Student Name: Vishisht Khilariwal
Roll Number: 2014120

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering

on 5th July, 2017

BTP Track: Research

BTP Advisor
Dr. Rahul Purandare

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **Improving Memory Utilization and Security of Software Applications by Performing Static Program Analysis and Program Transformation** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Rahul Purandare**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

....
Vishisht Khilariwal

Place & Date: 5th July 2017, New Delhi

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

....
Dr. Rahul Purandare

Place & Date: 5th July 2017, New Delhi

Abstract

Memory or RAM of a device contains a lot of data.This work focuses on how to ensure security of our critical data like our passwords or encryption keys stored in the memory which if become public can pose a big security as well as privacy risk. Also our aim is to develop different techniques to perform memory optimisations.

Keywords: Security, Static Analysis, Memory Optimisation, Android Applications, Java, StringBuilder, String

Acknowledgments

I would like to take this opportunity to thank every individual who has assisted me in undertaking this work, specifically, Dr. Rahul Purandare for being an invaluable mentor and guide and my friends and family for their constant support throughout.

Contents

Chapter 1

Introduction

Memory of a device usually contains a lot of critical data like passwords, user-names, encryption keys etc. Attacks like memory dump attack can leak this critical information to a potential adversary, so the longer this data stays in memory, bigger is the risk of this data being leaked out, creating a big security as well as a privacy risk. So we need to minimise the time such data stays in the memory.
A lot of this critical data is stored in the form of strings in Java which are immutable. So once this data is created it will remain in Java's String pool till it is collected by the garbage collector. So we need an alternative data structure that can be used to replace strings for such critical data. This data structure needs to have the property of being able to reset so that it leaves no trace of the stored data.
We also need to figure out the last read and all the references of these critical values in a program or a block of code. This will enable us to get rid of these values at the earliest without impacting the task to be performed by the program. By performing Static Program Analysis and Program Transformations we plan to optimise the code such that it remains the same semantically but decreases the threat of such attacks.

1.1 Static Program Analysis & Program Transformations

Static Analysis is method of analysing code without actually having to run the code. This technique can be used to identify a wide spectrum of things in a piece of code ranging from identifying the last read or write of a variable to identifying different references to the same variable. Information gathered by Static Analysis can be used to perform optimisations in the code and pinpoint the exact locations of issues.
Program Transformation is the process of converting a block of code into another semantically similar piece of code which is better at achieving the desired goals which in our case is improved level of security.

1.2 Threats

Memory or RAM of a computer can contain a lot of sensitive data. This data can be leaked to an adversary creating privacy and security concerns. Memory dumps are a way for the developers to troubleshoot the issue that caused a program to fail or crash. These memory dumps can contain anything that was in the active part of the memory like passwords or other critical data. An adversary can initiate a memory dump (for example by buffer overflows) and can gain access to the contents of the memory. Other methods of executing such attacks also exit like the Direct Memory Access Attack, Heartbleed bug (bug in OpenSSL library which could potentially leak the contents of memory).

1.3 Memory Optimisation

Optimising memory usage of an application is one the most effective ways to imporve its performance. Frequent allocation/ deallocation of memory as well as memory hungry instances can slow down the execution of the processes. It is therefore necessary to keep looking for new techniques which allow us to optimise memory comsumption.

Chapter 2

Motivation & Research Problem

Sensitive data should be cleared as soon as possible after its usage to prevent it from getting exposed through memory dump attacks. We propose an approach that will perform a static analysis to keep track of the sensitive data propagation in the program and identify variables that need to be reset immediately after their last usage on all program paths. We also develop techniques to synthesize mutable classes that are more-memory efficient and provide the same functionality of their immutable counter-parts, and then use them in the programs in place of their counter-parts. As a result we make a two-fold contribution: first, make a program more secure and second,more memory-efficient.

Chapter 3

Work Done in Winter 2017

3.1 Research Approach

As a first step, we focused on android applications to see which Java classes are performance bottlenecks. It was a study of 20 popular open-source apps and profiling them using Android Studio Memory Monitor. This gave us an idea about which classes perform frequent memory allocation/deallocation and consume most of the memory (which are a known performance bottleneck).

Java was the main focus of the research because:

- Strings in Java is a immutable class because every instance of string is stored in the String pool of Java and is not destroyed. It is important to handle strings so that leakage of critical data can be prevented.
- Java is the language used to create Android applications as well as many other widely used applications.

Open source apps were used because of 2 main reasons

- Code of the private APK's cannot be analysed.
- Memory monitor of Android Studio only allows taking memory snapshot of open-source apps since they can be localy compiled and run on an Android Device.

3.2 Tools Used

3.2.1 Android Studio

Android Studio is the official IDE for Android. It was used to compile and run the open-source applications on the Android Mobile Devices. It provides a host of other tools to perform heap analysis.

