Keep it Simple, Keep it Safe – Cybersecurity Research on Impacts of Increasing Complexity of Modern Enterprise Solutions

Vulnerable Beneath the Source Code: Increasing Vulnerability by Decreasing Perceived (and Increasing Actual) Complexity - *Shawn Ware*

Patch Delays: Researching the Threat of Unpatched Systems Reducing Security on Layered Security Solutions - *David Phillips*

UNIVERSITY OF NEBRASKA OMAHA
COLLEGE OF INFORMATION SCIENCE & TECHNOLOGY

Sensitive Data Availability in High-Level Language Applications

PURPOSE

As information systems continue to expand in availability and capability, enterprises invest larger sums of money into development and maintenance of these systems. Increasingly, these systems are being developed with high-level (if not interpreted) languages, taking advantage of these languages' increased readability and maintainability. In May of 2017, it was estimated that over fifty percent of enterprise applications were developed using high-level programming languages {1}.

Java (winner of the "Programming Language of 2015" award {2}) is used by an estimated 9 million developers {3} and is touted as a, "write once, run anywhere" solution. It's adaptability and maintainability have solidified its use in enterprise applications throughout the world {4}. One of Java's touted benefits is its, "garbage collection," a series of algorithmic memory-management functions employed by the Java Runtime Environment (JRE) to remove memory allocation concerns from the developer. Ideally, memory is reserved on an as-needed basis, and memory allocations that are no longer used by the application are freed to the operating system.

However, Java's automated and abstracted memory management poses large cybersecurity concerns. As developers continue to learn and practice in high-level languages, concepts regarding memory management are no longer considered. Though Java (as of Java Standard Edition version 8 [*Java SE 8*]) includes methods for handling *user secrets* (such as passwords) {5}, sensitive information is frequently handled using potentially insecure methods.

This research is to determine what, if any, vulnerability exists in an average enterprise application developed in Java for sensitive-but-unprotected data (e.g. home addresses, social security numbers, banking details, etc.). Specifically, this research is into 1) the duration of vulnerability (how long does sensitive data occupy memory after its use), 2) the most common causes for sensitive data exposure, and 3) the *minimum viable solution* for reducing the likelihood of sensitive data persisting within memory beyond its usefulness.

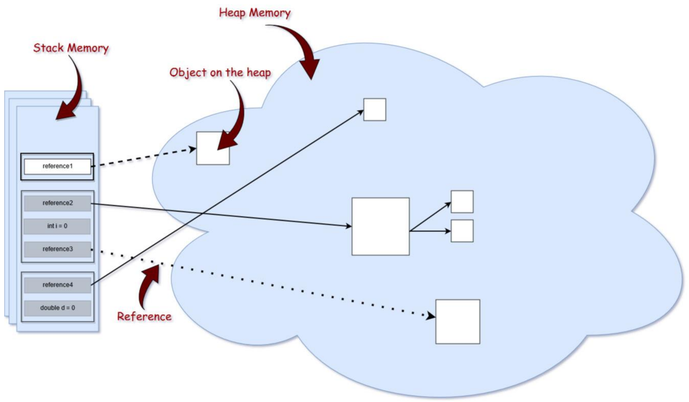


Figure 1 (Above):
A simplified map of Java's memory allocations. The arrows represent the references between the object's memory location stored in the stack (left) with the object and any child objects within the heap (right).

Image credit: (Marian, Constantin. 2018)

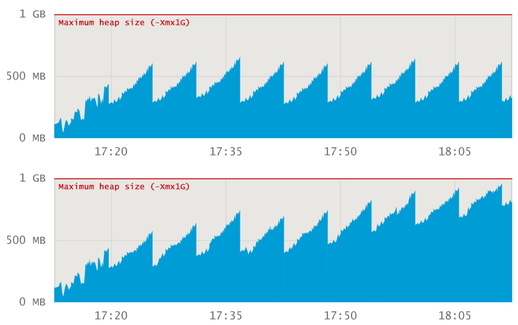
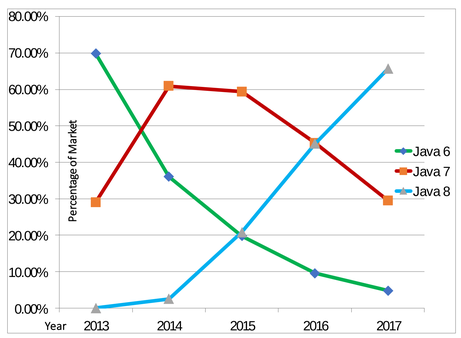


Figure 2 (Left):
A comparison between JVM memory usage (shown in blue) when running a well-programmed application (top) versus an application that uses resources inefficiently (bottom), and thus does not properly dispose of memory objects once they're no longer useful.

