1 Related Work

There are many existing tools that closely relate to Flair in their goals. We provide an overview of the most prevalent of those tools below:

Covert Covert is a tool for the compositional analysis of inter-application communication in Android. It allows for the anaylsis of both single applications and bundles of applications. It extracts the source code from each application, extracts communication and security information, and runs formal anaysis on that information to identify any inter-app vulnerabilities.[**?**].

DidFail Didfail is an Android security system built on *FlowDroid* and *Epicc* focused on identifing both inter an intra-compnent vulnerabilities in bundles and individual applications. Didfail first identifies what communications allowed by each individual application. It then analyzes these possible communication to determine any vulnerabilties that may be present in the bundle or application.[**?**]

DIALDroid DIALDroid is an inter-application ICC security analysis system aimed at the analysis of extremely large bundles of applications. It identifies ICC leaks and privelege escalations within large bundles of applications. DIALDroid extracts permisions and possible inter-component communications from the APK files and then analyzes that extracted information. This analysis' percision is varies based on the time the application takes to complete. If the analysis of an application is not complete in a specified time limit, DIALDroid switches to a low precision analysis. If the low precision analysis does not complete within another specified time limit, the analysis is abandoned altogether. After the analysis of all applications in a bundle, the results are stored in a MySQL database to be retrieved when requested.[**?**].

SEALANT SEALANT is an android security system with the goal of both finding and preventing malicious android activity. It is composed of two tools: an analyzer and an interceptor. In this paper we focus only on the analyzer tool as it is closely related to our research. The analyzer works similarly to other ICC analysis tools. First, it extracts ICC paths from APK files. It then identifies vulnerable paths using formal analysis. Lastly, it enters the identified vulnerabilities into a list that will be used by the SEALANT interceptor tool.

2 Empirical Evaluation

We tested Flair's performance in relation to the following questions:

RQ1: 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?
RQ2: How accurately does Flair analyze privilege escalation vulnerabilities within applications when run against a bundle of benchmark applications?
RQ3: How does Flair's accuracy in analyzing privilege escalations compare to other inter-application communication analysis tools?

2.1 Methods of Testing

To test these questions we ran Flair against four other inter-application analysis tools: Covert, Didfail, SEALANT, and DIALDroid. We used seven bundles of popular applications presented in table 1, each with fifty applications, to answer RQ1. Each of these bundles contained different applications or applications in a different order than the other bundles. We then used two bundles of benchmark bundles also presented in table 1 to answer RQ2 and RQ3.

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
DroidBench2.0	3**(Not Final)**
ICC-Bench	9

Table 1: Bundles used in analysis.

Each test we ran to examine RQ1 was incremental in nature. A shell script was utilized to copy each application from a given bundle into a specific input directory. The shell script would then call a tool to be run against that input directory. Each application from a given bundle was copied one by one into the input directory, and the tool was run after each application was copied. This resulted in each test for a given bundle and tool having 50 total runs. The order in which the bundles were copied into the input directories of each tool remained constant allowing the times outputted to be compared directly.

To investigate RQ1 we ran the test described above and recorded the time it took for the tools to analyze the bundles. At every step in the iterative process the time was recorded before the tool was run against the

input directory and after the analysis was completed. These times were then written to a CSV file. Each test outputted its times to a single CSV file and the times from all tests were then combined into a master spreadsheet. These times were translated into graphs and are presented later in this paper. All tests were run three times to increase the validity of our results.
We also noticed that DIALDroid would fail to analyze some applications. Each time it would fail to analyze an application it would output an error log file. After each test we would calculate the failure rate with the following calculation:

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

Where *failed* is the number of error logs given by the tool, *num* is the total number of applications run in the test, 1250, and *FR* is the failure rate in that test. These failure rates were recorded and averaged. We did not notice any of the other tools failing to analyze applications during our tests.

To answer both RQ2 and RQ3 we used the benchmark bundles DroidBench 2.0 ICCTA-Branch, and ICC-Bench to analyze how well each tool is at detecting ICC vulnerabilities. We selected various applications from each bundles intended to represent a wide range of security vulnerabilties present in android. Each tool was run against each specific benchmark applications and it was then recorded whether or not the tool was able to detect the vulnerability present in the benchmark application. These results are presented in ¸pBenchmark table, later in this paper.

2.2 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.

2.3 Testing Environment

All testing for RQ11 and RQ2 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. We intended to run each tool with 3 Gb of memory allocated to their respective VM. In our testing it became apparent that both SEALANT and DIALDroid were unable to run with only 3 Gb of memory, therefore we assigned each tool 14 Gb of memory as this was the minimum amount of memory both tools were able to run at. This is advantageous for these two tool. Nonetheless, the results in ¸fResults section show that they are not as efficient as Flair in terms of scaleability and just as accurate in detecting vulnerabilities.