3.2.2 Android Studio Memory Monitor

Android Monitor provides a Memory Monitor so you can more easily monitor app performance and memory usage to find deallocated objects, locate memory leaks, and track the amount of memory the connected device is using. The Memory Monitor reports how your app allocates memory and helps you to visualize the memory your app uses. This tool was used to take memory snapshots which were later analysed.

3.2.3 HPROF Viewer & Analyser

The HPROF Viewer displays Class names which are responsible for the memory, total number of instances of a class, size of an instance in the heap, reference tree etc. This tools also allows to analyse the memory for memory leaks as well as things like duplicated strings.

3.2.4 Android Instrumentation Tests

Instrumented unit tests are tests that run on physical devices and emulators, and they can take advantage of the Android framework APIs and supporting APIs, such as the Android Testing Support Library. We used the Android Instrumentation Tests to ensure that we could replicate the functioning of an app before taking memory snapshots so that the comparison between the heaps before and after the optimisations would be comparable. These Android Instrumentation Tests were developed by the developers of the code themselves and bundled together with the application.

3.3 Work Done

3.3.1 Selecting Sample

We selected about 20 open-source Android apps all of which had Android Instrumentation tests provided by the developers. The basic threshold for selecting the apps was that every app should have at least more than 5000 downloads. The sample included a wide variety of apps ranging from web browsers to chatting applications and blogging apps.Following are a few of the apps that were analysed.

- Wikipedia
- DuckDuckGo
- Bitcoin-wallet
- WordrPess
- orbot

- Wire
- JumpGo
- Lightning Browser
- F-Droid
- Flym

3.3.2 Method of Analysis

These Android apps were installed and run on Android devices. The Android Instrumentation Test suites which perform some predefined activities specific to each application were run for each of the apps. Heap snapshots were taken using the Android Memory Monitor at the beginning, end and during the running of the Android Instrumentation Tests. The memory snapshots were then later analysed to identify the most common classes and data structures that were responsible for hogging the majority of the memory. Also the HPROF analyzer was used to identify any memory leaks or duplicate instances that might have crept in during the execution.

3.3.3 Results of Analysis

The analysis revealed that String, character arrays and integer arrays featured amongst the most common, high memory using data structures across all the apps.It also showed that during the execution of these apps, a lot of duplicate strings were being produced. Based on these results a couple of optimisations were proposed to decrease the number of duplicated strings in the code and decrease the memory usage of character arrays by using static analysis techniques to identify the places where these strings were being generated and force the program to check if the same string already exists in the String Pool of java and trying to replace the char arrays in the code with byte arrays.

3.4 Future Plans

Moving focus from memory optimisation to the security aspect we plan to study the implementation of Java classes like Stringbuilder, strings and AbstractStringbuilder and develop a custom class which will inherits the qualities of both strings and Stringbuilder. This would allow us to reset the critical data to null unlike strings in Java which are immutable and remain in the memory until it is removed by the garbage collector. This custom class would be used in place of strings allowing us to remove the data from the memory as soon as it is no longer needed. We also plan to develop an analysis which would enable us to identify the places which are safe to replace the string with our custom stringbuilder class without any security risks. The custom stringbuilder class would implement a reset function which would effectively remove the data from the memory by making the values null.

Chapter 4

Custom String Builder

4.1 Implementation

We created to classes to achieve our goals.

- customStringBuilder
- customCharSequence

4.1.1 customStringBuilder

customStringBuilder is the main class which holds the data in a character array and provides the user with different functions like append, delete, replace etc.

4.1.2 customCharSequence

Helper class which allowed us to implement our own private functions since several important functions like String.lastIndexOf() are not public and can only be used by classes inside the java.lang package.

4.2 Java StringBuilder vs customStringBuilder

StringBuilder uses a character array to store the data. When we append or insert some new data to the StringBuilder, it may overflow the size of the character array. To prevent this overflow StringBuilder checks the size of the input and if required creates a new character array (which is double the size of the previous array) to accommodate the new data. The old array is dereferenced and just sits in the memory with the data. This process repeats every time the capacity needs to be increased and creates a threat since the memory gets filled with old arrays containing multiple copies of the data.

In customStringBuilder we altered this process and made sure that every time such expansion of the capacity takes place we set every bit of the old array to null ('\0') before it is dereferenced and marked for garbage collection.

```
void expandCapacity(int minimumCapacity) {
    int newCapacity = (value.length + 1) * 2;
    if (newCapacity < 0) {
        newCapacity = Integer.MAX_VALUE;
    } else if (minimumCapacity > newCapacity) {
        newCapacity = minimumCapacity;
    }
    char[] temp = value;
    value = Arrays.copyOf(value, newCapacity);
    for(int i=0; i<temp.length; i++)
    {
        temp[i]='\0';
    }
}
```

We also implemented a function setToNull which would can be used when the data is no longer needed and needs to be destroyed. This functions sets every byte of the character array to null('\0').

