Bouncer’s secure counting system

Bouncer Technologies
June 2020

1 Secure counting abstraction

Bouncer enables app builders to count events that it associates with hardware devices. This section describes our design of the secure counting abstraction by motivating why app builders would want to count and some of the limitations of current approaches, in addition to describing the basics of how counting works.

Bouncer’s secure counting is a counter that is based on security hardware found on modern smartphones. Our secure counter is a novel abstraction where Bouncer tracks events, like cards added, on a per-device basis. These events help app builders detect attacks and track devices that attackers have used previously. However, as a first-class design consideration our secure counters maintain end-user privacy and remain intact even across factory resets of the device.

1.1 Why counting?

Before we describe how we count, we explain *why* one would want to count events. One key observation about modern attackers is that they tend to use real hardware devices to carry out their attacks. Hardware-based mechanisms from Apple [3] and Google [4] provide app builders with solid mechanisms for ensuring that a request comes from a legitimate iOS or Android device. Some attackers even carry out this hardware-based technique at scale [6] due to these limitations on their attacks.

Given that app builders can push attackers into using legitimate hardware devices, attackers try to repeat the same attacks using the same physical and relatively expensive hardware. App builders, knowing this, will try to count events associated with a device that indicate the existence of an attack. For example, credit card fraudsters will add many cards to accounts using the same device and will login to several accounts using the same device. If app builders can count these events on a per-device basis, they can detect the attacks, as we show in Section 2.

Unfortunately, app builders have a difficult tradeoff that

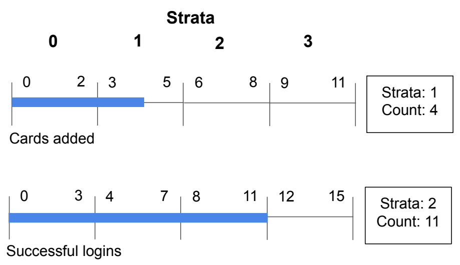

Figure 1: Counts and their associated strata. This figure shows counts for cards added and successful logins and their corresponding strata.

they need to make to be able to detect these events. They can either use privacy-friendly device IDs, which attackers can reset by uninstalling the app or performing a factory reset of the device. Or app builders can use persistent device IDs, which violate the privacy of their end users and Apple’s App Store policy prohibits [2, 5]. Existing industry solutions to counting that we have first-hand experience with suffer from these problems. Our secure counter is novel because it respects end-user privacy while still empowering apps to maintain counts even across resets.

1.2 Secure counting basics

At the heart of our secure counting abstraction is Apple’s DeviceCheck abstraction [3]. DeviceCheck uses hardware-backed tokens stored on a device, which our server uses to query two bits per device from Apple’s servers. DeviceCheck is supported on all devices running iOS 11.0 and above, which accounts for 98.3% of all iOS devices. However, two bits are not enough for app builders who want to count directly arbitrary events.

Instead of using DeviceCheck’s two bits to encode values directly, we use them to define a range of possible counts.

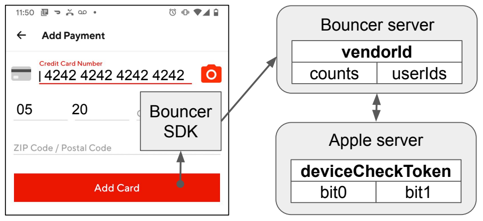

Figure 2: The architecture for the secure counting abstraction. This figure shows how Bouncer updates counts after an app adds a card. The app calls into the Bouncer SDK, which calls the Bouncer server, where Bouncer maintains a database of counts. The Bouncer server manages the DeviceCheck bits by accessing Apple's servers on behalf of the app.

Figure 1 shows a device where we are tracking cards added and successful logins. For this app, the app builder expects a maximum of 11 cards added and 15 successful logins, which Bouncer divides into four sections, or strata. We divide the counts by four so that we can represent each of the four strata using Apple's two hardware bits. In our example, this device has a count of four cards added and eleven successful logins, which map to strata one and two respectively.

