SpanDex: Secure Password Tracking for Android

Landon P. Cox  
Peter Gilbert  
Geoffrey Lawler  
Valentin Pistol  
Ali Razeen  
Bi Wu  
Sai Cheemalapati  
Duke University  
Duke University  
Duke University  
Duke University  

Abstract
This paper presents SpanDex, a set of extensions to Android’s Dalvik virtual machine that ensures apps do not leak users’ passwords. The primary technical challenge addressed by SpanDex is precise, sound, and efficient handling of implicit information flows (i.e., information transfers induced by a program’s control flow). SpanDex handles implicit flows by borrowing techniques from symbolic execution to precisely quantify the amount of information a process’ control flow reveals about a secret. To apply these techniques at runtime without sacrificing performance, SpanDex runs untrusted code in a data-flow sensitive sandbox, which limits the mix of operations that an app can perform on sensitive data. Experiments with a SpanDex prototype using 50 popular Android apps and an analysis of a large list of leaked passwords predicts that for 90% of users, an attacker would need 80 or more login attempts to guess their password. Today the same attacker would need only one attempt for all users.

1 Introduction

Today’s consumer mobile platforms such as Android and iOS manage large ecosystems of untrusted third-party applications called “apps.” Apps are often integrated with remote services such as Facebook and Twitter, and it is common for an app to request one or more passwords upon installation. Given the critical and ubiquitous role that passwords play in linking mobile apps to cloud-based platforms, it is paramount that mobile operating systems prevent apps from leaking users’ passwords. Unfortunately, users have no insight into how their passwords are used, even as credential-stealing mobile apps grow in number and sophistication [13, 14, 27].

Taint tracking is an obvious starting point for securing passwords [12]. Under taint tracking, a monitor maintains a label for each storage object. As a process executes, the monitor dynamically updates objects’ labels to indicate which parts of the system state hold secret information. Taint tracking has been extensively studied for many decades and has practical appeal because it can be transparently implemented below existing interfaces [12, 22, 6, 15].

Most taint-tracking monitors handle only explicit flows, which directly transfer secret information from an operation’s source operands to its destination operands. However, programs also contain implicit flows, which transfer secret information to objects via a program’s control flow. Implicit flows are a long-standing problem [9] that, if left untracked, can dangerously underestimate which objects contain secret information. On the other hand, existing techniques for securely tracking implicit flows are prone to significantly overstating which objects contain secret information.

Consider the following example. Sentence contains explicit flows from a to x and from b to y as well as implicit flows from s to x and s to y. A secure monitor must account for the information that flows from s to x and s to y, regardless of which branch the program takes: y’s value will depend on s even when s is non-zero, and x’s value will depend on s even when s is zero.

Existing approaches to tracking implicit flows apply static analysis to all untaken execution paths within the scope of a tainted conditional branch. The goal of this analysis is to identify all objects whose values are influenced by the condition. Strong security requires such analysis to be applied conservatively, which can lead to prohibitively high false-positive rates due to variable aliasing and context sensitivity [11, 15].

In this paper, we describe a set of extensions to Android’s Dalvik virtual machine (VM) called SpanDex that provides strong security guarantees for third-party apps’ handling of passwords. The key to our approach is focusing on the common access patterns and semantics of the data type we are trying to protect (i.e., passwords).

SpanDex handles implicit flows by borrowing tech-
niques from symbolic execution to precisely quantify the amount of information a process' control flow reveals about a secret. Underlying this approach is the observation that as long as implicit flows transfer a safe amount of information about a secret, the monitor need not worry about where this information is stored. For example, mobile apps commonly branch on a user's password to check that it contains a valid mix of characters. As long as the implicit flows caused by these operations reveal only that the password is well formatted, the monitor does not need to update any object labels to indicate which variables' values depend on this information.

To quantify implicit flows at runtime without sacrificing performance, SpanDex executes untrusted code in a data-flow defined sandbox. The key property of the sandbox is that it uses data-flow information to restrict how untrusted code operates on secret data. In particular, SpanDex is the first system to use constraint-satisfaction problems (CSPs) at runtime to naturally prevent programs from certain classes of behavior. For example, SpanDex does not allow untrusted code to encrypt secret data using its own cryptographic implementations. Instead, SpanDex's sandbox forces apps that require cryptography to call into a trusted library.

SpanDex does not "solve" the general problem of implicit flows. If the amount of secret information revealed through a process' control flow exceeds a safe threshold, then a monitor must either fall back on conservative static analysis for updating individual labels or simply assume that all subsequent process outputs reveal confidential information. However, we believe that the techniques underlying SpanDex may be applicable to important data types besides passwords, including credit card numbers and social security numbers. Experiments with a prototype implementation demonstrate that SpanDex is a practical approach to securing passwords. Our experiments show that SpanDex generates far fewer false alarms than the current state of the art, protects users passwords from a strong attacker, and is efficient.

This paper makes the following contributions:

- SpanDex is the first runtime to securely track password data on unmodified apps at runtime without overpainting or poor performance.
- SpanDex is the first runtime to use online CSP-solving to force untrusted code to invoke trusted libraries when performing certain classes of computation on secret data.
- Experiments with a SpanDex prototype show that it imposes negligible performance overhead, and a study of 50 popular, non-malicious unmodified Android apps found that all but eight executed normally.

The rest of this paper is organized as follows: Section 2 describes background information and our motivation, Section 3 provides an overview of SpanDex's design, Section 4 describes SpanDex's design in detail, Section 5 describes our SpanDex prototype, Section 6 describes our evaluation, and Section 7 provides our conclusions.

## 2 Background and motivation

Under dynamic information-flow tracking (i.e., taint tracking), a monitor maintains a label for each storage object capable of holding secret information. A label indicates what kind of secret information its associated object contains. Labels are typically represented as an array of one-bit tags. Each tag is associated with a different source of secret data. A tag is set if its object's value depends on data from the tag's associated source. Operations change objects' state by transferring information from one set of objects to another. Monitors propagate tags by interposing on operations that could transfer secret information, and then updating objects' labels to reflect any data dependencies caused by an operation. We say that information derived from a secret is safe if it reveals so little about the original secret that releasing the information poses no threat. However, if information is unsafe, then it should only be released to a trusted entity.

### 2.1 Related work: soundness, precision, and efficiency

The three most important considerations for taint tracking are soundness, precision, and efficiency. Tracking is sound if it can identify all process outputs that contain an unsafe amount of secret information. Soundness is necessary for security guarantees, such as preventing unauthorized accesses of secret information. Tracking is precise if it can identify how much secret information a process output contains. Precision can be tuned along two dimensions: better storage precision associates labels with finer-grained objects, and better tag precision associates finer-grained data sources with each tag.

Imprecise tracking leads to overpainting, in which safe outputs are treated as if they are unsafe. A common way to compensate for imprecise tracking is to require users or developers to declassify tainted outputs by explicitly clearing objects' tags.

Tracking is efficient if propagating tags slows operations by a reasonable amount. The relationship between efficiency and precision is straightforward: increasing storage precision causes a monitor to propagate tags more frequently because it must interpose on lower-level operations; increasing tag precision causes a monitor to do more work each time it propagates tags. Finding a suitable balance of soundness, precision, and efficiency...