Sensitivel Data Availability in High-Level Language Applications

**PURPOSE**

An information systems continue to expand in availability and capability, enterprises invest millions of dollars into applications to develop and maintain commercial-grade systems. Increasingly, these applications are being developed with high-level (i.e., not interpreted) languages. Taking advantage of these languages' increased modularity and maintainability, is microservices (S7) - an emerging pattern of enterprise applications were developed using high-level programming languages (S).

Java (version of the “Programming Language of 2015” award (S)) is used by an estimated 9 million developers (S). According to a report by Intuit (S), developers are using Java for running the application, and relevant metrics (including memory allocation by the JVM, overwriting of variable values), interception (invocation of explicit procedures or functions when a method is called), and any child objects within the heap (right).

A simplified map of Java's memory allocations. The image represents the relationships between the object's memory locations (left) and the object's operations on those memories (right).

**METHODS & ASSUMPTIONS**

It is assumed that the enterprise's application space is available for analysis. This assumption allows the team to derive detailed findings in the resource-handling behavior most likely to influence data persistence.

It is assumed that the application does not store sensitive information within the application. This assumption prevents unnecessary difficulty in research and analysis, as no authentication platforms are third-party and/ or separately deployed applications, an Kopf of correct solutions. For example, research that is not possible in the case of high-level languages.

It is assumed that the performance and/or latency penalty caused by the debugging processes is minimal. This assumption allows for the data collected regarding collection persistence to be analyzed without cause for the delay in response and/ or latency penalty caused by the debugging processes.

A protocol specific Java VM is being developed to allow extraction testing of Java’s memory locations, management, and object availability persistence. The application features native entry, database access maintained through Java’s standard database APIs (9), and data processing and storage. The application will be tested by extracting data when handling events, activation, cancellation, and storage. The details will follow best practices whenever high-level languages are used, since code change is code maintainability or automatically (to be independently validate developer preferences for code fidelity over performance).

The application will be hosted on the VMs and interconnected with a testbed with switches by researchers. Various debugging and benchmarking applications (including nvprof, curious, and/or other tools) will be used to analyze the performance of the applications for high-level languages. High-level language and garbage collection statistics will be collected periodically as needed during testing, and other interaction with the applications for both.

Additional time will be invested in research into alternative solutions for handling of sensitive (but not secret) information in higher-level languages, including object-caching (exploit of high-level languages, and/or virtual functions and/ or variables that are not needed during an event), and other methods to reduce sensitive data availability.

**REFERENCES & RESOURCES**

dm/index.html.

**PURPOSE**

The purpose of layered solutions is to provide an ability of an institution to run enterprise applications in a manner that maximizes modularity and maintainability, reduces the likelihood of sensitive data persisting within memory beyond its usefulness.

**METHODS & ASSUMPTIONS**

It is assumed that the application can be analyzed without cause for the delay in response and/ or latency penalty caused by the debugging processes.

It is assumed that the performance and/or latency penalty caused by the debugging processes is minimal. This assumption allows for the data collected regarding collection persistence to be analyzed without cause for the delay in response and/ or latency penalty caused by the debugging processes.

Analysis and reporting of any findings or recommendations are expected to begin April 1 and proceed through April 23, 2018. During this time, the data collected through earlier efforts will be compiled and analyzed. Further time may be devoted to further research or new methodologies for research. This task will be completed by March 30, 2018.

**REFERENCES & RESOURCES**