1 Empirical Evaluation

We tested Flair's performance in relation to the following questions:
Question 1: How does Flair compare to other inter-application communication analysis tools in respect to the amount of time it takes to analyze a bundle of applications?
Question 2: How does Flair compare to other tools with respect to the number of applications it fails to analyze?
Question 3: How accurately does Flair analyze vulnerabilities within applications when ran against a bundle of benchmark applications?
Question 4: How does Flair's accuracy in analyzing applicaions compare to other inter-application analysis tools?

1.1 Methods of Testing

To test these questions we ran Flair four other inter-application analysis tools: Covert, Didfail, SEALANT, and DIALDroid. We used seven bundles of popular presented in table 1 android applications, each with fifty applications, to answer Question 1 and Questions 2. We then used three bundles of benchmark bundles also presented in table 1 to answer Question 3 and Question 4.

Bundle	Apks
Android Bundle 1	50
Android Bundle 2	50
Android Bundle 3	50
Android Bundle 4	50
Android Bundle 5	50
Android Bundle 6	50
Android Bundle 7	50
Benchmark Bundle	XX
Benchmark Bundle	XX
Benchmark Bundle	XX

Table 1: Bundles used in analysis.

Each of the bundles used to examine Question 1 and Question 2 were run incrimentally. Each test started with only one application from the bundle and the tool was run against it. Then another application was added and the tool was run against the new bundle of two applications. This process was repeated until all the applications from the specified bundle were added. Therefore, each test consisted of 50 consecutive runs of a tool, each consisting of one more application than the last. This was done to reflect how in a real life situation applications are added to a collection that is being checked for vulnerabilities. It is benficial for us to see how each tool performs when new applications are added to a bundle. To more accurate results we ran each tool against each bundle three times and used the averages from those runs for our data.

We gathered data for Question 1 by recording the time it tool each tool to analyze bundles of applications. We recorded the time each tool took for each successive run of a test. Hence, there was fifty recorded times for each test of a tool. The averages for each bundle were then entered into box plot graphs and evaluated.

To answer Question 2 we used error logs provided by eac tool. The tools provided a single error log file for each application that they failed to analyze. We then calculated the failure rate of each tool by:

$$\frac{failed}{num} = FR$$

Where *failed* is the number of error logs given by the tool, *num* is the total number of applications run in the test, here it is 1250, and *FR* is the failure rate for that tool in that test. The failure rates were recorded for each test and then averaged for each tool across all bundles.

Here will go text explaining Question 3 proceedure.

Here will go text explaining Question 4 proceedure.

1.2 Variables

Independent: Our independent variables present in our test are as follows: (1) Tool used to analyze. (2) Specific bundle which the tool analyzed.
Dependent: The dependent variables in our study are as follows: (1) Time it takes for tool to analyze a bundle. (2) Number of applications that a tool fails to analyze. (3) The accuracy of the analysis.

1.3 Testing Enviroment

All testing for Question 1 and Question 2 was done on virtual machines running within Oracle VirtualBox. Each of these machines was running Ubuntu 16.04 and had 8 processing threads clocked at 2.28 Ghz. The memory varied due to the fact that each tool had differing memory requirments. For Covert, Didfail, and Flair the machine was given 3 Gb of memory. SEALANT required more memory to run and was given 6 Gb. DIALDroid, the most memory intensive, was given 14 Gb. Since SEALANT and

DIALDroid had much higher memory requirements than the other tools, we gave them lower limit of the amount of memory that they needed to run. We did this so that each tool would have enough memory to run efficiently, but the higher memory requirement tools did not have a clear advantage. In addtion, there was no swap space avaliable for any of the tools.

1.4 Threats to Validity

Innconsitant Runs: Thought we ran all tests in isolated virtual machines, there still could have been issues realted to a run that is not consistant with the tools true performance. To minimize these affects we ran each tool against each bundle three times and took the averages.

Interal issues: Issues related to the structure and design of the tools themselves. We are not aware of any issues of this kind.

Measurement Issues: Issues related to the methods we used to record and analyze our data. We did not notice any issued related to this.