The software counts and hardware strata complement each other where we use software counts in the common case where the user maintains the same *vendorId* (Apple's privacy-friendly deviceId abstraction [1]), but hardware strata to recover lost count values when we see a device reset. For attackers that reset their device, our counting abstraction provides monotonically increasing count values, but for legitimate users who reset their device, by dividing the counts up into four strata we limit the amount that our counts will increase on a device reset to avoid falsely flagging good users.

Figure 2 shows our overall architecture for how app builders use the secure counting abstraction and how Bouncer keeps track of counts. From a high level, the app invokes an `increment` function in our SDK to increment the count for cards added, successful logins, or any events they want to track. The app includes an anonymous, but consistent, userId along with the request. Our SDK then retrieves a fresh DeviceCheck token from the device and the vendorId and passes these along with the userId to the Bouncer server. The Bouncer server maintains a device database indexed via the vendorId to keep track of counts and userIds for this device. The Bouncer server also accesses Apple's servers, on behalf of the app, to query and set DeviceCheck bits. Apps need to register a DeviceCheck private key with Bouncer to enable us to access Apple's servers on their behalf. Subsequently, the app can query counts from Bouncer's server.

1.3 Counting and inconsistencies

Figure 3 shows an example of how the counting system state advances as three different events occur while counting cards added and logins. The system starts with two cards added and one login counts. We define the strata for each of these counts by dividing the maximum expected number of events by four, and each range represents a stratum. If the system increments the cards added count, it causes the count to cross a stratum as it moves the count from two to three, putting that count into stratum 1. The system defines the overall software stratum for a device as the maximum of all counter strata, so Bouncer advances the software stratum to 1 and the DeviceCheck stratum to 1 as well to match the software stratum. At this point, the cards added, and login counts are in different stratum, which is acceptable as long the app continues to use the same vendorId.

If the user resets their vendorId, then subsequent requests will appear to come from a new device with all counts set to 0 and the software stratum set to an initial state ("-" in the figure). However, the DeviceCheck stratum is 1, causing an inconsistency. As a result of this inconsistency, Bouncer sets *all* counts to the maximum value for their strata, which in this case is five for the cards added count and seven for the login count when they are in stratum 1. *By setting the counts to the maximum value within their strata as defined by the DeviceCheck stratum, we guarantee that all counts are equal to or greater than what they were before the inconsistency, thus maintaining monotonically increasing counts even after a vendorId reset.*

Our rules for counting are:

- The strata for a count = floor(count * 4 / max_count).
- The software strata = max(strata for all counts).
- If DeviceCheck strata $>$ software strata, set all counts to the maximum count according to the DeviceCheck strata.
- If DeviceCheck strata $<$ software strata, set the DeviceCheck strata = software strata.

Our system also handles counts where the maximum count is less than four and we handle the case where the attacker moves vendorIds between attacker-controlled devices, but we omit the details from this paper.

Initialization edge case We handle the case where a user resets their device before advancing strata with the help of an uninitialized state. For a fresh device, both hardware and software strata are set to this state before the app is used for the first time, after which both advance to stratum 0. For a user who resets their device at this point, the software state goes back to being uninitialized, while the hardware state is still in stratum 0. When we see such a configuration, we push the user to the maximum count of stratum 0 in software to

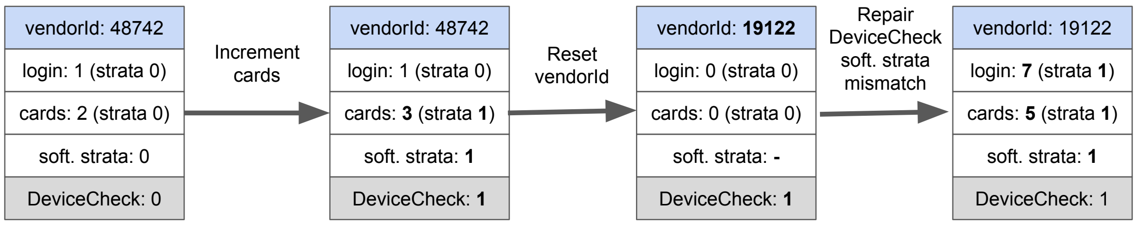

Figure 3: Example of Bouncer's secure counting system. In this figure, we show the counting system in four different states with three transitions between them. In this example, Bouncer is counting cards added and logins, and tracking these on a per vendorId basis.

match the hardware stratum. When the user adds their next card, both hardware and software strata will go to stratum 1, thereby ensuring monotonically increasing counts.