2.4 Threats to Validity

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

Internal 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.

2.5 Results

2.5.1 RQ1:

We first look to evaluate the analysis time of Flair in relation to Covert, DIALDroid, and SEALANT. The box plot graphs presented in Figure 1 show the spread of analysis times for each tool at various points in our incremental testing. Though, the graph shows increases in bundle size by ten(or five if we uses 5 different graphs), only one app was added at a time in our testing. This data is presented in accordance with Figure 2 which represents the average analysis time of all tests run for all tools. All of the tools show differing trends in average analysis time, spread of their analysis times, and symmetry of the analysis times at a given bundle size.
The data presented in ¸pAverage Lines graph show that Flair maintains analysis times far below any of the other tools. Flair is able to do this since it does not have to re-analyze the entire bundle when another application is added. Unlike the other tools, Flair only analyzes the application that was added to the bundle, greatly decreasing analysis times. The other tools show much higher analysis times, with Didfail by far taking the longest to analyze bundles of applications. In addition, Didfail was unable to analyze bundles larger than 30 applications. Once 30 applications were reached, Didfail failed to produce any results from the analysis. Covert shows a steadily increasing trend of analysis times. Therefore, even though Covert forms the foundations for Flair, Flair is able to greatly reduce times analysis of bundles in an incremental environment. SEALANT

and DIALDroid both maintain average analysis times that are greater than both Flair and Covert, but are far below those of Didfail. Even though they use far more resources than either Flair or Covert, they are unable to compete in scale-ability.
In ¸pBox Plot Figure we present the spread and symmetry of each tools analysis times at various points in the incremental testing. Each tool has a certian amount of variability presented in the IQR of their given box plots. Flair maintains the lowest amount of variability by far. Therefore, Flair's analysis times are able to more easily predicted than any of the other tools. Covert in particular shows an extremely large amount of variability in the IQRs of its box plots, particularly when the bundle sizes increase to being over thirty applications DIALDroid and SEALANT both maintain spreads that are lower than those of Covert or even Didfail, but are still far greater than the spreads for Flair. It is clear that Flair is far more consistant in its analysis of bundles, where the other tools become far less predictable when the bundle sizes increase. The symmetry of these tools also differs. SEALANT has very symmetrical data all the way through the testing. Covert maintains a similar level of symmetry in its analysis times. DIALDroid and Covert, on the other hand, have widely varying symmetries presented in their box plots. These tools tend to have their analysis times for various bundles skewed to the right, though Covert becomes more symmetrical as the bundle size increases. Didfail, like Covert, has data that skews to the right with smaller bundles, but trends towards being more symmetrical as the bundles sizes increase.// To assess the rate of failure we used the methods described in 2.1 to find the percentage of applications in each test which failed. The data we collected shows that while Covert, SEALANT, Didfail, and Flair do not fail to analyze any of the applications, DIALDroid failed a significant *28.54 percent* of applications on average across all runs. This was primarily because DIALDroid would attempt to allocate more than our allotted 14 GB of memory, which was more than twice that allocated to any of the other tools. SEALANT, Covert, Didfail, and Flair all were able to run without any memory issues with far less memory. Therefore, Flair is able to analyze applications as reliably in terms of failure rate as well as SEALANT, Covert, and Didfail, while analyzing applications more reliably than DIALDroid.

In case we use five different graphs:

Didfail Graph: The box plots in ¸pDidfail box plot graph show the spread and symmetry of the time it took Didfail to analyze the seven different bundles of applications. Didfail's analysis tends to skew to the right, and overall is not very symmetrical. Also, Didfail has an increasing IQR spread as the size of the bundles increases.

SEALANT Graph: ¸pSEALANT box plot graph shows the spread of analysis times for SEALANT across the seven bundles. SEALANT maintains a relatively consistent spread and IQR of analysis times as the sizes of bundles increases. SEALANT also have very symmetrical data.

DIALDroid Graph: The box plot graph for DIALDroid is presented in ¸pDIALDroid box plot graph. This data shows that DIALDroid, like Didfail tends to have data that skews right, but DIALDroid's analysis times become slightly more symmetrical as the size of the bundles increase. The IQR, though, grows slightly as the bundle size increases, showing greater variability in analysis times.

