Waratek CTO explains why TLS compliance is difficult and what can be done to ease the pain

TLS 1.0 and 1.1 were deprecated in March 2020, but companies with large estates of web applications are still struggling to upgrade to the new standards. Waratek Founder & CTO John Matthew Holt discusses with CEO John Adams why compliance continues to elude organizations and how Waratek can reduce the time and cost to comply.

Click the button to listen to the full conversion or read the edited transcript below.

LISTEN HERE

Q. Why do companies need to upgrade their TLC protections?

John Matthew Holt: TLS 1.0, which is now deprecated and removed, was born out of SSL v3 in 1999, so we're talking about 20 plus years ago.

Over the years a number of high-profile vulnerabilities and exploits and attacks were identified for v1.0. The root of these high-profile vulnerabilities stemmed from elements of weak cryptography incorporated into this first version of the TLS specification. So, in response to this, TLS 1.1 was developed. While TLS 1.1 didn't have any major known vulnerabilities like Beast or Poodle, it did share the same weak cryptographic elements with its younger sibling.

In October 2018, all of the main browser vendors – Google, Apple, Mozilla, and Microsoft - all jointly agreed to deprecate TLS 1.1 and 1.0 by March 2020. That had the consequence of meaning that any service that didn't support TLS 1.2 essentially went dark.

There is a long tail of core business applications that are still using and are stuck on these outdated cryptographic libraries inside their software stack that do not support TLS 1.2. At the time of its (TLS 1.2) release, some of the major web browsers and platforms didn't actually add support for TLS 1.2 until some significant period of time later.

Q. We hear it’s so difficult for organizations to upgrade; but why is it so difficult?

John Matthew Holt: Yeah. Great question. Any kind of core business critical applications that were installed running Java v5 or any applications running Java v6 earlier than 6u121 don't have any support for TLS 1.2 and that means they can't be accessed or used by the Web browsers from Google and Apple and Microsoft and so on. If you have a server application that is critical to your business and it is using off the shelf Java v7, TLS 1.2 is going to be disabled by default causing problems for you.

Fast forwarding to Java v8 - which is not the latest language for applications, but it is the lion's share of all business applications today - TLS 1.3 (the latest standard) is not supported. And so when people want to start to use the latest cryptographic standards, they have to move to something like Java v11 or higher.

The difficulty comes from the fact that organizations and enterprises are built on these applications installed at a certain point in time. The stack and the libraries used by apps are now problematic because they don't support the new specifications and protocols, but upgrading an app is a non-trivial activity that often requires development time / development effort with lots of testing and lots of cost and time. Many organizations don't have the resources to do that.

Q. This problem isn't going away. What, in your view, are the options that are available to address it?

John Matthew Holt: Anyone that's had that first-hand experience maintaining Java applications will know that upgrading to the latest JDKs are a prime difficulty. For server-side Java applications particularly, there's really only one stopgap to get around this and that's to employ the use of reverse proxies to receive or to communicate TLS 1.2 connections with client endpoints and then proxy that connection internally to the actual application stack using the old TLS 1.0 or 1.1 connections.

That certainly can work technically, but it's far from desirable and far from scalable.

Q. We've helped protect numerous companies that have this problem. They'd be really great if you could spell out a very viable alternative for companies facing this challenge.

John Matthew Holt: The root of the problem is that the software is outdated. More specifically the platform is out of date. So, if a solution can address the fact that the software stack is out of date, then it will have a very convenient side effect that TLS protocol compliance would also be upgraded and addressed. That's exactly what Waratek does.

I can upgrade that workload instantly with just a single restart up to Java 14 and above without any source code changes, no incompatibilities, and the entire process literally takes seconds. I want to upgrade say from Java 6 to Java 14, as an example. I want to go from Oracle Java SE to Open JDK or Amazon Credo or some other some other open source JDK. Immediately the APIs are upgraded, the performance is upgraded, and critically the TLS is upgraded to whatever is the newest version that that JDK you've upgraded will support.

Q. Have clients experienced this and is it scalable across an entire estate?

John Matthew Holt: The deployment process with the *Upgrade* agent is simple and seamless. It's modeled on the standard way of deploying Java agents so you can use automation tools like Chef and Puppet to deploy this at scale. Many of the customers use these kind of automation tools to apply the *Upgrade* agent, not just to upgrade one or two workloads in the corner, but to upgrade hundreds or sometimes thousands of workloads to completely retrofit their application - leapfrogging all of the missed patches and Java updates from a security perspective – and they get immediate regulatory compliance with the software stack by upgrading the TLS from the older standards to 1.3.