Improving Memory Utilization and Security of Software Applications by Performing Static Program Analysis and Program Transformation

Motivation

Memory of a device may contain a lot of critical data like password, encryption keys etc. during the execution of the code.

Memory dump attacks, HeartBleed bug etc. are some of the threats which can leak this critical data.

Improving memory utilization of a process allows to remove critical data as well as improve the performance of the code.

Problem Statement

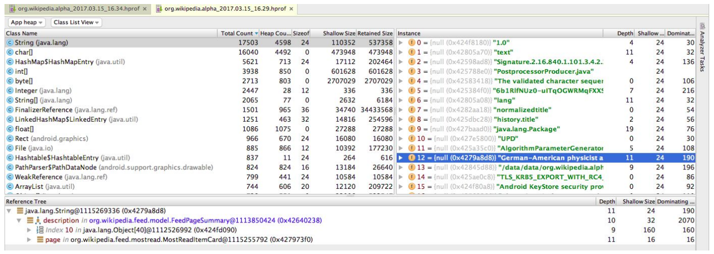
Sensitive data should be cleared as soon as possible after its usage to prevent it from getting exposed. We develop techniques to synthesize mutable classes that are more-memory efficient and provide the same functionality of their immutable counter-parts, and then use them in the programs in place of their counter-parts.

We also 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.

Work Done

20 open-source Android applications (Wikipedia,, DuckDuckGo etc.) were selected for analysis.

Memory utilization of these apps was analysed by running them with Instrumentation tests and taking heap snapshots.



org.wikipedia.alpha_2017.03.15_16.34.hprof × | org.wikipedia.alpha_2017.03.15_16.29.hprof ×

App heap | Class List View

Class Name	Total Count	Heap Cou...	Sizeof	Shallow Size	Retained Size
String (java.lang)	17503	4598	24	110352	537358
char[]	16040	4492	0	473948	473948
HashMap$HashMapEntry (java.util)	5621	713	24	17112	202464
int[]	3938	850	0	601628	601628
byte[]	2713	803	0	2707029	2707029
Integer (java.lang)	2447	28	12	336	336
String[] (java.lang)	2065	77	0	2632	6184
FinalizerReference (java.lang.ref)	1501	965	36	34740	34433568
LinkedHashMap$LinkedEntry (java.util)	1251	463	32	14816	254596
float[]	1086	1075	0	27288	27288
Rect (android.graphics)	966	670	24	16080	16080
File (java.io)	885	866	12	10392	177230
Hashtable$HashtableEntry (java.util)	837	11	24	264	616
PathParser$PathDataNode (android.support.graphics.drawable)	824	824	16	13184	26640
WeakReference (java.lang.ref)	799	441	24	10584	10584
ArrayList (java.util)	744	606	20	12120	209722

Instance	Depth	Shallow ...	Dominat...
0 = {null (0x424f8180)} "1.0"	4	24	30
1 = {null (0x42805a70)} "text"	11	24	32
2 = {null (0x42598ad8)} "Signature.2.16.840.1.101.3.4.2.	4	24	136
3 = {null (0x425788e0)} "PostprocessorProducer.java"		24	
4 = {null (0x42583418)} "The validated character sequer	0	24	106
5 = {null (0x425384f0)} "6b1RlfNUz0-uITqOGWRMqFXX	7	24	216
6 = {null (0x42805a08)} "lang"	11	24	32
7 = {null (0x4282aa18)} "normalizedtitle"	0	24	54
8 = {null (0x425dbc28)} "history.title"	2	24	56
9 = {null (0x427baad0)} "java.lang.Package"	19	24	76
10 = {null (0x427e5800)} "UPD"	0	24	30
11 = {null (0x425a35c0)} "AlgorithmParameterGenerator	5	24	108
12 = {null (0x4279a8d8)} "German-American physicist a	11	24	190
13 = {null (0x42845d88)} "/data/data/org.wikipedia.alph	9	24	196
14 = {null (0x425ae0c8)} "TLS_KRB5_EXPORT_WITH_RC4_	0	24	86
15 = {null (0x424f80a8)} "Android KeyStore security prov	0	24	92

Reference Tree	Depth	Shallow Size	Dominating ...
java.lang.String@1115269336 (0x4279a8d8)	11	24	190
description *in* org.wikipedia.feed.model.FeedPageSummary@1113850424 (0x42640238)	10	32	2070
Index 10 *in* java.lang.Object[40]@1112526992 (0x424fd090)	9	160	160
page *in* org.wikipedia.feed.mostread.MostReadItemCard@1115255792 (0x427973f0)	11	16	16

The most common classes using the maximum space in the memory are String, char[] and int[]. The shallow size of these classes varied from 100kb to 600kb approx. The heap snapshot also showed about 200 to 500 instances of duplicated strings.

Optimisations attempted included checking the String pool of Java before creating a new String, converting char[] and int[] to byte[].

Tools Used

Android Studio: To build and run the applications.

Memory Monitor: Taking memory snapshots of the applications.

HPROF Viewer & Analyser: For analysing the HPROF files.

Android Instrumentation Tests: To compare before and after performance of app after optimisation.

Future Plan

Create a class similar to StringBuilder aimed at replacing the use of Strings. This would allow us to nullify the critical data as soon as it is no longer needed.
Also develop an analysis to identify the locations where the Strings can be safely substituted by the new custom class.