Artifacts Scorecard

Detecting Sensitive Data Disclosure via Bi-directional Text Correlation Analysis

Jianjun Huang Xiangyu Zhang
Department of Computer Science
Purdue University, USA
{huang427, xyzhang}@cs.purdue.edu

Lin Tan
Electrical and Computer Engineering
University of Waterloo, Canada
lintan@uwaterloo.ca

1. INSIGHTFULNESS

Sensitive data disclosure is an important threat to user privacy on mobile platforms [6, 2]. Existing works leverage taint analysis to discover potential sensitive data disclosures [5, 3]. However, these approaches require defining sensitive data sources, which are usually APIs with a sensitive return value. If any *forward* data flow is observed between such sources and any sinks, disclosure defects are reported. Later, researchers realized that even some generic APIs may return sensitive values, depending on the context. That means, under certain conditions, sensitive data disclosure occurs for the data obtained via some generic APIs. SUPOR [4] and UIPicker [7] aimed to identify which user inputs on the user interfaces may be sensitive and track if the sensitive user inputs are disclosed via forward data flow analysis.

The above solutions have limitations. Generic APIs can produce sensitive data but not all of them are related to UI (*e.g.,* loading data from files or receiving data from network). In other words, we may not be able to determine whether the return value of a generic API is sensitive or not. We cannot simply treat all generic APIs as the taint sources as that will lead to a large number of false warnings. In addition, forward data flow analysis is insufficient. In many cases, a piece of data may not be recognized as sensitive when it is emitted through a sink. But later it is associated with some program artifacts that indicate the sensitiveness of the data. Note that there may not exist any forward data flow from the sensitiveness revelation point to the sink point.

BIDTEXT detects sensitive data disclosures by examining the text labels correlated with variables. The text labels can be from either the code or the UI. After extracting these labels, BIDTEXT leverages a novel type system like mechanism to propagate these labels through both *backward* and *forward* data flow. Then sensitive data disclosures are reported when a parameter at a sink point is typed with a sensitive textual label.

2. USEFULNESS

BIDTEXT is highly usefulness in practice. It is implemented as a static analysis tool for Android apps, making it capable of scanning a large scale of apps. App developers can use it to discover potentially sensitive data disclosures in their apps that may not be able to be detected by existing solutions. App markets providers can also apply it to the submitted apps on the market, helping the developers improve their apps.

The artifact package provides the source code of BIDTEXT, as well as the executables and a portion of the Android apps used in the evaluation. Therefore, the users can either build on the source code or simply apply BIDTEXT on Android apps.

3. USABILITY

Easy to understand: The BIDTEXT paper provides detailed description of the design and evaluation. In the artifact package, we describe how to use the tool and give an example to address how to understand the results.

Accompanied by tutorial notes: In addition to the PDF that accompanies this scorecard, we have also provided a tutorial along with the executables to guide the users to use the tool.

Easy to download, install, or execute: We provide a Ubuntu-based virtual machine image which contains the source code, the executables and a portion of the test apps. It can be downloaded at: https://github.com/hjjandy/FSE16-BidText-Artifacts-VM. We also provide a non-VM repository containing the executables and the test apps. This repo contains scripts that can run the executables on both Windows and Linux. The users are only required to have Java 7 or 8 with AMD64 architecture installed. This repo can be found at: https://github.com/hjjandy/FSE16-BidText-Artifacts. The source code of BIDTEXT can be found here: https://bitbucket.org/hjjandy/toydroid.bidtext.

Available in a virtual machine image: Yes.

Available online: Yes.

Supported by configuration management tools to permit easy updates: The source code and executables can be updated with Git and the source code comes with the Gradle [1] compilation support.

4. REFERENCES

[1] Gradle build toos: Modern source build automation. http://gradle.org/.

[2] EGELE, M., KRUEGEL, C., KIRDA, E., AND VIGNA, G. PiOS: Detecting privacy leaks in iOS applications. In *NDSS* (2011).

[3] GORDON, M. I., KIM, D., PERKINS, J., GILHAMY, L., NGUYENZ, N., AND RINARD, M. Information-flow analysis of Android applications in DroidSafe. In *NDSS* (2015).

[4] HUANG, J., LI, Z., XIAO, X., WU, Z., LU, K., ZHANG, X., AND JIANG, G. Supor: Precise and scalable sensitive user input detection for android apps. In *USENIX Security* (2015).

[5] HUANG, W., DONG, Y., AND MILANOVA, A. Type-based taint analysis for Java Web applications. In *FASE* (2014).

[6] LU, K., LI, Z., KEMERLIS, V., WU, Z., LU, L., ZHENG, C., QIAN, Z., LEE, W., AND JIANG, G. Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting. In *NDSS* (2015).

[7] NAN, Y., YANG, M., YANG, Z., ZHOU, S., GU, G., AND WANG, X. UIPicker: User-input privacy identification in mobile applications. In *USENIX Security* (2015).