The bottom image resembles the focus of this research; objects persist in memory despite being no longer useful, and are thus prone to being acquired. However, not all applications with memory management issues would graph equivalently; there are multiple types of memory management and object reference mistakes, and each causes a different version of the bottom graph.

Figure 3 (Right):
A history of Java version utilization by percentage, from 2013 through 2017. Since Java 8's release in 2013, it did not become the majority version until 2016 (and a major majority until 2017).

Left Image Credit: (Šor, Vladimir. 2017)
Right Data Credit: (Salnikov-Tarnovski, Nikita. 2017)



METHODS & ASSUMPTIONS

It is assumed that the enterprise application's source code is available for analysis. This assumption allows the team to provide detailed feedback on the insecure data-handling practices most likely to result in sensitive data persistence.

It is assumed that the application does not implement stringent authentication within the application. This assumption prevents unnecessary difficulty during research and analysis. As most authentication platforms are third-party and/or separate applications entirely, any proof-of-concept applications used during research will not implement strict authentication.

It is assumed that the performance and/or latency penalty incurred by the debugging processes is minimal. This assumption allows for data collected regarding garbage collection performance to be analyzed without caveat.

Research into the vulnerability of sensitive data began using a Virtual Private Server (*VPS*) leased by one of the researchers through OVH Hosting, running Debian 9 64-bit {6}. Installed on the server is Oracle's Java Development Kit (*JDK*) version 8, update 161 (*JDK 1.8.0_161*){7}. Apache Tomcat (version 8.5.14) {8} is installed and configured to manage Java Web Applications (*WAR*s). The VPS supports SSH, and the Java installation has been configured to enable verbose debugging, as well as remote (console) debugging.

A proof-of-concept Java WAR is being developed to allow extensive testing of Java's memory allocation, management, and object availability post-use. The application features text entry, database object retrieval (through Java's standard database APIs {9}), and data processing and storage. The application will follow best practices when handling data entry, validation, sanitization, and storage. The application will follow best-practices when available, unless best practices is superseded by code readability or maintainability as appropriate (to believably simulate developer preference for code legibility over performance).

The application will be hosted on the VPS and interacted with through a standard web browser by researchers. Various debugging and benchmarking applications (including JVisualVM, Jrocket, and others) will be attached (remotely connected to) the Java Virtual Machine (*JVM*) responsible for running the application, and relevant metrics (including memory allocation by the JVM, object lifecycles, and garbage collection performance statistics) will be collected periodically as needed during, before, and after interaction with the application by a user.

Additional time will be invested for research into alternative solutions for handling of sensitive (but not *secret*) information in higher-level languages, including object-clearing (explicit overwriting of variable values), interception (invocation of explicit procedures or functions when triggered by an external event), and other methods to reduce sensitive data availability.

RESEARCH TIMELINE

A large amount of research remains to be completed. As the Cybersecurity Capstone course proceeds, work will continue at a steady pace. Currently, the VPS and requisite frameworks are in place and preliminary research into enterprise Java applications is underway. Development of the sample application has begun, and team members are contributing towards documentation and development of methodologies for research. This initial work will be completed by March 5, 2018.

Exploratory research is expected to begin March 5 and proceed through March 31, 2018. During this time, the sample application will be heavily utilized, as will various open-source and commercial debugging utilities. Work will initially be focused on automation of site interaction through frameworks such as Cypress.io, and experimentation with options and utilities provided by the debugging applications. Some time is reserved for debugging and optimization of the application. After automation of interaction has reached a stable point, more time will be devoted towards collection of data for analysis. The application will be tested in various situations and under various configurations (all documented and repeated). Dedicated team members will be responsible for collection of debugging information and compilation of research data. Other team members will continue with their earlier research, and develop a functionally identical application using discovered techniques for sensitive data. If time permits, the second application will be deployed on the remote server and similarly tested, with data collected separately for comparison.

Analysis and reporting of any findings or recommendations is expected to begin April 1 and proceed through April 22, 2018. During this time, the data collected through earlier efforts will be compiled and analyzed. Further time may be devoted to further, narrow research on any findings for which the team determines may be particularly relevant. Any relevant information will be transferred into written and graphic presentation materials to prepare for Capstone presentations, which are presently set for April 23 and 25, 2018.

