CynoSure Prime

THURSDAY, SEPTEMBER 10, 2015

How we cracked millions of Ashley Madison bcrypt hashes efficiently

Not long after the release of the Ashley Madison leaks, many groups and individuals attempted to crack the bcrypt hashes. Since the developers used a cost factor of 12 for the bcrypt hash, this made the process an extremely compute intensive task. We decided to take a different approach and made some rather interesting discoveries.

Without much information about the $loginkey variable and how it was generated, we decided to dive into the second leak of git dumps. We identified two functions of interest and upon closer inspection, discovered that we could exploit these functions as helpers in accelerating the cracking of the bcrypt hashes.

Through the two insecure methods of $logkinkey generation observed in two different functions, we were able to gain enormous speed boosts in cracking the bcrypt hashed passwords. Instead of cracking the slow bcrypt hashes directly, which is the hot topic at the moment, we took a more efficient approach and simply attacked the md5(lc($username).”::”.lc($pass)) and md5(lc($username).”::”.lc($pass).”:”.lc($email).”:73@^bhhs&#@&^@8@*$”) tokens instead. Having cracked the token, we simply then had to case correct it against its bcrypt counterpart.

The $loginkey variable seemed to be used for automatic login, but we didn’t spend much time investigating further. It was generated upon user account creation and was re-generated when the user modified their account details including username, password and email address.

Discovery 1
Date: 31/08/2015
Filename: amlib_member_create.function.php
Function: amlib_member_create()
Lines of interest: 69, 70
Algorithm: md5(lc($username).”::”.lc($pass))
According to line 70 of the code, the $loginkey variable was generated by hashing the lowercased username and password with MD5. Great, did this mean that we could crack the password by attacking the loginkey with md5($salt.$pass)? Line 69 would suggest otherwise, since $password was set to the bcrypt hash by the “encryptPassword” function. We wondered if it had always been this way, and a quick git blame revealed that this line was changed on 2012-06-14 with commit 1c833ec7. Here’s the difference:

```
      $username = !empty($Values['username_suggest']) ? $Values['username_suggest'] : $Values['username'];
-      $password = User::encryptPassword($Values['password']);
+      $password = $Values['password'];
      $loginkey = md5(strtolower($username).'::'.strtolower($password));
```

This meant that we could crack accounts created prior to this date with simple salted MD5.
Also, the possible charset was reduced by 26 due to the use of strtolower().

Discovery 2
Date: 31/08/2015
Filename: AccountProvider.php
Function: generateLoginKey()
Lines of interest: 78, 79

Algorithm: md5(lc($username).”::”.lc($pass).”:”.lc($email).”:73@^bhhs&#@&^@8@*$”).
This function used a slightly different routine to generate the $loginkey as it incorporated the use of $username, $password, $email variables along with a constant salt string called $hash; and together, these variables were hashed with the MD5 algorithm.
It appeared that generateLoginKey() was invoked when a user modified their account attributes (username, password and email) and as a result of this, a new loginkey was issued for that account. From our understanding, it appeared that bcrypt was not always used to hash the password prior to it being fed to the generateLoginKey function. This meant that this method could be used to recover passwords of accounts which had been modified prior to this code change.

Exploiting the discoveries

Two different algorithms were added to MDXfind to support these discoveries. The simple version, MD5AM, implements the earlier code of md5(lc($username).”::”.lc($pass)); while MD5AM2 implements the more complex version, but pre-bcrypt. MD5AM2 uses both the username and the email address, as well as a fixed salt, in the form of md5(lc($username).”::”.lc($pass).”:”.lc($email).”:73@^bhhs&#@&^@8@*$”).

Both required this information to be extracted from separate SQL database entries, but were combined for ease of parsing into one string. The username/email combination was supplied to MDXfind with the -u switch as a separate file.

A third optimization was performed to identify and exclude MD5 hashes that could not be cracked using our discovered methods. These were $loginkeys created using the secure methods of md5(lc($usename).”::”.lc($bcrypt-string)). Since we had the username, and the bcrypt hash, we were able to isolate these hashes quickly.

Regardless, MDXfind was able to load all of the hashes, and because of the manner in which it stores and searches the hashes, we incurred negligible penalty from the search overhead of the “unsolvable” hashes. This allowed us to find more than 2.6 million passwords in just a few hours, using just *one* CPU box only.

Alternatively

The hashes from the the dump were also converted into an appropriate format suitable for loading as md5($salt.$pass) into any existing CPU/GPU crackers (only for Discovery 1).

Case Correction

Having the solved md5 tokens however, did not mean that we “knew” the original password. The token used only the lowercase value of the password and thus a secondary step to toggle the case of each character generating each variant was necessary, in order to properly crack the bcrypt hashes. Fortunately, this was a fixed-set problem with each bcrypt hash, thus only one salt needed to be checked for each bcrypt against the case variants.

A separate run correcting our cracked tokens against the bcrypt counterpart validated that we had in fact solved millions of bcrypt hashes...in days, not years. As of posting our team has successfully cracked over 11.2 Million of the bcrypt hashes.

Update:
Dan Gooden from Ars technica has also written an article which gives a thorough explanation of the process.

#FOLOW_US #JOIN_US #LOVE_US #HATE_US #CONTACT_US **@CynoPrime**
Twitter: @CynoPrime
Blog: cynosureprime.blogspot.com
Email: cynosureprime@gmail.com

Posted by CynoSure Prime at 3:35 AM

7 comments:

Miglen September 10, 2015 at 3:52 AM

Awesome work! I really hope that the people had changed their passwords already, otherwise lot's of cheaters will suffer. :)

Reply

Gabor September 10, 2015 at 4:01 AM

Nice finding

Reply

Brav0Hax September 10, 2015 at 4:35 AM

Good job, just point me to the pass list when you guys upload it ;)

Reply

Maria September 10, 2015 at 5:05 AM

Ah! That moment when you realize that you've been immensely outsmarted.
Great job!

Reply

taze September 10, 2015 at 5:42 AM

Check out the big brain on Brad ! Have fun sorting out the bots from the humans.

Reply

Chris Hamilton September 10, 2015 at 7:56 AM

Awesome, I hope you do some analysis of the passwords as far as a Uniques & Count and post it somewhere. Could be useful to strip out the top 100 and drop the bottom 30% and add to a dictionary.

Reply

Chris Hamilton September 10, 2015 at 7:56 AM

Awesome, I hope you do some analysis of the passwords as far as a Uniques & Count and post it somewhere. Could be useful to strip out the top 100 and drop the bottom 30% and add to a dictionary.

Reply

Enter your comment...

Comment as: ggyy (Google) Sign out

Publish Preview Notify me

Home Older Post

Subscribe to: Post Comments (Atom)

BLOG ARCHIVE

▼ 2015 (4)

▼ September (1)

How we cracked millions of Ashley Madison bcrypt h...

► August (1)

► May (2)