1 Legend

- TODO for Adam and Jinghao
- TODO for Adam
- TODO for Jinghao
- TODO for Adam and Adrienne

2 Outline

1. Introduction
2. Background/Motivation
   (a) Need Taint Tracking for database input sanitization
   (b) Character level improves precision of taint tracking and propagation (fewer false positives)
   (c) * Insert additional stuff from Erika's paper here *
3. Related Works
   (a) * Insert related papers that we've read here *
4. Basic Implementation
   (a) * Describe Erika's work here *
   (b) Modifications to String Class
   (c) Source Tainting and propagation
5. Optimizations
   (a) Motivation for optimizations
      i. Intuitively, most strings either have none or one taint interval
         - All foreign input fully tainted at source
         - Substring of single-interval strings have at most one taint interval
         - Concatenation of multiple strings where there is only one taint interval results in a string with a single taint interval
         - All immediate derivations from fully-tainted or untainted strings have either a single taint interval or none

      ii. Strings that have multiple taint intervals tend to be longer
         - A bitmap is a more compact and efficient representation
         - Doing group operations on bitmaps is faster than manipulating individual boolean array elements
      iii. The very few strings with multiple taint intervals may have an unbounded number of intervals. Those pathological cases must be handled gracefully. Example:

```
for (String s : user_input)
  result += s + ","; // s is fully tainted, but the comma is not
```

   (b) Representation: TaintSet data structure
      i. Single Interval
      ii. Integer-array Bitmap
      iii. Special case: Full Interval

6. Profiling
   (a) Measuring taint properties of Strings
      i. Measure the number of taint intervals of strings in calls to string methods
      ii. Histogram based on use (invocation of the listed methods) not existence or creation
      iii. Retrofitted Erika's code because counting distinct intervals in boolean arrays easier
   (b) Measuring frequency of method calls
      i. Updated static hash table with each call to a retrofitted String method
      ii. Goal: Determine where to focus optimizations
   (c) Methodology and Details
      i. Track calls to the following methods in String
         A. `replace(CharSequence, CharSequence)`
         B. `substring(int, int)`
         C. `substring(int)`
         D. `toLowerCase(Locale)`
         E. `toUpperCase(Locale)`
         F. `trim()`
         G. `concat(String)`

H. `matches(String)`

I. `split(String, int)`

J. `split(String)`

K. `replaceAll(String, String)`

L. `replaceFirst(String, String)`

ii. Static methods in String: Upon first invocation, start a "logger" thread that periodically writes to log files

- No synchronization tools ("synchronized") is used because the cost of a loss of a couple counts is minor

iii. Performed a sequence of actions on JForum running on a server with a retrofitted String class:

A. Visit the home page

B. Click on a forum to view its index page

C. Click on a topic with many posts

D. Click reply

E. Post reply

iv. Inspected log file after each action and kept track of changes

(d) Results

i. Method Calls

Method	**A**	**B**	**C**	**D**	**E**
replace	0	0	0	0	0
substring	1825	6765	4619	136	4697
toLowerCase	78	885	734	20	996
toUpperCase	1	12	13	0	10
trim	1533	1111	2	0	144
concat	99	234	0	0	112
matches	0	0	0	0	0
split	0	0	0	0	0
replaceAll	0	0	0	0	0
replaceFirst	0	0	0	0	0

ii. Taint histogram

Taint Interval Counts	**A**	**B**	**C**	**D**	**E**
0	3733	11481	7563	156	7577
1	11	11	13	0	76
More than 1	0	0	0	0	0

(e) Future work

        i. Track StringBuffer/StringBuilder. Possible explanation: Most concatenation invocations are from Buffer/Builder (For example, the automatic conversion of + to append calls)
        ii. Track taint histogram for each method. Perhaps the calls to substring() are not problematic despite the sheer count because those calls typically are on untainted Strings.

7. Testing Methodology and Benchmarks
    (a) Test Setup
        i. running on gradgrind * get stats *
        ii. consistent machine for both micro and macro tests
    (b) Microbenchmarking
        i. Methodology
            A. Testing for steady state since servers will be in such a state
            B. Using Benchmark framework * cite source *
                - Run framework for x (determine actual number) runs and average over those runs
                - Framework automatically runs until steady state per test
                - Each run is another instantiation of the VM running a set of tests until steady state/test
            C. Tested from command line, switched out VMs to test different versions
                - IBM-JAVA
                - Basic Implementation (Erika)
                - Set Version
            D. Test various string class methods
                - String Creation
                - Concatenation
                - Substring
                - Trim
                - Case Changing
                - Replacement
            E. Test Various length strings
                - 12

        - 1024
      F. Test Various taint versions for taint-capable VMs
        - No taint
        - full taint
        - 1/2 taint
        - 2/3 taint
   (c) Benchmarks
      i. * show them *
      ii. Remarks
         A. Explanation for beating ibm-java in certain cases...
   (d) Macrobenchmarking
      i. Methodology - Latency Testing
         A. Measure time from receiving request to finish servicing request
         B. Use JForum to simulate webapp with medium reading to writing load
         C. Testing POST/GET requests on a pre-populated JForum database
         D. Server to Server on LAN * what is capable speed? *
         E. Client:
            - Server with a python script being invoked from the command line
         F. Server:
            - Server running Tomcat instances with a JForum application (only one running at a time)
            - One version is standard Tomcat6 running IBM-JAVA, other is a modified version to
            - Taint sources running either Erika's VM or the TaintSet VM
            - Mysql instance to handle database requests
      ii. Benchmarks
         A. * show them *

8. Conclusions/Future Work
   (a) Uppercase/lowercase
   (b) More comprehensive macrobenchmarks