the &lt;loglevel&gt; is omitted from the rule.</td>
</tr>
<tr>
<td>error</td>
<td>Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.</td>
</tr>
<tr>
<td>warn</td>
<td>Use of deprecated APIs, poor use of API, 'almost' errors and other runtime situations which are undesirable or unexpected, but not necessarily &quot;wrong&quot;. Expect these to be immediately visible on a status console.</td>
</tr>
<tr>
<td>Info</td>
<td>Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.</td>
</tr>
<tr>
<td>debug</td>
<td>Detailed information on the flow through the system. Expect these to be written to logs only.</td>
</tr>
<tr>
<td>trace</td>
<td>Most detailed information. Expect these to be written to logs only.</td>
</tr>
</tbody>
</table>

Table 3 Log Level descriptions
Case Study Apache Struts vulnerability CVE-2013-2251

To illustrate the how Waratek AppSecurity can pre-emptively render harmless extremely serious vulnerabilities, we will show how our example rule file would have mitigated some of the worst aspects of Apache Struts vulnerability CVE-2013-2251.

Apache Struts is a commonly used, free, open-source, Model View Controller framework for creating Java web applications. In mid 2013 a zero day vulnerability was identified in Object Graph Navigation Language (OGNL); an expression language for getting and setting properties of Java objects. This vulnerability allowed a remote user to execute java code from a browser on a server running a vulnerable version of Apache Struts. The example below shows the browser URI that could be used to execute Java code (other manual exploits are available from Open Security Research):

```
https://xxxx/struts2-blank/example/HelloWorld.action?redirect:${%23a%3d%28new
%20java.lang.ProcessBuilder%28new%20java.lang.String[]{%27uname%27,%27-a%27}%29%29.start
%28%29,%23b%3d%23a.getInputStream%28%29,%23c%3dnew%20java.io.InputStreamReader%28%23b%29,%23d
%3dnew%20java.io.BufferedReader%28%23c%29,%23e%3dnew%20char[50000],%23d.read%28%23e%29,%23matt
%3d%23context.get%28%27com.opensymphony.xwork2.dispatcher.HttpServletResponse
%27%29,%23matt.getWriter%28%29.println%28%23e%29,%23matt.getWriter%28%29.flush
%28%29,%23matt.getWriter%28%29.close%28%29}
```

The code example above demonstrates the ease by which the vulnerability can be exploited to execute arbitrary Java code from a browser. In this instance static method access and execution are enabled and the response output is hijacked to return a message to the browser. The message itself is the output from the execution of the `uname -a` binary. The attacker can easily substitute other more harmful execution commands that can potentially download malware payloads.

In order to completely mitigate this vulnerability’s exposure to binary execution on a server the following rule can be used:

```
# MAC rule for disabling binary execution from the JVM
file:exec:*:deny:warn
```

An attempt to execute the exploit now result in a 404 HTTP error as the request is interpreted as a request for the URL: `/struts2-blank/example/.jsp`. Waratek AppSecurity logs the following message that confirms an attempt to execute a binary was thwarted:

```
2015-04-14 19:09:15,645 WARN DENY myhost File:Exec [/bin/*] - [/bin/uname]
```
8. Summary

In this paper we have introduced Waratek AppSecurity and identified the key capabilities it offers. We have shown the positive impact that these capabilities have for the way business manage cyber security by enabling a virtual patching paradigm. We have demonstrated how AppSecurity implements RASP in a simple and effective manner and how AppSecurity aligns with NoDev and NoOps design principles which optimise the way business allocate resources to IT in order to generate business value.

Java is now ubiquitous in business and the demand for greater protection from Cyber crime continues to grow unabated. Waratek AppSecurity is recognised by the industry to be arguably the most innovative development in cyber security in over a decade, with Waratek being chosen by the RSA Innovation Sandbox 2015 judges as the Most Innovative Company with an inventive solution to a “massive problem”. It has the potential to revolutionise the way Java applications are protected and makes Java one of the most secure platforms for commercial applications.
9. Appendix

Figure 1 Depiction of n+1 HA infrastructure Java security patching process resulting in downtime.
Figure 2 Depiction of service managed service disruption to allow security patching.
Figure 3 Depiction of operational choreography to maintain service during security patching
Figure 4 Network traffic is switched between active and non active JVMs during security patching.
Figure 5 Zero Downtime deployment of rules file

Waratek AppSecurity can load new application firewall rules without requiring a restart or redeployment of the application.
About Waratek

Waratek protects applications from logic, network and vulnerability exploits without code changes, agents or hardware devices. A pioneer in Runtime Application Self Protection, Waratek solutions are available for datacenter and Cloud environments. Waratek Ltd is headquartered in Dublin, Ireland. Waratek Inc. is based in Atlanta, a Delaware corporation and a licensee of Waratek Ltd in the Americas.

Waratek received the RSA Conference’s Innovation Sandbox award as the Most Innovative Company of 2015 saying it was an inventive solution to a “massive problem.” Computer Technology Review awarded Waratek the title of ‘Most Valuable Security Product.’ Waratek has also been named Gartner Cool Vendor, selected for the Fintech Innovation Lab and voted top Global Innovator in the Innotribe Challenge for virtualization solutions.
References

1 The Zero Day Initiative is a program for rewarding security researchers for responsibly disclosing vulnerabilities. Other programmes are listed at bugsheet.

2 NoDev refers to a set of IT management principles that inform managers as to which tasks an enterprise should allocate development resources towards as opposed to utilising third party applications. For example for financial institutions NoDev advises minimising development effort towards tasks such as: security patching, job scheduling, application integration and writing test code; focusing instead upon user experience and developing business logic.

3 NoOps refers to a set of IT management principles that advocates minimising dependency on IT Operations through automation, self service and the simplification of IT processes and architectures.

4 For more information on CVE-2013-2251 please refer to cve.mitre.org.

5 OGNL stands for Object-Graph Navigation Language; it is an expression language for getting and setting properties of Java objects, plus other extras such as list projection and selection and lambda expressions. You use the same expression for both getting and setting the value of a property. More information can be obtained from the Apache Commons OGNL website.