Security Vulnerability Notice

SE-2014-02-GOOGLE-4

[Google App Engine Java security sandbox bypasses, Issues 37-39]

DISCLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER SECURITY EXPLORATIONS, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY SECURITY EXPLORATIONS OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SECURITY EXPLORATIONS, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF SECURITY EXPLORATIONS OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

Security Explorations discovered three additional security vulnerabilities in Google App Engine for Java. A table below, presents their technical summary:

ISSUE #	TECHNICAL DETAILS	
37	Origin	`com.google.apphosting.runtime.security.shared.intercept.java.lang.invoke.MethodHandles.Lookup` class
	Cause	incorrect implementation of `findStatic` method of the `MethodHandles.Lookup` mirror class
	Impact	access to unintercepted method handles of static, security sensitive methods
	Type	partial GAE security bypass vulnerability
38	Origin	`com.google.apphosting.runtime.security.shared.intercept.java.lang.invoke.MethodHandles.Lookup` class
	Cause	missing `RuntimeVerifier.verifyAccessible` security check in the `findStatic/findVirtual/findSpecial` methods
	Impact	reflective access to members of classes loaded by non-user Class Loaders
	Type	partial GAE security bypass vulnerability
39	Origin	`com.google.apphosting.runtime.security.shared.intercept.java.lang.invoke.MethodHandles.Lookup` class
	Cause	missing `RuntimeVerifier.verifyAccessible` security check in the `bind` method
	Impact	reflective access to methods of classes loaded by non-user Class Loaders
	Type	partial GAE security bypass vulnerability

Issue 37 makes it possible to invoke static methods of certain, security sensitive classes such as `java.net.URLClassLoader` class. The problem stems from the fact that GAE API Interception mechanism assumes that static method lookups can be only done with respect to the classes that declare them. In Java, static methods are "inherited" by subclasses and are resolved in a similar way as instance methods. As a result, static methods can be successfully resolved from subclasses of the classes that declare them. In our Proof of Concept codes we exploit this condition to obtain access to the unintercepted `newInstance` method handle of `java.net.URLClassLoader` class. This leads to an arbitrary Class Loader instantiation and Class Sweeper / JRE Class Whitelisting escape.

Issues 38 and 39 manifest an inconsistency in the way security checks are implemented by GAE Reflection API interception layer. All of the mirrored core Reflection API classes include the invocation of a `RuntimeVerifier.verifyAccessible` security check verifying whether a given Class member (a Method, Field or Constructor) can be accessed from a user class loader namespace. Such a security check is however missing from Method Handles API. As a result, sole access to a restricted Class object[1] can be exploited to create arbitrary instances of restricted GAE classes and to call their methods.

Issue 38 stems from a missing security check in the shared `find` method used internally by `findVirtual`, `findSpecial` and `findStatic` methods of the `MethodHandles.Lookup` mirror class. Issue 39 has similar origin, but the check is missing from the `bind` method of the `MethodHandles.Lookup` mirror class (it does not rely on the `find` method).

[1] such as a Class originating from a non-user Class Loader namespace or that is not on the JRE Class Whitelist.

Issues 37 and 38 could be combined together to achieve a complete GAE Java security sandbox escape. The following exploitation scenario is implemented in our Proof of Concept code (POC29) to illustrate that:

1) Issue 37 is used to create an instance of a `java.net.URLClassLoader` class (UCL loader),
2) UCL loader is used to obtain a reference to the restricted `com.google.apphosting.runtime.security.URLClassLoaderFriend` GAE class, this is possible since the parent of the UCL loader is a system Class Loader in which namespace this class resides,
3) Issue 18 [1] is used to obtain a reference to a restricted `sun.misc.Resource` JRE class, it gets extracted from the parameter list of a private `defineClass` method of `java.net.URLClassLoader` class,
4) Issue 38 is used to obtain a method handle to the constructor of `URLClassLoaderFriend` class,
5) constructor method handle obtained in step 4 is used to instantiate the `URLClassLoaderFriend` class, current Thread's context class loader is provided (`UserClassLoader` instance) as constructor's argument,
6) Issue 38 is used to obtain method handle references to `getResource` and `defineClass` methods of `URLClassLoaderFriend` class,
7) the resource object corresponding to a `PrivLoader` class is obtained by invoking the `getResource` method handle with a path argument denoting a location of the `PrivLoader` class,
8) the resource object obtained in step 7 is provided as an argument to the `defineClass` method handle invocation, as a result, an intermediate Class Loader class (`PrivLoader`) is defined in `UserClassLoader` namespace and outside of a GAE Class Sweeper sandbox,
9) an instance of a `PrivLoader` class is created and used to define a privileged `HelperClass` class,
10) `HelperClass` class is instantiated and a Security Manager is turned off.

It's worth to note that the above scenario makes use of Issue 18, which was evaluated by Google as *working as intended* (WAI) issue. We have warned Google[2] that this weakness may turn out to be helpful in a future attack. In our exploit scenario, Issue 18 proves to be helpful again as it allows to obtain a reference to a restricted JRE class (from a `sun.*` package). In a standard JRE environment, this would not be possible as user code would need to possess proper privileges to achieve that[3].

Attached to this report, there is a Proof of Concept codes that illustrates the impact of the vulnerabilities described above. It has been successfully tested in a production GAE environment patched against security issues we reported to Google in Dec 2014 / Jan 2015.

[2] we did it on 27-Dec-2014 at the time of providing Google with arguments regarding WAI Issues 17-20. These arguments have been also presented in our technical report (2.4.1.2 Closing thoughts).
[3] `RuntimePermission("accessClassInPackage.sun")`

REFERENCES

[1] "Google App Engine Java security sandbox bypasses", technical report http://www.security-explorations.com/materials/se-2014-02-report.pdf

---

About Security Explorations

Security Explorations (http://www.security-explorations.com) is a security start-up company from Poland, providing various services in the area of security and vulnerability research. The company came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 50 security issues uncovered in the Java technology over the recent years. He is also the hacking contest co-winner and the man who has put Microsoft Windows to its knees (vide MS03-026). He was also the first one to present successful and widespread attack against mobile Java platform in 2004.