Artifact: Detecting Sensitive Data Disclosure via Bi-directional Text Correlation Analysis

Jianjun Huang Xiangyu Zhang
Department of Computer Science
Purdue University, USA
{huang427, xyzhang}@cs.purdue.edu

Lin Tan
Electrical and Computer Engineering
University of Waterloo, Canada
lintan@uwaterloo.ca

This artifact provides an executable environment of BIDTEXT from our FSE'16 paper *Detecting Sensitive Data Disclosure via Bi-directional Text Correlation Analysis*. The goal is to reproduce the results shown in the paper, as well as to let the users be able to apply BIDTEXT on Android apps of their interest.

1. WHERE ARE THE ARTIFACTS?

The source code is publicly available at https://bitbucket.org/hjjandy/toydroid.bidtext. Gradle [1] build is supported. And JDK 7 or newer versions is required to compile the source code.

A virtual machine (VM) image with Ubuntu 14.04 can be found here: https://github.com/hjjandy/FSE16-BidText-Artifacts-VM. It is created by Oracle VM VirtualBox [2]. The VM contains the source code, executables and a small set of test apps.

The users can also download the executables and the test apps at https://github.com/hjjandy/FSE16-BidText-Artifacts. It provides support to execute the tool on both Windows and Linux if Java 7 or newer versions with AMD64 architecture has been installed.

2. WHAT ARE CONTAINED?

Take the VM image as an example for below sections. In HOME directory, two files and three folders are related to the artifacts.

File "paper.pdf" is a non-camera ready version of our FSE paper, used as a reference of the artifacts. File "FSE16-Artifacts-Eval-README" is a detailed description of the artifacts, including how to run the tool and how to understand the results. It also contains some issues that need special attention to run the tool well.

Folder "BidText-Source" contains a full copy of the source code. The users can update it via "git pull". "BidText-TestApps" contains a portion of Android apps used in the paper. Inside it, there are four special cases, corresponding to the four examples presented in the paper. They are the motivation example (section 2), the example for bi-directional propagation on PHI statement (section 3.2.2), the example of two different types of sources flowing to the same sink (section 5.3.1) and the case study in section 5.3.2. Besides, subfolder "Others" contains the 100 apps used to measure the accuracy of BIDTEXT, whose results are presented in Table 1. "BidText-Bin" contains the necessary libraries, configurations and scripts to execute BIDTEXT. Note that it is infeasible to host all the apps used in the paper due to the sheer volume of the apps.

3. HOW TO RUN THE TOOL?

In order to apply the tool on an Android app, the user can run in a terminal under the folder "BidText-Bin": `./RUN Path_to_APK`. The user can also use the four start scripts to perform analysis on the four corresponding cases. For example, `./Motivation` is equal to `./RUN $HOME/BidText-TestApps/Motivation/com.buycott.android-22.apk`. If the users want to test the 100 apps, just execute `./Eval-100`.

Since BIDTEXT requires a lot of memory to perform analysis, the user is required to allocate enough memory for the VM when creating a VM in VirtualBox. It should be at least 5GB because the default setting of JVM heap size for running BIDTEXT in the VM is 4GB, which is not enough to evaluate all the 100 apps. We suggest 12GB to evaluate the 100 apps. The users can modify "bidtext.prop" to set a larger JVM heap size for BIDTEXT.

The non-VM artifacts also contain start scripts (batch files) to allow easy execution of BIDTEXT on Windows. The commands and the settings are the same as in the VM.

4. HOW TO UNDERSTAND THE RESULTS?

After analyzing an APK file, BIDTEXT generates the results (if reported any) in folder "*APK_name*.bidtext" which is aside the APK file. Each reported sink point has an individual result file named "*idx.Sink_Type*.txt", *e.g.,* 384.LOG.txt. The result file contains the sink API, the enclosing method of the sink. For each identified sensitive textual label, surrounded by "********", the propagation path is listed. Each path element is the `String` representation of a WALA [3] `Statement`, an IR used during the analysis. Use a piece of the results for the motivation as an example. The example here is simply modified for simplicity.

```
1 ********username********
2 NORMAL_RET_CALLER:Node: <CampaignActivity$20$1,
      run()V> 66 = invokevirtual <JSONObject,
      getString(String)String> 18,64
3 NORMAL_RET_CALLER:Node: <CampaignActivity$20$1,
      run()V> 18 = invokevirtual <JSONArray,
      getJSONObject(int)JSONObject> 5,172
```

NORMAL_RET_CALLER is the type of the `Statement` and "Node" indicates the location of the statement. The numbers are the variables values, distinct for any SSA variables. For example, "66" can be treated as a variable `v66`, which can only be defined once in the enclosing method. For more details about the representation, please refer to WALA document. The sensitive textual label "username" propagates from line 1, which is a method invocation. Through inspecting the app's code, we know that the label is a constant string stored in `v64` while the JSON object is associated with `v18`. In the second statement, we see that `v18` is the return value of a method call. That means, the two statements constitute a *backward* propagation. Please refer to Figure 2 in our paper to get a better understanding of the motivation example.

[1] http://gradle.org/
[2] https://www.virtualbox.org/
[3] http://wala.sourceforge.net