Ben Brandhorst

SDEV 325 6380

Detecting Software Vulnerabilities

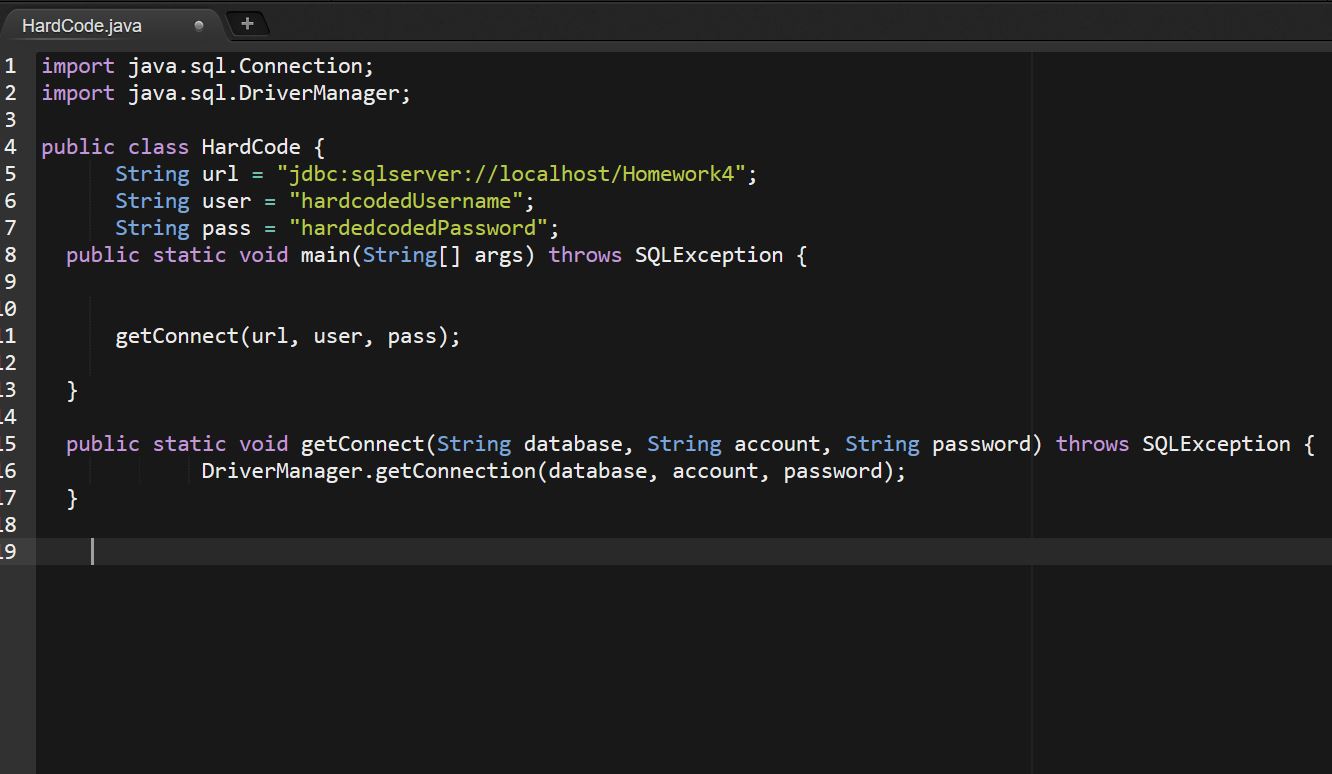
Professor Luke Donoho

09/28/2019

**Demonstrating Porous Defenses**

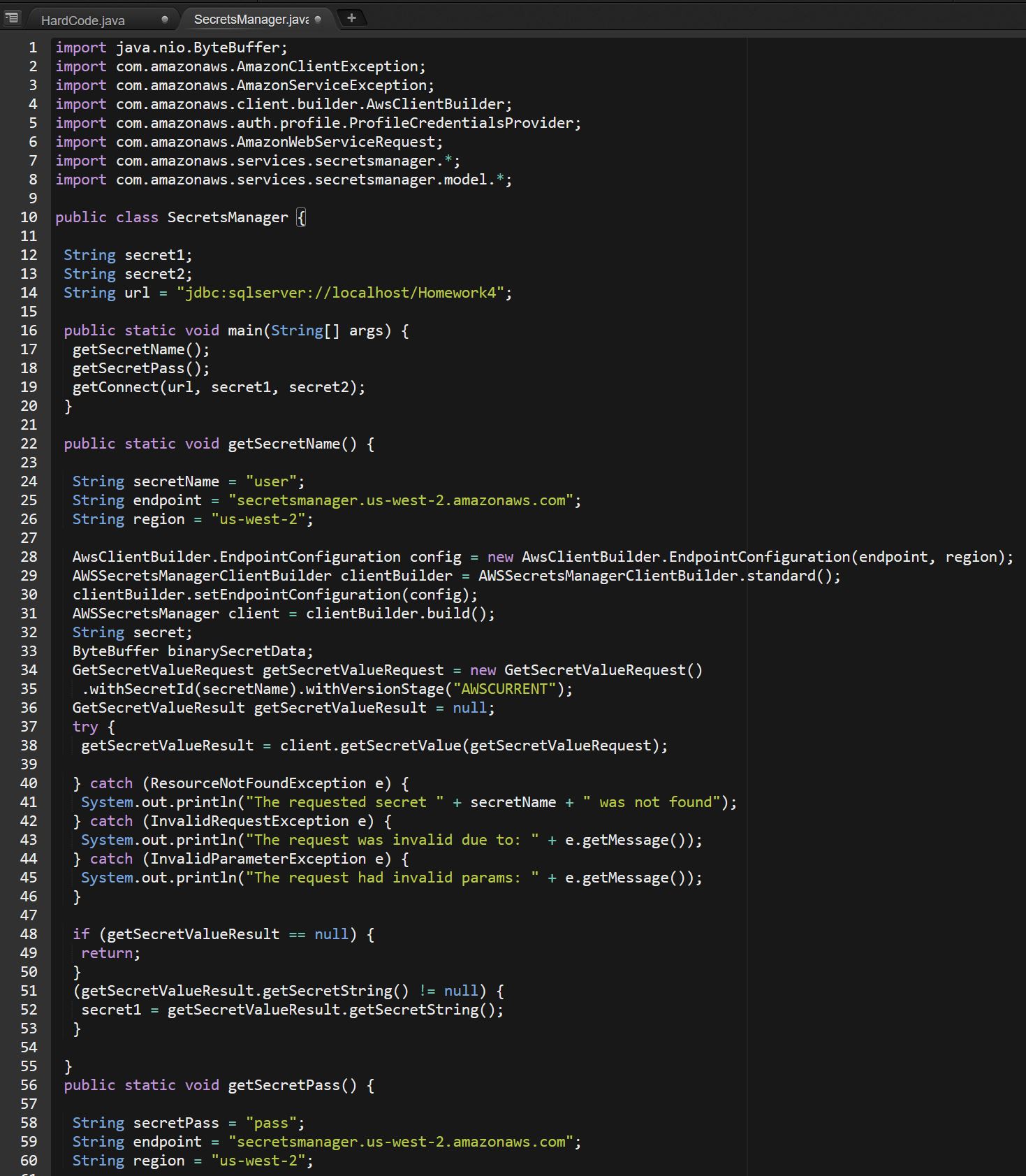
**Example #1**

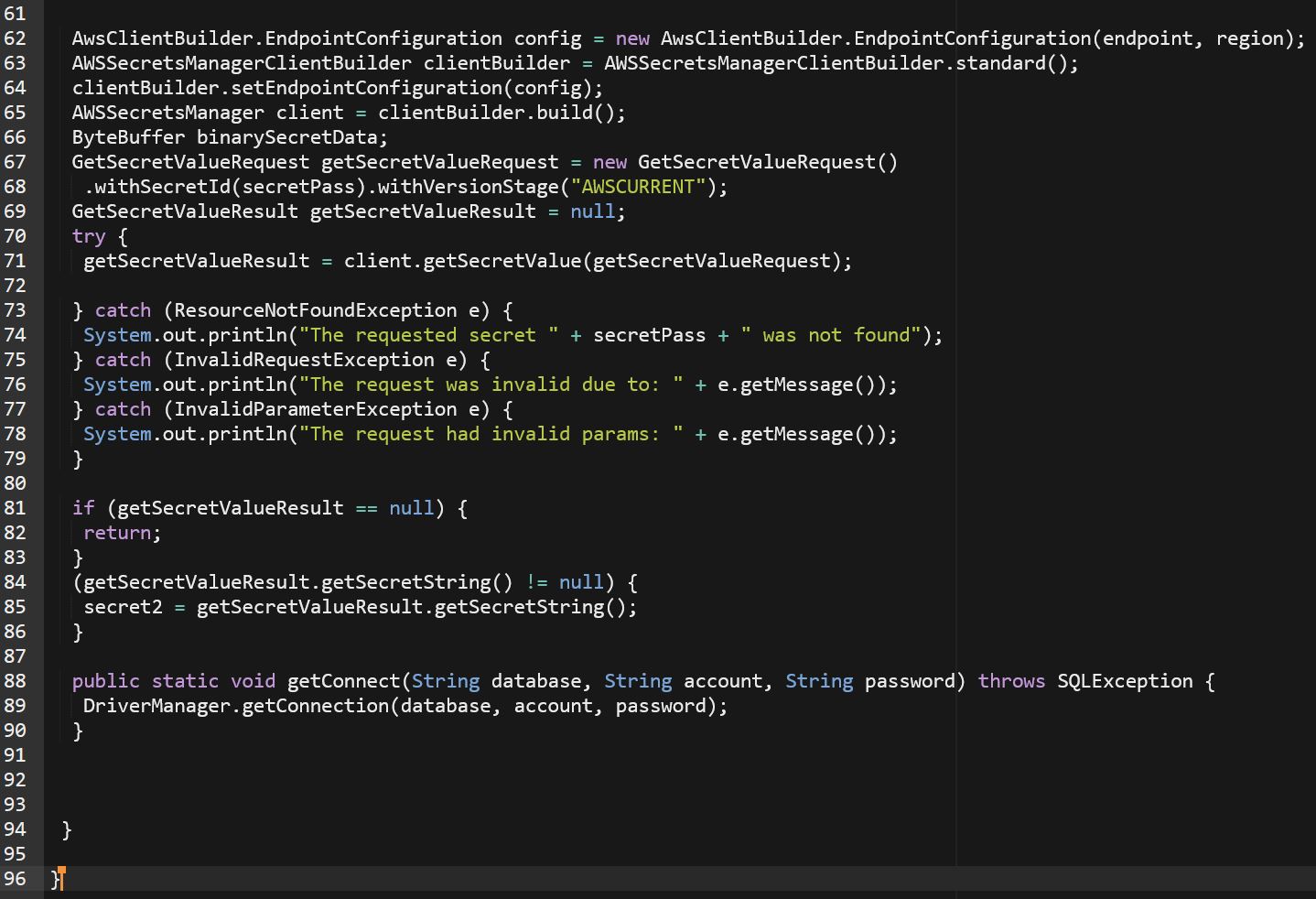
**Use of Hard-coded Credentials**

****

The above example shows a snippet of Java code that creates a SQL connection with hardcoded credentials for both the username and password. This is an example of the Outbound variant of CWE-798: Use of Hardcoded Credentials. The Outbound