REFERENCES & RESOURCES

{1}: TIOBE. (2018, February). "TIOBE Index for February 2018". The software quality company. Retrieved from https://www.tiobe.com/tiobe-index/

{2}: Iyer, Kavita (2017, March 11). "Top 20 most popular programming languages in 2017". *TechWorm*. Retrieved from https://www.techworm.net/2017/03/top-20-popular-programming-languages-2017.html

{3}: "charm" (2010, May 11). "Number of Java Developers". *Infomory.com*. Retrieved from http://infomory.com/numbers/number-of-java-developers/

{4}: Wilson, Breanne (2016, June 20). "Why is Java the most popular programming language?". *Oracle Blogs*. Retrieved from https://blogs.oracle.com/oracleuniversity/why-is-java-the-most-popular-programming-language

{5}: Oracle (n.d.). "PBEKeySpec (Java Platform SE 8)". *Oracle*. Retrieved from https://docs.oracle.com/javase/8/docs/api/javax/crypto/spec/PBEKeySpec.html

{6}: See https://www.ovh.com/world/vps

{7}: See https://www.oracle.com/technetwork/java/javaee/downloads/index.html

{8}: See https://tomcat.apache.org/tomcat-8.0-doc/

{9}: See https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

The Impact of Known Vulnerabilities on a Layered Solution

PURPOSE

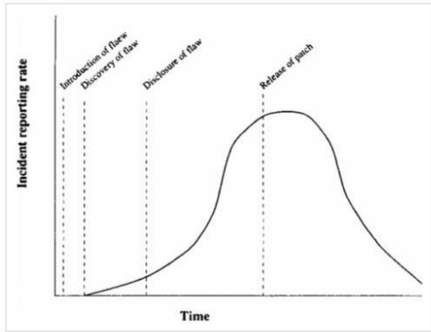
The purpose of layered solutions is to give the ability of an institution to use layered commercial products to deliver access to their critical data while maintaining security. Traditionally, government devices were designed and certified to be used to access their most sensitive data. This is extremely costly and time consuming. Recently, the government has been reviewing proposals which would utilize commercial devices to deliver the same results as the government devices. To increase the government's assurance, it is attempting to ascertain if the use of multiple or layered solutions will provide the level of assurance that a government device would deliver.

This task will attempt to determine the security benefit of using layered solutions in an institution and if it affords any advantages. The two layers being investigated will be enterprise VPN solutions. It will be prudent to research the National Vulnerability Database and the Common Vulnerabilities and Exposures database in order to create a timeline of vulnerabilities for enterprise solutions over the years at varying code levels. It will be important to distinguish between the time the vulnerability is known and the time when the patch is released. There is also another aspect to take into account. It cannot be assumed that when the patch is released, that this will be the time that the institutions devices would be patched. There will need to be a patch window, in which, an institution would review, test, and assign the patch to a change management request. The research will need to be limited to only vulnerabilities that could possibility compromise a security layer. The timeline will be developed to aid in displaying overlapping vulnerabilities of the layered solutions selected, if they exist.

It will be also important to ascertain the possibility of an adversaries' ability to compromise layered solutions that do not have overlapping timelines. This would involve an adversary's ability to compromise one layer and then wait until the other solution has a new vulnerability known to it. History has shown, in many cases adversaries will gain some access and sit and wait until another opportunity presents itself.

Lastly, it will be important to research how long it takes an exploit to be created once the vulnerability is known. This information will be easily identified as the timeline is created with vulnerabilities, patches and available exploits of VPN solutions. It will be important to also point out if there have been successful exploits of these vulnerabilities, not just hypothesized exploits that should be possible.

This is type of research is extremely important to all institutions to help protect sensitive data and mitigate the adversaries' ability to breach their network. By compiling all of this data and creating a timeline to analyze this data, the hope is to provide useful information to confirms or denies the advantage of introducing layered solutions into an institutions network security posture.



RESEARCH TIMELINE & METHODOLOGY

The above graph is what an exploitation curve might look like. It's intuitive, but not accurate for all situations. Once the flaw is discovered, some exploitations happen and the vulnerability may be passed around the dark corners of the internet. Once the vulnerability is disclosed, you see a large increase in number of exploitations, which makes sense as the flaw is now completely public. Once the patch is released, you don't see an immediate fall off since not many organizations can deploy a patch immediately. Once patching begins however, we see an the number of exploitations fall, but not quite as quickly as they rose. Towards the end you see that the number of exploitations never bottom out at zero because some systems never get patched.

