**Code Correctness: Byte to Array String Conversion**

Example of Fortify Finding -- The call to String() on line 1226 of AddressController.java converts a byte array into a String, which may lead to data loss.

**Explanation**

When data from a byte array is converted into a String, it is unspecified what will happen to any data that is outside of the applicable character set. This can lead to data being lost, or a decrease in the level of security when binary data is needed to ensure proper security measures are followed.

**Example**

The following code converts data into a String in order to create a hash.

*FileInputStream fis = new FileInputStream(myFile);*

*byte[] byteArr = byte[BUFSIZE];*

*...*

*int count = fis.read(byteArr);*

*...*

*String fileString = new String(byteArr);*

*String fileSHA256Hex = DigestUtils.sha256Hex(fileString);*

*// use fileSHA256Hex to validate file*

Assuming the size of the file is less than BUFSIZE, this works fine if the information in myFile is encoded the same as the default character set. However, if it's using a different encoding, or is a binary file, it will lose information. This in turn will cause the resulting SHA hash to be less reliable, and could mean it's far easier to cause collisions, especially if any data outside of the default character set is represented by the same value, such as a question mark.

**Recommendation**

Generally speaking, a byte array potentially containing noncharacter data should never be converted into a String object as it may break functionality, but in some cases this can cause much larger security concerns. In a lot of cases there is no need to actually convert a byte array into a String, but if there is a specific reason to be able to create a String object from binary data, it must first be encoded in a way such that it will fit into the default character set.

**Example**

The following uses a different variant of the API in Example 1 to prevent any validation problems.

*FileInputStream fis = new FileInputStream(myFile);*

*byte[] byteArr = byte[BUFSIZE];*

*...*

*int count = fis.read(byteArr);*

*...*

*byte[] fileSHA256 = DigestUtils.sha256(byteArr);*

*// use fileSHA256 to validate file, comparing hash byte-by-byte.*

In this case it is straightforward to rectify, since this API has overloaded variants including one that accepts a byte array, and this could be simplified even further by using another overloaded variant of DigestUtils.sha256() that accepts a FileInputStream object as its argument.

Other scenarios may need careful consideration as to whether it's possible that the byte array could contain data outside of the character set, and further refactoring may be required.

**References**

* http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/code\_correctness\_byte\_array\_to\_string\_conversion.html