variant of this vulnerability tends to represent the larger of the two threats (compared to the Inbound variant) because any user of this program may be able to extract the hard coded credentials from the source code and have access to the back end with those credentials. Additionally, once the progrma is shipped, it’s impossible to modify the credentials without a patch.

**Mitigating with AWS Secrets Manager**

****

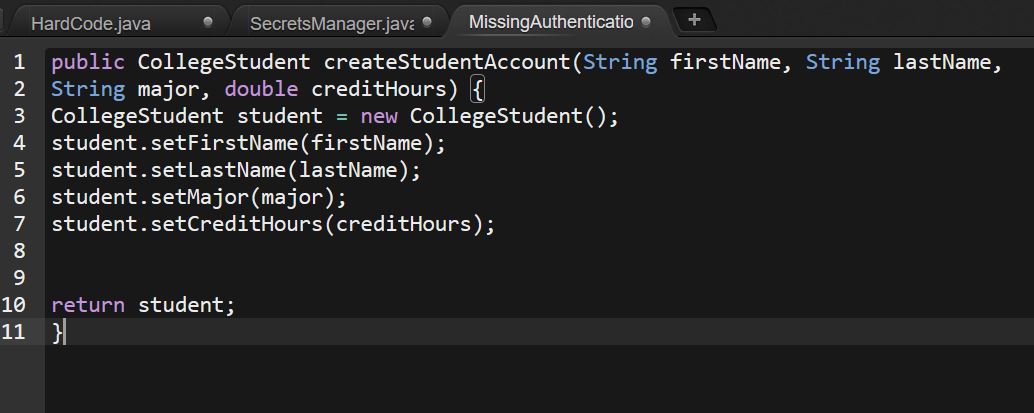


Attempts to mitigate against this vulnerability can lead down quite the rabbit hole. It seems there is no one right answer and they are all situation specific. In the end, since we are already using AWS, I decided to use another tool of theirs and go with AWS Secrets Manager. Here we store the SQL username and password within AWS Secrets Manager and then create a connection that retrieves the information and uses that retrieved info to make the SQL connection. This removes the hard coded vulnerability and allows us to update the credentials without needing to patch software since this information is stored separately from the client.