1.5 Results

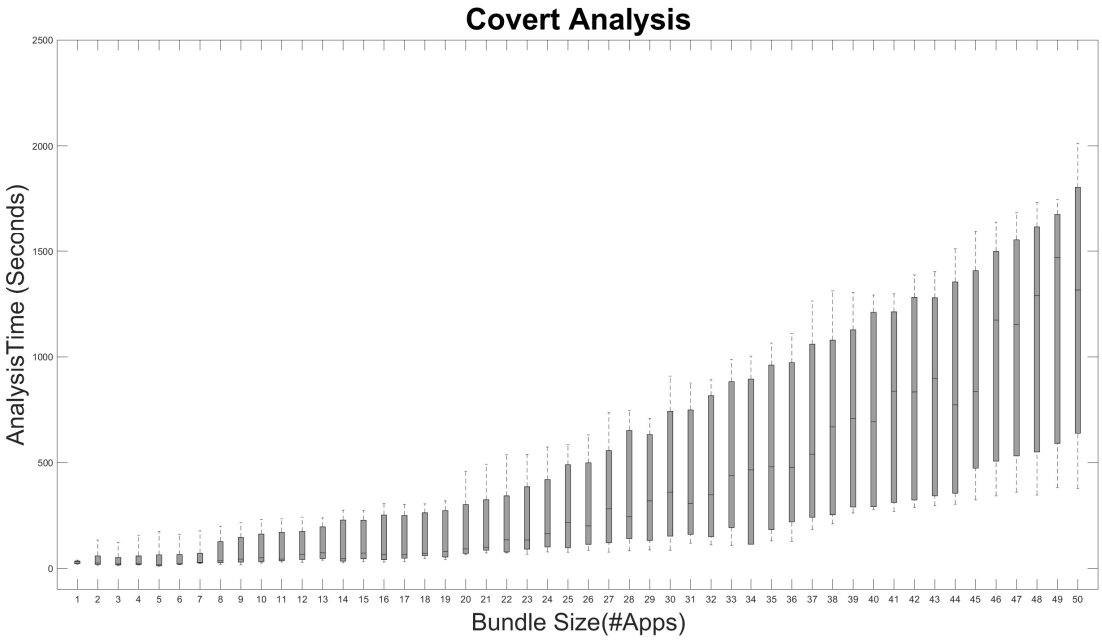

Figure 1: Box plot graph showing Covert analysis times.

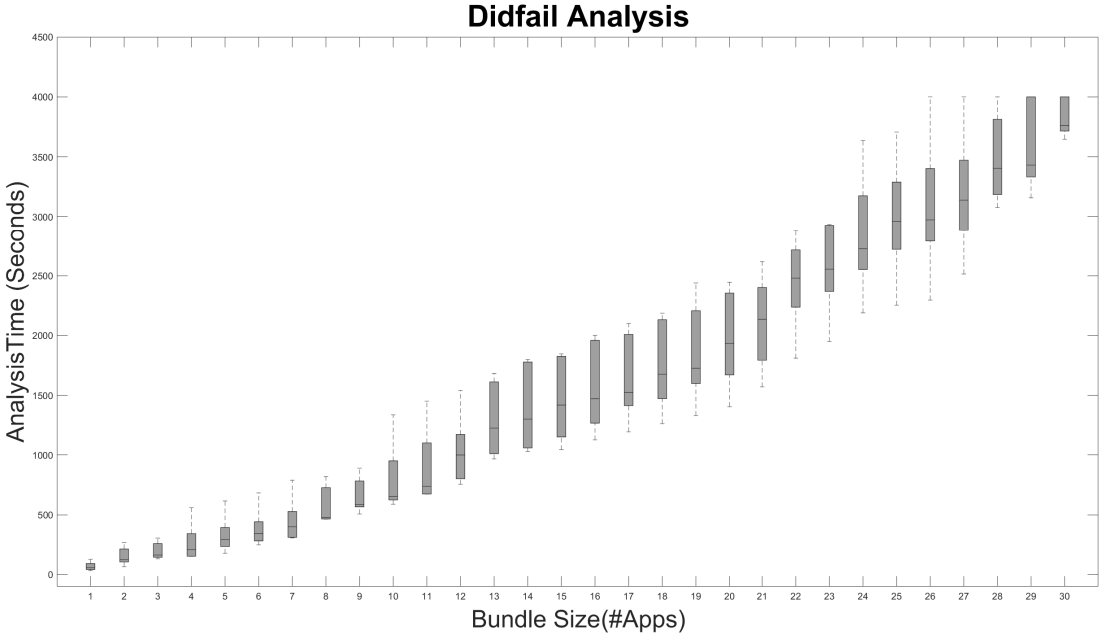

Figure 2: Box plot graph showing Didfail analysis times.