Chapter 5

Results

5.1 Method Of Testing

We ran some basic test programs to validate the working of our customStringBuilder class. We took some heap dumps of the running java programs while using the Java StringBuilder and then using customStringBuilder.
Java heap dumps were taken using this command.

```
jmap -dump:live,file=heapdump4.bin <pid>
```

Live memory analysis was also done using the JavaVisualVM. (Figure 5.1 and 5.2)
Taking heap dumps and analysing the resulting .bin files showed that Java StringBuilder created a lot of dirty copies of data while expanding capcity but customStringBuilder used a more safe function to expand capacity and a did not create any dirty copy.

5.2 Issues faced

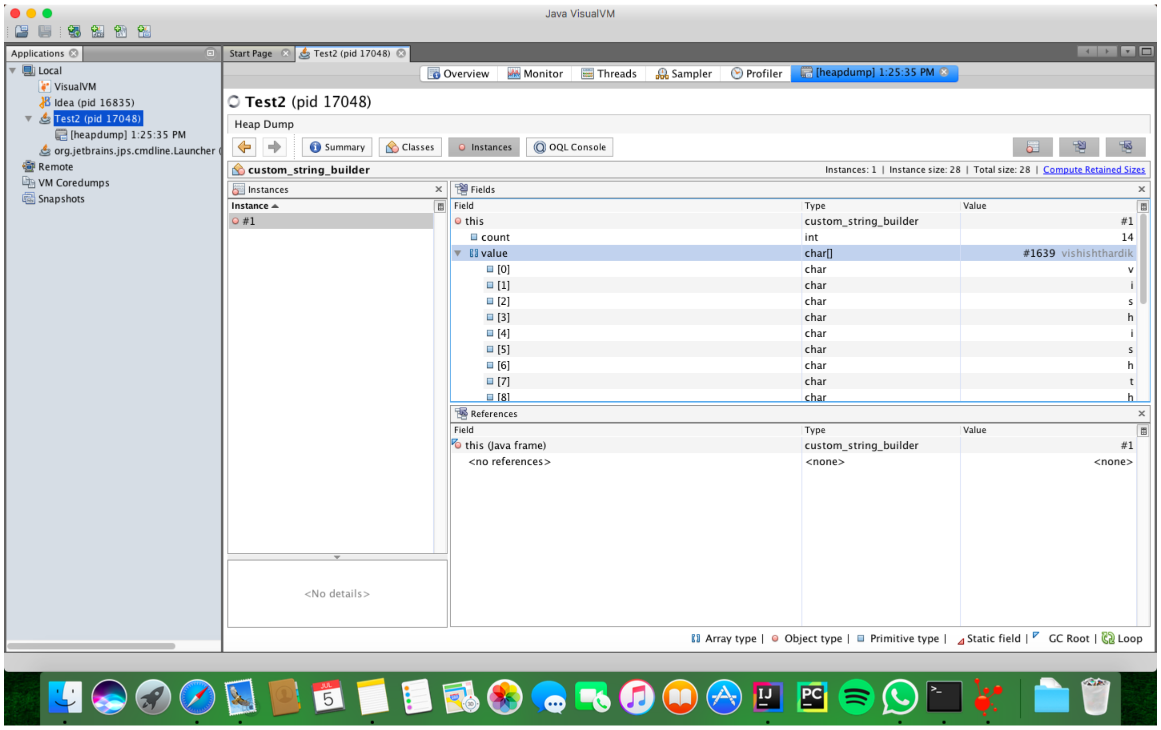
customStringBuilder worked as expected when used with char arrays, charSequence, stringBuffer etc. But custom failed when the values were passed to the customStringBuilder using a string literal. For example the code below works as expected.

```
custom_string_builder sb = new custom_string_builder();
char c[] =
{'v', 'i', 's', 'h', 'i', 's', 'h', 't', 'h', 'a', 'r', 'd', 'i', 'k'};
sb.append(c);
```

But the below code fails to remove the data using setToNull function.

```
custom_string_builder sb = new custom_string_builder();
sb.append("This is a test String");
```

The reason for this unexpected failure is that when we add ”This is a test String” to our customStringBuilder Java treats this literal as a string object and adds it to the Java string pool. And therefore even after using setToNull we can find this data in the memory dumps.



