**Configuring your Lab Environment**

**Fix your first Java Data Input Issues**

Benjamin M. Brandhorst

University of Maryland Global Campus

SDEV 425 7980 Mitigating Software Vulnerabilities

Doctor Ronald McFarland

March 29th, 2020

**Part 1: Environment Setup and Simple Hello, World**

I selected Netbeans for my Java development IDE because it is required for the final homework assignment in the course. I figure the earlier homework assignments will be less time intensive than the latter, so I might as well familiarize myself with the development environment now while I have more time to tinker around with settings. Figure 1 shows my Hello World program source code. Figure 2 shows the program output.

**A screenshot of a cell phone

Description automatically generated**

Figure 1 – Source code for my unique Hello World program

**A screenshot of a cell phone

Description automatically generated**

Figure 2 – Program Output

**Part 2: Fix Security Issues in a Simple Java Application**

SDEV425\_1 is a Java application that uses a command line argument to specify a file and then prints each line of that file using the Buffered Reader. While this program compiles and runs without errors, there are several vulnerabilities that have not been addressed. The Carnegie Mellon University Software Engineering Institute lists its Software Engineering Institute Computer Emergency Response Team (SEI CERT) Oracle Coding Standard for Java. Using examples gathered from those resources, I have listed the vulnerabilities I discovered in this program and what mitigation steps have been taken to correct these issues.

**FIO16-J. Canonicalize path names before validating them**

(Mohindra & Snavely, 2017) show us that because there are no checks against the arguments a user can provide, the program as written can be exploited using a directory traversal or path equivalence vulnerability. This is addressed through canonicalization of the user supplied arguments. Figure 1 shows the white listed file path used to check the canonicalized arguments against. Figure 2 shows the method employed to verify that the user supplied arguments fall within the acceptable parameters before the contents of the file are read.

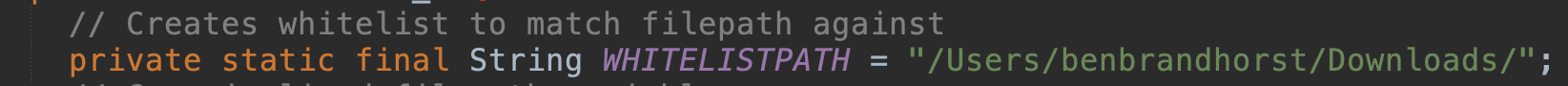


Figure 1 – Demonstrating WHITELISTPATH variable contents

A screenshot of a cell phone

Description automatically generated

Figure 2 – ConfirmPath method which canonicalizes arguments and limits file path to whitelist

**IDS51-J.** **Properly encode or escape output & IDS52-J. Prevent code injection**

Output sanitization is a necessity to prevent the introduction of malicious data. As written, the SDEV425\_1 program has no checks in place to sanitize input before being output by the program. This leaves it vulnerable to malicious data being unwittingly read and output. (Mohindra & Leung, 2018) write about how it is unsafe to assume that data is trusted to be output without checks because data may indirectly originate from untrusted resources. Our program is simple enough that there is no input validation on the content of the txt file because the contents are only used when they are read and printed. Because of this, it is necessary for us to confirm that the contents are valid and safe to display. We have accomplished through the use of a whitelist regular expression. Figure 3 shows the regex each line of the file being read is matched against before it is printed. This also serves to mitigate against code injection because each line is matched against a whitelisted regex. This ensures that only email address format text will be read.

A close up of a screen

Description automatically generatedFigure 3 – Regex example ("How to validate an email address using a regular expression?," n.d.)

**MSC05-J.** **Do not exhaust heap space**

(Chatnani & Lallier, 2015) tell us that a Java OutofMemoryError will happen if a program tries to use more heap space than is available. As written, our program does not check the size of the file before attempting to read its contents. Because there is no upper boundary set on memory space required to read files, the program is left vulnerable to heap space being exhausted. This is corrected by setting a limit to the file size being read. Figure 4 shows the FILELIMIT variable set to 1,000,000 bytes or 1 megabyte. Figure 5 shows the method used to confirm the file provided through user supplied arguments, is under the 1 megabyte limit before being read.

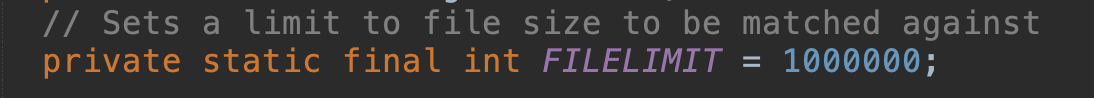


Figure 4 – FILELIMIT variable set to 1,000,000 bytes or 1 megabyte

A screenshot of a cell phone

Description automatically generated

Figure 5 – confirmSize method used to verify file is within appropriate size range before being used

**Part 3: Demonstrate Mitigations Used**

**FIO16-J. Canonicalize path names before validating them**

The only file path that can be used for arguments must include the following directory: **/Users/benbrandhorst/Downloads.** If a user supplies an argument that does not include that path, they will receive an error. Figure 6 shows an example where I’ve changed the argument file path to be /Users/benbrandhorst/UMUC/SDEV425/Homework1/EmailAddresses.txt.

A screenshot of a cell phone

Description automatically generatedFigure 6 – Non whitelisted file path example

**IDS51-J. Properly encode or escape output & IDS52-J. Prevent code injection**

The program will only print out lines in files that are in an email format. Figure 7 shows what happens when I modify the EmailAddresses.txt file to include lines of data in another format. This use of a regex whitelist also protects against code injection.

A screenshot of a cell phone

Description automatically generated

Figure 7 – Introduction of non whitelisted text format to file

**MSC05-J. Do not exhaust heap space**

The program will only accept files of a size less than 1 megabyte. If the file size is greater, the user will receive an error indicating that their file is too large. Figure 8 shows what happens when I attempt to open an extremely large text file. LargeEmail.txt is a 77 megabyte text file created by copying and pasting the original EmailAddresses.txt file contents over and over.

A screenshot of a cell phone

Description automatically generated

Figure 8 – Example of output when trying to read a large file

References

Chatnani, K., & Lallier, K. (2015, July 3). MSC05-J. Do not exhaust heap space - SEI CERT Oracle coding standard for Java - Confluence. Retrieved from https://wiki.sei.cmu.edu/confluence/display/java/MSC05-J.+Do+not+exhaust+heap+space

How to validate an email address using a regular expression? (n.d.). Retrieved from https://stackoverflow.com/questions/201323/how-to-validate-an-email-address-using-a-regular-expression

Mohindra, D., & Leung, D. (2018, November 8). IDS51-J. Properly ENCODE or escape output - SEI CERT Oracle coding standard for Java - Confluence. Retrieved from https://wiki.sei.cmu.edu/confluence/display/java/IDS51-J.+Properly+encode+or+escape+output

Mohindra, D., & Snavely, W. (2017, January 5). FIO16-J. Canonicalize path names before validating them - SEI CERT Oracle coding standard for Java - Confluence. Retrieved from https://wiki.sei.cmu.edu/confluence/display/java/FIO16-J.+Canonicalize+path+names+before+validating+them

Mohindra, D., & Snavely, W. (2017, November 16). IDS52-J. Prevent code injection - SEI CERT Oracle coding standard for Java - Confluence. Retrieved from https://wiki.sei.cmu.edu/confluence/display/java/IDS52-J.+Prevent+code+injection