The other alternatives of using environmental variables, config files, and hashing or encrypting the credentials on the client side all had drawbacks that seem to introduce different vulnerabilities.

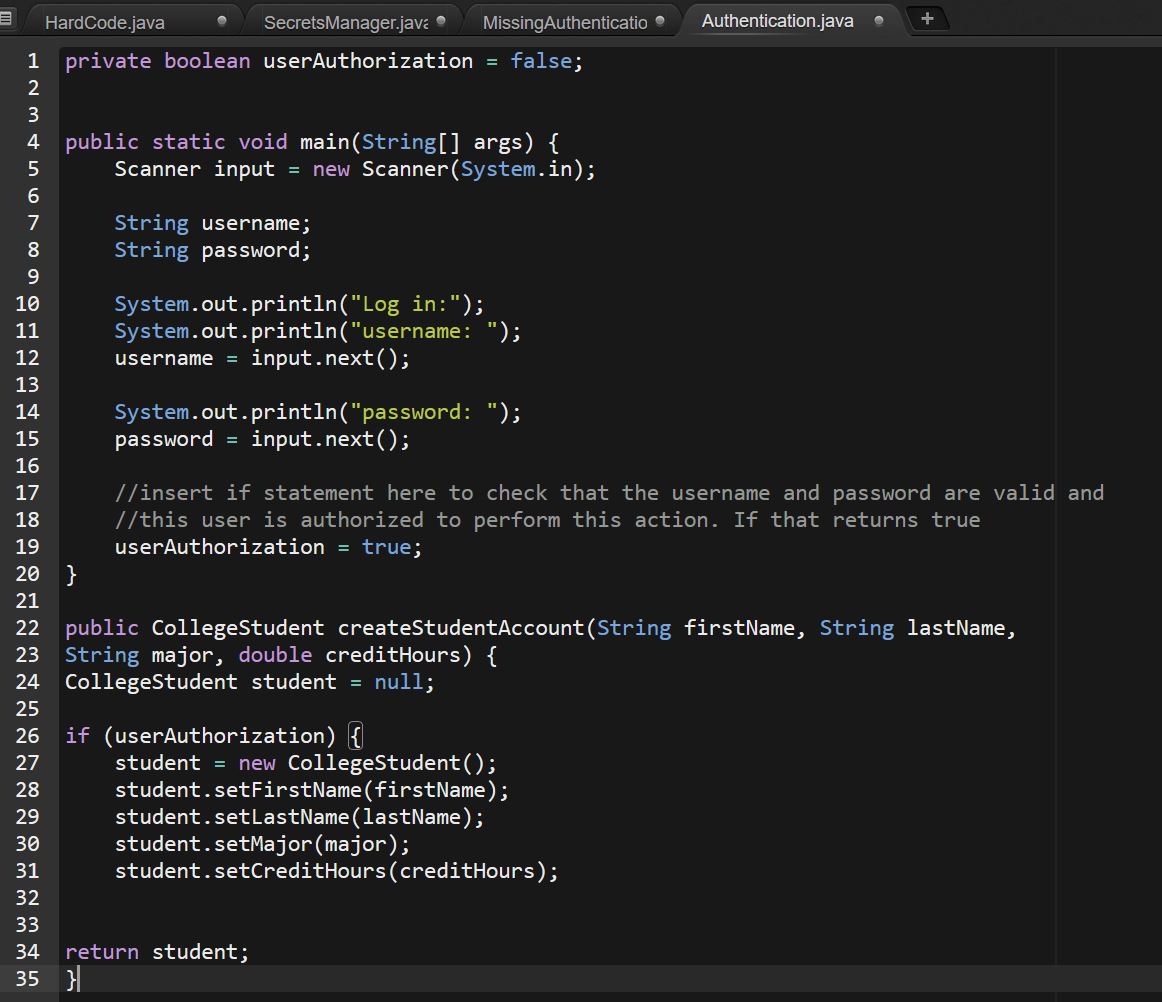
**Example #2**

**Missing Authentication for Critical Function**

****

In the above example the method createStudentAccount is part of a larger program which manages student accounts. Here there is no check to verify that the user is authorized to create new accounts and is susceptible to exploitation by users who should not be able to perform this action.

**Authentication Included**

****

The solution to this vulnerability is to include an authentication prior to allowing users to perform actions which should be privileged. In this case, this is done by using a Boolean which is only set to true if the user presents the correct credentials and is authorized to perform the actions requested.