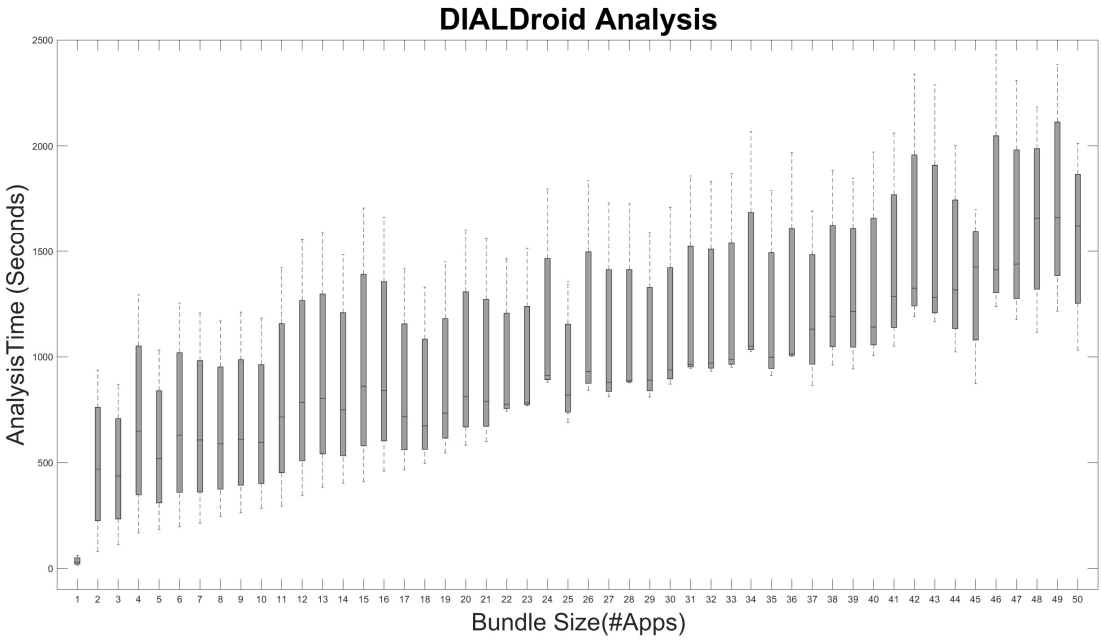

Figure 3: Box plot showing DIALDroid analysis times.

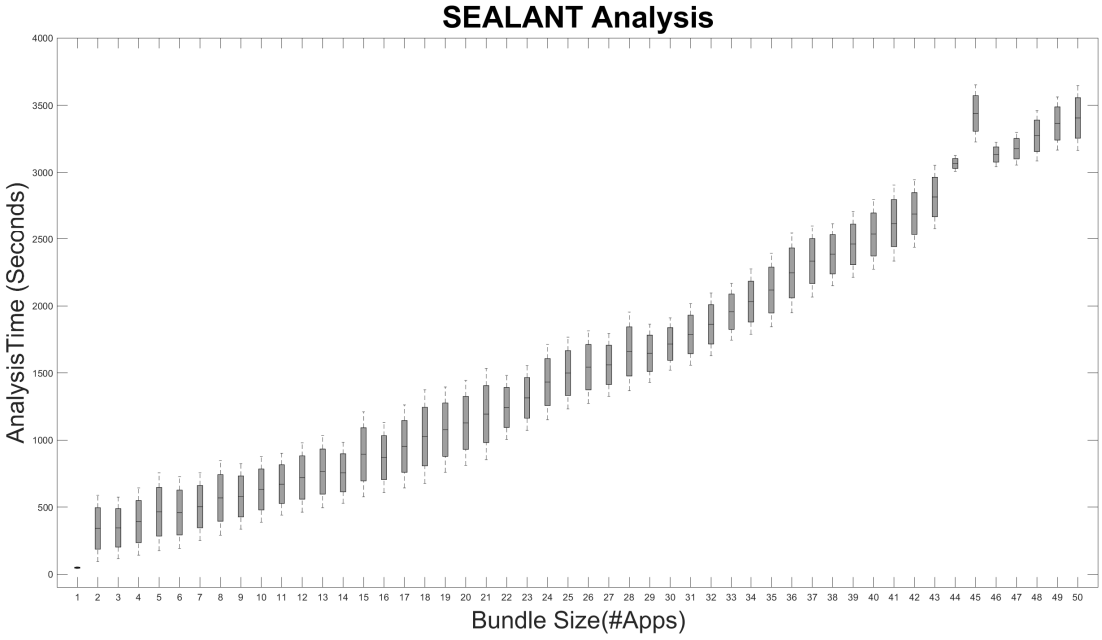

Figure 4: Box plot showing SEALANT analysis times.

