This document includes execution instructions, results, and analysis for homework 3. Parts 1 through 3 involved writing software and manipulating compiled classes. Source code is included as HW3\_dwilso.zip. Additionally, a compiled jar is provided, ‘java\_security\_hw3.jar’. Execution directions in this readme depend on this jar as well as a folder containing modified class files.

# Part 1: Byte code verification

## Execution:

Modified classes can be executed using a helper class. The argument supplied to the help runs a specified modified class. From the folder ‘modifiedClasses’ within ‘java\_security\_hw3’, execute the following:

java -cp . java\_security\_hw3.part1.VerifierHelper <number of Pass to run>  
  
Example:

java -cp . java\_security\_hw3.part1.VerifierHelper 11

## Code:

Original classes can be found in ‘java\_security\_hw3/src/main/java/java\_security\_hw3/part1’

## Output and Discussion:

### Pass 11 and 12

#### Fails:

Both fail in pass 1

#### Rule broken:

Pass11 has a corrupt constants section. This was achieved by removed a constant entry.  
Pass12 has a major version which is unsupported in java 8. The major version was made 55.

### Pass 22 and 22

#### Fails:

Both fail in pass 2

#### Rule broken:

Pass21’s superclass was completely removed. Because of this, it cannot be loaded and fails.  
Pass22 was made to extend a final class by editing the bytecode. Extending final is not allowed.

### Pass 31 and 32

#### Fails:

Both fail in pass 3

#### Rule broken:

Pass31’s parameters were removed from the ‘add’ method. Helper cannot call this method now.  
Pass32’s field x was changed from an int to a string. This causes NoSuchFieldError in constructor.

### Pass 41 and 42

#### Fails:

Both fail in pass 4

#### Rule broken:

Pass41’s ‘printHello’ method is completely removed. This causes a runtime no method error.  
Pass42’s class name was changed to PassNot42. This causes a runtime NoClassDefFoundError.

# Part 2: Java annotations

## Code:

Annotations and their use can be found under ‘java\_security\_hw3’ at ‘/src/main/java/java\_security\_hw3/part2’ The class Zeta is annotated as required in the assignment.

## Usefulness:

Annotations are useful in Java for easily extending capabilities of classes. Documentation, code generation, and object relation mapping have all been done using Java’s annotation facilities. From a security perspective, annotations can be useful for defining how a class and its methods can be used. For example, an annotation can be defined as above for ‘authorizationCheck’. This could mean that an authorization check must be performed a method is called. It could also mean that the method does an authorization check so one is not necessary before calling said method. The same mechanics could be implemented for auditing and locking as used in the assignment. One definition of security based annotations is JSR 250, included in Java SE version 6. These define role based access controls, resource handling, processing flow when using and injection, as well as other functionality.

# Part 3: Java annotations

## Code:

The DOSPuzzle class can be found in ‘java\_security\_hw3’ under ‘src/main/java/java\_security\_hw3/part3’ It is a single class with a single main method for executing a demo of a DOS puzzle. It uses SHA1 hashing as described in the assignment for generating increasingly computationally difficult ‘puzzle’s to be solved by a client.

## Execution:

The DOSPuzzle demo class can be run using the supplied jar file in ‘java\_security\_hw3/bin/hw3.jar’

java -cp hw3.jar java\_security\_hw3.part3.DOSPuzzle

## Description:

The DOSPuzzle class runs a simple DOS puzzle solver simulation. A hash is computed from a string of bits. The hash and some ordered subset of the bits, starting from the beginning of the string is supplied to a client. This client then attempts to find the missing bits. These are then return to the server and verified. The client in this simulation performs a brute force search, trying all possible combinations until the hash is found.

## Output:

Running part 3 - DOS Puzzle

Sending puzzle with 5 missing bits:

Proposed solution [01010] is correct!

Sending puzzle with 6 missing bits:

Proposed solution [001010] is correct!

Sending puzzle with 7 missing bits:

Proposed solution [1001010] is correct!

Sending puzzle with 8 missing bits:

Proposed solution [11001010] is correct!

Sending puzzle with 9 missing bits:

Proposed solution [111001010] is correct!

Sending puzzle with 10 missing bits:

Proposed solution [0111001010] is correct!

Sending puzzle with 11 missing bits:

Proposed solution [10111001010] is correct!

Sending puzzle with 12 missing bits:

Proposed solution [010111001010] is correct!

Sending puzzle with 13 missing bits:

Proposed solution [1010111001010] is correct!

Sending puzzle with 14 missing bits:

Proposed solution [01010111001010] is correct!

Sending puzzle with 15 missing bits:

Proposed solution [101010111001010] is correct!

Sending puzzle with 16 missing bits:

Proposed solution [1101010111001010] is correct!

Sending puzzle with 17 missing bits:

Proposed solution [11101010111001010] is correct!

Sending puzzle with 18 missing bits:

Proposed solution [011101010111001010] is correct!

Sending puzzle with 19 missing bits:

Proposed solution [1011101010111001010] is correct!