Know Your Opponent – an Inference Attack Against iOS Game Center

written by Karl Fosaaen | February 11, 2013

Lately I've been looking at iOS. After looking into the Passbook application, I started poking around with the iOS Game Center application. The iOS Game Center allows iOS users to connect with friends, play games, and compare scores for their games. Think of it as Xbox Live for iOS.

Each Game Center user has an alias (or nickname, handle, etc.), a first and last name, and an email address tied to their account (also tied to your Apple ID). A user's alias is publicly accessible by all Game Center users. The user's first and last names are provided if they are shown in the "Friend Recommendations" feature or if they share a mutual friend with another user. If an email address is tied to an account, a SHA1 hash of the email address might also be accessible. Sometimes there's more than one email address tied to an account, so multiple hashes will be returned when the account information is queried. Finally, each user has a playerID in the format of G:145811274 (my ID). This is the unique identifier used by Game Center to identify an account. I was only able to see this identifier while intercepting traffic.

For this attack, I proxied all of my traffic through the Burp Suite proxy. This allowed me to easily capture (and replay) all of the requests that Game Center was making to Apple servers. I did have to install the Portswigger CA certificate on my iPhone to intercept the SSL traffic. The prime targets for data enumeration were the "Friend Recommendations" and the friends of my friends list.

Since I wasn’t really using Game Center, I had to add some friends. I started by gathering all of the playerIDs for everyone in my recommended friends. This list appeared to be populated by recommendations based off of the people that I follow on Twitter and my Facebook friends. I also pulled down the top 150 users from the leader boards to add to my list as well. I intercepted an add request with Burp, moved the request into the intruder function and used my list of playerIDs (~250 IDs total) to automate the friend request process. After adding several friends (~20) I requested that Game Center send me a list of all of the friends of my friends. I then added them to the list of people to friend request. I should also note that requesting over 500 people as friends will probably result in your iPhone/iPad/etc. exploding in notifications of friend approvals.

I should note here that it would be very easy to set up a script to run once a day, to pull down a list of friends of friends and automatically friend request everyone that is one hop away from me.

Leader board listings

Listing of Friend Recommendations and the "Friends of a Friend" list

Attack

If you haven't figured it out yet, the inference attack that I will demonstrate has to do with guessing the email address from the SHA1 hash. Since we already have an alias or handle for the user, along with the person's first and last name, it's not a long stretch for us to try and guess the email address(es) tied to their iTunes account. After enumerating all of the information available for my recommended friends and next-hop friends (friends of my friends), I wrote a quick PowerShell script to read the data and generate potential email addresses.

Considering most people (that I know) use some variation of their name, or a handle for their email address it was pretty easy to generate variations to use for guessing. In order to

test multiple email domains, I appended each variation with hundreds of popular email address domains. Below is a sample of the potential email user names that I tested:

- kfosaaen@example.com
- k.fosaaen@example.com
- karlfosaaen@example.com
- karl.fosaaen@example.com
- karl.f@example.com
- karlf@example.com

After generating the potential emails, I then created SHA1 hashes of these email addresses and compared them to the hashes in the collected data. The script I used is really simple and may not have any practical use for you, but I put it out on GitHub: https://github.com/kfosaaen/EmailGenerator

After I wrote the PowerShell script, I realized that I could just use HashCat to do the generation (and some brute forcing) for me. I just used the generated emails as my dictionary and some custom rules to help with the address guessing.

Results

By the end of my data collection, I had attempted to add over five-hundred people as friends on Game Center. If you happened to be one of those people, I'm sorry. Overall, I was able to add 174 new friends to my Game Center account. Thanks to those friends, I was able to collect 1,635 records of Game Center ID, Alias, Email Hash, and Full Names. I actually stopped collecting records after I hit 45 friends, but I'm sure that I could get many more at this point. Those records also had to be paired down to remove special characters in user names, so the final list came out to 1,534 records. Of those, I was able to crack three hundred (19.5%) of the email addresses in about seven minutes.

This was all done over the course of about four days. Given

more time and a better script for generating potential email addresses, I think that the percentage would be a lot higher for the collected email addresses. I did stop the attack at the point where I felt that I had a good proof-of-concept, as I really didn't care to harvest a ton of emails. I have been in contact with Apple about this since January 10th, so they've had a fair amount of time to deal with the issue on their end.

Conclusion

I know this isn't a ground breaking attack that exposes tons of sensitive user data, but it's important to note that if a piece of data is important enough to hash, it should be hashed well. It should also not be available to all users. From an attacker's perspective, having this information would be very valuable to anyone trying to attack a specific iTunes account. If you're a Game Center user, you can protect your info by turning off the "Public Profile" feature in the Game Center settings.

In order to fix the issue, Apple should consider the business need for returning SHA1 hashes of user email addresses. I don't know what the hashes are currently being used for on the iPhone side, but there may be a need for them. If they are needed, then Apple should be salting these hashes to reduce the risk of an attacker cracking them.