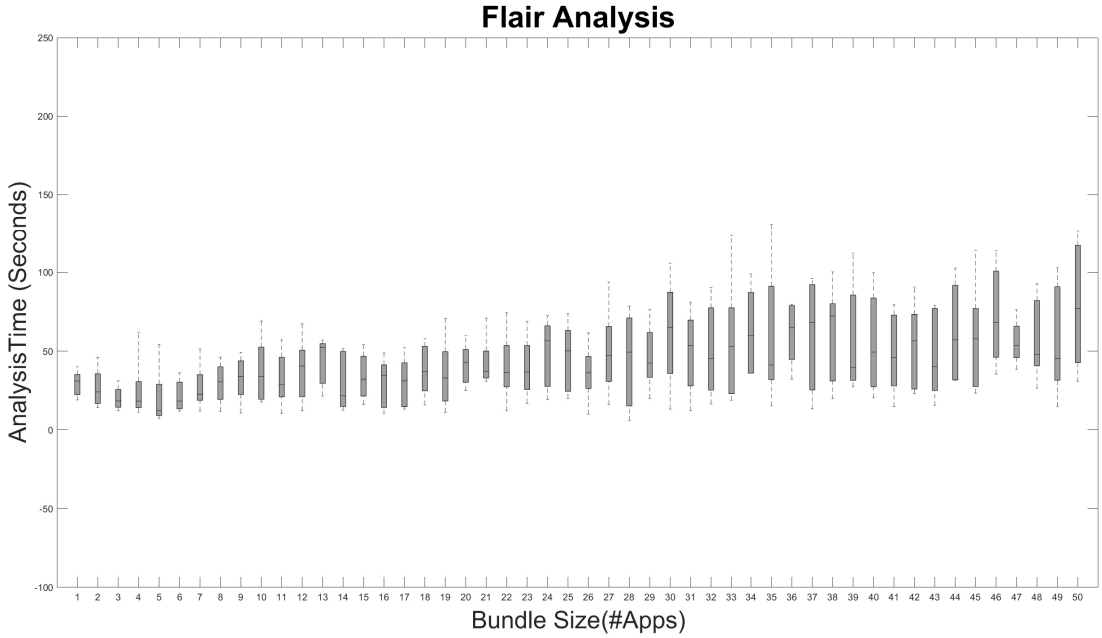

Figure 5: Box plot showing Flair analysis times.

Presumable would enter in results analysis here.

2 Related Work

Covert Jianghao probably explained this already

DidFail Jianghao probably explained this already

DIALDroid DIALDroid is an inter-application ICC security analysis tool with four key operations: ICC Entry / Exit Point Extraction, DataFlow Analysis, Data Aggregation, and ICC Leak Calculation.[1] DIALDroid extracts models from a given application and performs static analysis to find any ICC exit or entry leaks present. Then the information on any leaks is stored in a MySQL relational database to be retrieved when requested by SQL queries. During the initial analysis stage DIALDroid configures the precision based on the length of the analysis. During the first five minutes of analysis DIALDroid uses a high precision algorithm to find ICC entry and exit leaks. After five minutes DIALDroid switches to a low precision algorithm that has a higher change of false negative. If the analysis runs for more than a given timeout, such as 15 minutes, DIALDroid abandons the analysis of that application altogether.

SEALANT Here is some text explain the tool Sealant.

To test each of these tools we used Oracle VirtualBox to create virtual machines running Ubuntu 16.04. For each tool the virtual machine was given the minimum amount of memory that we found it would run on. Covert, DidFail, and Flair all were given 3GB of memory, SEALANT was given 6GB, and DIALDroid, by far the most memory intensive, was given 14GB. Throughout all the tests The CPU was kept consistent at four physical cores and 8 threads. We found this to be the best way of testing each tool. Each tool was given enough memory to run efficiently, but none were given so much as to have a clear advantage over any other.
We ran each tool against seven bundles of applications. Each bundle contained 50 popular android application.

3 Conclusion

Here we will have a conclusion.