Local
  VisualVM
  Idea (pid 16835)
  Test2 (pid 17048)
    [heapdump] 1:25:35 PM
  org.jetbrains.jps.cmdline.Launcher (
Remote
VM Coredumps
Snapshots

Test2 (pid 17048)

Heap Dump

custom_string_builder — Instances: 1 | Instance size: 28 | Total size: 28 | Compute Retained Sizes

Instances

Instance ▲
#1

<No details>

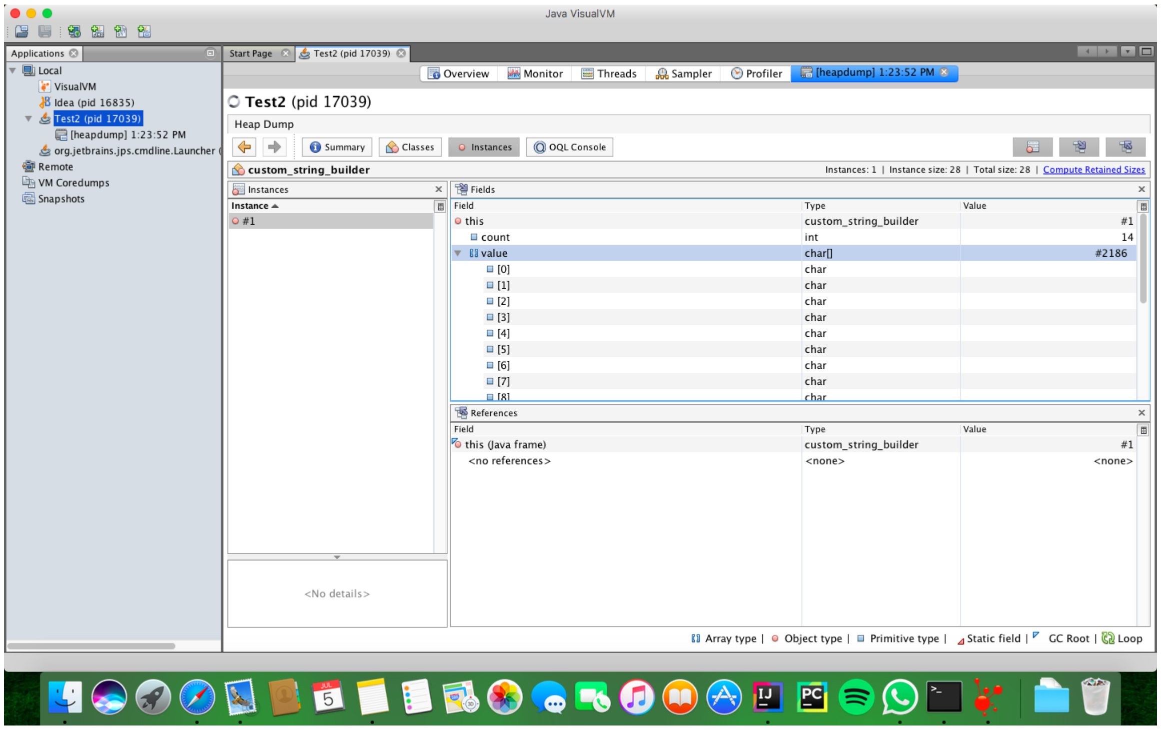
Fields

Field	Type	Value
this	custom_string_builder	#1
count	int	14
value	char[]	#1639 vishishthardik
[0]	char	v
[1]	char	i
[2]	char	s
[3]	char	h
[4]	char	i
[5]	char	s
[6]	char	h
[7]	char	t
[8]	char	h

References

Field	Type	Value
this (Java frame)	custom_string_builder	#1
<no references>	<none>	<none>

Figure 5.1: Without setToNull().



Local
  VisualVM
  Idea (pid 16835)
  Test2 (pid 17039)
    [heapdump] 1:23:52 PM
  org.jetbrains.jps.cmdline.Launcher (
Remote
VM Coredumps
Snapshots

Test2 (pid 17039)

Heap Dump

custom_string_builder — Instances: 1 | Instance size: 28 | Total size: 28 | Compute Retained Sizes

Instances

Instance ▲
#1

<No details>

Fields

Field	Type	Value
this	custom_string_builder	#1
count	int	14
value	char[]	#2186
[0]	char	
[1]	char	
[2]	char	
[3]	char	
[4]	char	
[5]	char	
[6]	char	
[7]	char	
[8]	char	

References

Field	Type	Value
this (Java frame)	custom_string_builder	#1
<no references>	<none>	<none>

Figure 5.2: With setToNull().

Chapter 6

Future Plans

- Implement similar functions and classes in C/C++
- Identify the tool to be used for analyzing these classes. Weigh between the pros and cons of using Clang or LLVM.
- Find appropriate c/c++ benchmarks to test and compare the performance benefits.
- Write a tool for analysis.