2 Can Bouncer's secure counting catch real attacks?

To evaluate secure counting, we report on data from an app that shipped our SDK in their production system. They ran the system for two weeks in November 2019 in production but did *not* use the results actively to stop attacks, but rather passively recorded information. The company does have other rules that they use to block transactions, so although our count is passive, they do actively block transactions from suspicious users. Having a passive count is advantageous because we can inspect the data more deeply before attackers attempt to evade Bouncer.

In their setup they count cards that users add to an account for each device and set the maximum count to six per month. We also record all the unique userIds that we see for a device but record that in a database and do *not* use secure counting to track that yet. We took a random sample of ten users who hit this maximum count and report the results.

The first question we wanted to answer was whether attackers reset their device. We track device resets by observing an inconsistency between the software count and hardware stratum and record a timestamp for when the reset happens. In our sample, 7/10 attackers did reset their device, presumably as a countermeasure to the other security rules that the company used. For these reset devices, the company would have been unable to count any per-device events, including cards added, without using Bouncer. Bouncer was able to recover the cards added count after resets and maintain monotonically increasing counts.

The second question we wanted to answer was whether counting cards added to a device would be useful for stopping fraud. To answer this question, we pulled the userIds from our database for all users who added a card to one of the devices that hit the six-card limit and inspected all their transactions manually.

Fraudsters used all ten devices for attacks that the company would like to prevent, and the attacks fell into three categories. First, 4/10 devices took part in traditional stolen card fraud where the users of that device added cards from a broad range of zip codes (e.g., across multiple states), indicating that the cards were coming from a list of stolen credentials. Second, 3/10 devices took part in a credit-card specific version of credential stuffing, where they added 12, 42, and 100 unique cards to a device, presumably to check if the card data they had was valid. Interestingly, the devices that they used to check cards did *not* have any transactions on them. Third, 3/10 devices took part in a scheme where they abuse the pre-authorization system.

For the ten devices that we inspected manually, we had no false positives – fraudsters used all the devices we inspected for attacks. The attacks fell into three different categories, but they were all attacks.

Although we do not know the recall of the Bouncer card added count, which would be a measure of how much of the total fraud problem does this signal catch, we can confirm that the 7/10 devices used for stolen card fraud and failed transaction fraud had charges on them that the company's other systems had missed. As such, the company plans to start using the Bouncer card added count in production to block suspicious transactions.

Finally, of the ten devices that we inspected, one device had three unique users, and another had six unique users all who added cards on the same device, suggesting that tracking unique logins per device could be another useful signal.

3 Conclusion

This paper introduced Bouncer's secure counting system, a new system for stopping fraud. Our counting system tracks evens on a per-device basis using secure hardware, and it is robust against device factory resets while still respecting end-user privacy.

References

[1] Vendorid documentation, 2019. https://developer.apple.com/documentation/uikit/uidevice/1620059-identifierforvendor.

[2] Serge Egelman, Feburary 2019. https://blog.appcensus.io/2019/02/14/ad-ids-behaving-badly/.

[3] Apple Inc. Devicecheck documentation, 2019. https://developer.apple.com/documentation/devicecheck.

[4] Google Inc. Key and id attestation, 2019. https://source.android.com/security/keystore/attestation.

[5] Mike Isaac. Uber's c.e.o. plays with fire, April 2017. https://www.nytimes.com/2017/04/23/technology/travis-kalanick-pushes-uber-and-himself-to-the-precipice.html.

[6] Mashable. Say goodbye to those fake likes: Huge click farm discovered in thailand, 2017. https://mashable.com/2017/06/13/thailand-click-farm-caught/#ZGoNx7UDDOqj.