To the right, we see four different cases measuring the number of attacks per day. Case 1 looks at vulnerabilities that were disclosed, but never patched. Case 2 shows vulnerabilities that were disclosed and then had a patch released shortly after. Case 3 looks at vulnerabilities that were never officially disclosed before being patched. Finally, case 4 shows a situation where the vulnerability was disclosed and a patch was released on the same day. Do note this data is limited by the fact it was a relatively small sample of honeypot systems.

We plan on performing our own mathematical analysis to determine an average time between disclosure, patch availability, and patch deployment. We may allow users of the web app to change these variables to reflect different situations if we are confident in our initial results.

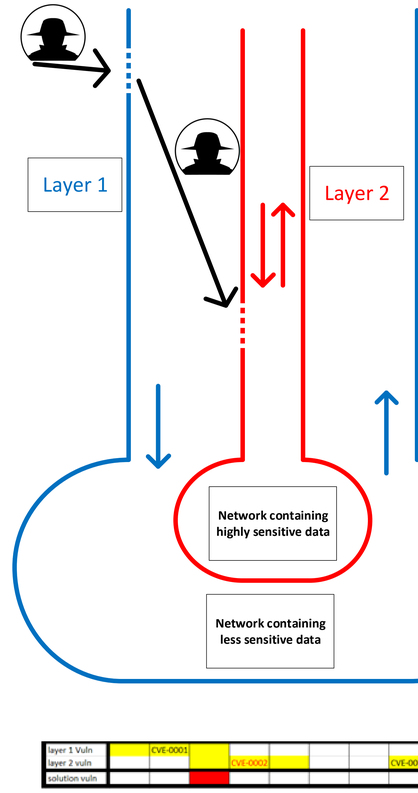


Figure 4: Simplified Diagram of Layered VPN Vulnerabilities

User enters the blue VPN (layer 1) to access the general network and perform tasks. To perform tasks that require highly sensitive data or systems, the user must connect to the red VPN (layer 2) in order to access the next level.

Our research is focused on determining what benefits a setup like this would bring. Our goal is to identify timelines where both layers have known vulnerabilities that can be exploited thus eliminating any benefits of the multi-layered solution.

Additionally, we are also looking for situations where an attacker could exploit the outer layer and establish a permanent "residence" in that layer and simply wait for the second layer to become vulnerable. For the purposes of our research, we are assuming there are no active countermeasures during an intrusion. We are also assuming the setup is using two different VPN solutions from different providers.

The dotted areas indicate potential vulnerabilities an attacker could leverage to exploit the VPN.

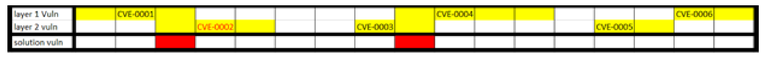


Figure 5 (Above): An analysis with a timeline demonstrating vulnerability windows of layered solutions

The timeline above is a rough example of what we hope to create by the end of our research. We plan to examine 10 enterprise VPN solutions and their vulnerabilities in order to create a web app which would allow a user to select the vendors they wish to use for a layered VPN solution. The web app would then output a timeline that shows how long both solutions were vulnerable individually and together as a layered solution.

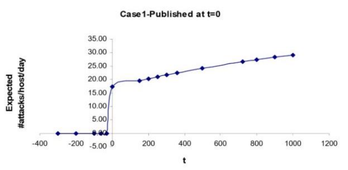


Figure 1 Simulated impact of publishing without patch

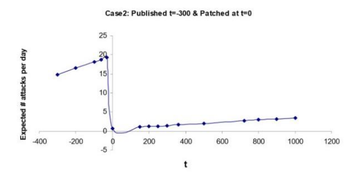


Figure 2 Simulated impact of patching a known vulnerability

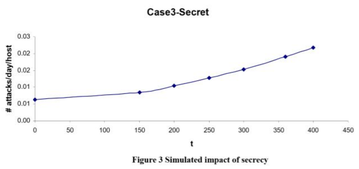


Figure 3 Simulated impact of secrecy

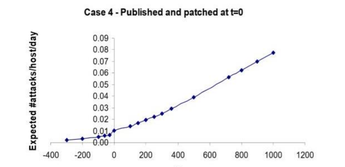


Figure 4 – Simulated effect of patching and publication

Graph Sources:
A. Arora, R. Krishnan, A. Handkumar, R. Telang, Y. Yang, "Impact of Vulnerability Disclosure and Patch Availability – An Empirical Analysis" H. John Heinz III School of Public Policy and Management, Carnegie Mellon University, April 2004.
Browne, H.K, Arbaugh, W.A, McHugh, John, Fithen, WL, "A trend analysis of exploitations". Security and Privacy 2001.