Covert Graph: ¸pCovert Box plot graph shows the box plot data for Covert's analysis times. This data shows that Covert's analysis times across our seven bundles was extremely skewed right when the bundles size was small, but then became increasingly symmetrical as the size of the bundle increased. The IQR and spread, though, started out small and grew considerably as the bundle size increased. It is apparent that as the bundles sizes increase Covert's analysis times become far more variable.

Flair Graph: The data in ¸pFlair box plot graph shows the spread and symmetry information for Flair's analysis times. This data shows that Flair is able to maintain extremely low analysis times for increasing bundle sizes. This is due to its iterative process. Flair only has to analysis applications that are added to the bundle, not the whole bundle. This allows Flair to maintain analysis times far below the other tool. Flair also maintains lower IQR ranges, showing that Flair is more consistent overall.

2.5.2 RQ2:

To test Flair's accuracy in identifying ICC vulnerabilities in Android we used a selection of test applications from two benchmark bundles: DroidBench 2.0 ICCTA Branch and ICC-Bench. The results of these tests are shown in ¸pBenchmarks Table. These results that Flair is able to detect nearly all ICC vulnerabilties within these benchmark bundles. Flair is able to maintain a very high level of accuracy in detecting ICC vulnerabilities, while still greatly decreasing analysis time and variability.

2.5.3 RQ3:

We also ran the same benchmark applications against Didfail, SEALANT,DIALDroid, and Covert so that we could compare how Flair related to other tools. While Covert was able to maintain the same levels of accuracy as Flair, our results showed that the other tools were not as accurate at detecting ICC vulnerabilities. Though, DIALDroid was the able to detect the two vulnerabilities that Flair was unable to detect, DynRegisteredReciever1 and 2, it was unable to detect many of the other vulnerabilities that we tested for. In particular, DIALDroid struggled to detect all vulnerabilties when there were multiple present in a single application. On both ICC bindService4 and ICC startActivityForResult4 it detected on of the present vulnerabilties, but failed to address the other. SEALANT failed to detect many of the present in our tests, as did Didfail. Didfail also reported multiple false positives, whereas we did not notice any of the other tools reporting false positives. Overall, Flair and Covert present accurate and reliable analysis of benchmark applications, while Didfail, SEALANT, and DIALDroid struggle to report all of the vulnerabilties present in Android applications today.

3 Conclusion

Here we will have a conclusion.

	Test Case	Covert	DIALDroid	Didfail	Flair	SEALANT
DroidBench2	ICC_bindService1	☑	☑	☒☐	☑	☑
	ICC_bindService2	☑	☐	☐	☑	☑
	ICC_bindService3	☑	☐	☐	☑	☑
	ICC_bindService4	(☑2)	☑☐	☒(☐2)	(☑2)	(☑2)
	ICC_sendBroadcast1	☑	☑	☑	☑	☑
	ICC_startActivity1	☑	☑	☐	☑	☐
	ICC_startActivity2	☑	☑	☐	☑	☑
	ICC_startActivity3	☑	☑	☐	☑	☑
	ICC_startActivity4			☒		
	ICC_startActivity5			(☒2)		
	ICC_startActivityForResult1	☑	☑	☐	☑	☑
	ICC_startActivityForResult2	☑	☑	☐	☑	☑
	ICC_startActivityForResult3	☑	☐	☐	☑	☑
	ICC_startActivityForResult4	(☑2)	☑☐	(☐2)	(☑2)	☑☐
	ICC_startService1	☑	☑	☒☑	☑	☐
	ICC_startService2	☑	☑	☒☑	☑	☐
	ICC_delete1	☑	☐	☐	☑	☐
	ICC_insert1	☑	☐	☐	☑	☐
	ICC_query1	☑	☐	☐	☑	☐
	ICC_update1	☑	☐	☐	☑	☐
	IAC_startActivity1	☑	☐	☒☑	☑	☐
	IAC_startService1	☑	☐	☑	☑	☐
	IAC_sendBroadcast1	☑	☐	☑	☑	☐
ICC-Bench	Explicit_Src_Sink	☑	☑	☐	☑	☐
	Implicit_Action	☑	☑	☑	☑	☐
	Implicit_Category	☑	☑	☑	☑	☐
	Implicit_Data1	☑	☑	☑	☑	☐
	Implicit_Data2	☑	☑	☑	☑	☐
	Implicit_Mix1	☑	☑	☑	☑	☑
	Implicit_Mix2	☑	☑	☑	☑	☐
	DynRegisteredReceiver1	☐	☑	☐	☐	☐
	DynRegisteredReceiver2	☐	☑	☐	☐	☐
Precision		100%	XX%	55%	100%	XX%
Recall		97%	XX%	37%	97%	XX%
F-measure		98%	XX%	44%	98%	XX%