blog.codinghorror.com

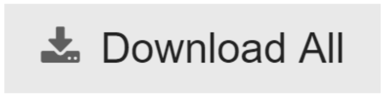
Hacker, Hack Thyself

12-15 minutes

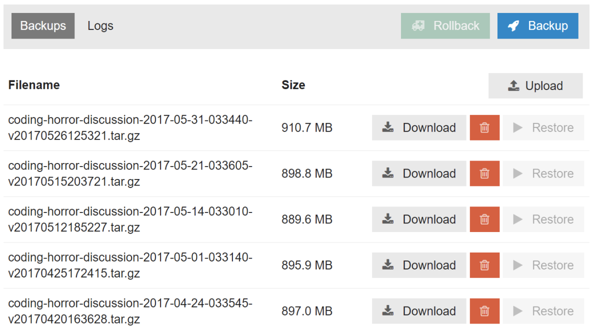
02 Jun 2017

We've read so many sad stories about communities that were fatally compromised or destroyed due to security exploits. We took that lesson to heart when we founded the Discourse project; we endeavor to build open source software that is secure and safe for communities by default, even if there are thousands, or millions, of them out there.

However, we also value *portability*, the ability to get your data into and out of Discourse at will. This is why Discourse, unlike other forum software, defaults to a Creative Commons license. As a basic user on any Discourse you can easily export and download all your posts right from your user page.



As a site owner, you can easily back up and restore your entire site database from the admin panel, right in your web browser. Automated weekly backups are set up for you out of the box, too. I'm not the world's foremost expert on backups for nothing, man!



Filename	Size			
coding-horror-discussion-2017-05-31-033440-v20170526125321.tar.gz	910.7 MB	Download		Restore
coding-horror-discussion-2017-05-21-033605-v20170515203721.tar.gz	898.8 MB	Download		Restore
coding-horror-discussion-2017-05-14-033010-v20170512185227.tar.gz	889.6 MB	Download		Restore
coding-horror-discussion-2017-05-01-033140-v20170425172415.tar.gz	895.9 MB	Download		Restore
coding-horror-discussion-2017-04-24-033545-v20170420163628.tar.gz	897.0 MB	Download		Restore

Over the years, we've learned that balancing security and data portability can be tricky. You bet your sweet ASCII a **full database download** is what hackers start working toward the minute they gain any kind of foothold in your system. It's the ultimate prize.

To mitigate this threat, we've slowly tightened restrictions around Discourse backups in various ways:

- Administrators have a minimum password length of 15 characters.
- Both backup creation and backup download administrator actions are formally logged.
- Backup download tokens are single use and emailed to the address of the administrator, to confirm that user has full control over the email address.

The name of the security game is defense in depth, so all these hardening steps help … but we still need to **assume that Internet Bad Guys will somehow get a copy of your database**. And then what? Well, what's in the database?

- Identity cookies

Cookies are, of course, how the browser can tell who you are.

Cookies are usually stored as hashes, rather than the actual cookie value, so having the hash doesn't let you impersonate the target user. Furthermore, most modern web frameworks rapidly cycle cookies, so they are only valid for a brief 10 to 15 minute window anyway.

- Email addresses

  Although users have reason to be concerned about their emails being exposed, very few people treat their email address as anything particularly precious these days.

- All posts and topic content

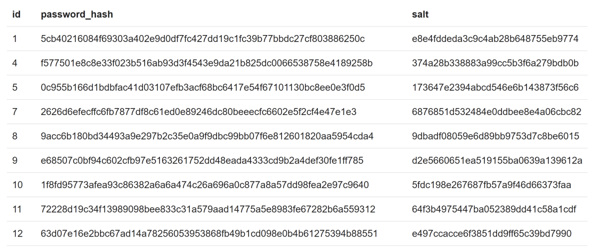
  Let's assume for the sake of argument that this is a fully public site and nobody was posting anything particularly sensitive there. So we're not worried, at least for now, about trade secrets or other privileged information being revealed, since they were all public posts anyway. If we were, that's a whole other blog post I can write at a later date.

- Password hashes

  What's left is **the password hashes**. And that's … a serious problem indeed.

  Now that the attacker has your database, they can crack your password hashes with large scale offline attacks, using the full resources of any cloud they can afford. And once they've cracked a particular password hash, **they can log in as that user … forever**. Or at least until that user changes their password.

  ⚠️ That's why, if you know (or even suspect!) your database was exposed, the very first thing you should do is reset everyone's password.



id	password_hash	salt
1	5cb40216084f69303a402e9d0df7fc427dd19c1fc39b77bbdc27cf803886250c	e8e4fddeda3c9c4ab28b648755eb9774
4	f577501e8c8e33f023b516ab93d3f4543e9da21b825dc0066538758e4189258b	374a28b338883a99cc5b3f6a279bdb0b
5	0c955b166d1bdbfac41d03107efb3acf68bc6417e54f67101130bc8ee0e3f0d5	173647e2394abcd546e6b143873f56c6
7	2626d6efecffc6fb7877df8c61ed0e89246dc80beeecfc6602e5f2cf4e47e1e3	6876851d532484e0ddbee8e4a06cbc82
8	9acc6b180bd34493a9e297b2c35e0a9f9dbc99bb07f6e812601820aa5954cda4	9dbadf08059e6d89bb9753d7c8be6015
9	e68507c0bf94c602cfb97e5163261752dd48eada4333cd9b2a4def30fe1ff785	d2e5660651ea519155ba0639a139612a
10	1f8fd95773afea93c86382a6a6a474c26a696a0c877a8a57dd98fea2e97c9640	5fdc198e267687fb57a9f46d66373faa
11	72228d19c34f13989098bee833c31a579aad14775a5e8983fe67282b6a559312	64f3b4975447ba052389dd41c58a1cdf
12	63d07e16e2bbc67ad14a78256053953868fb49b1cd098e0b4b61275394b88551	e497ccacce6f3851dd9ff65c39bd7990

  But what if you *don't* know? Should you preemptively reset everyone's password every 30 days, like the world's worst bigco IT departments? That's downright user hostile, and leads to serious pathologies of its own. The reality is that you probably *won't* know when your database has been exposed, at least not until it's too late to do anything about it. So it's crucial to slow the attackers down, to give yourself time to deal with it and respond.

  Thus, the only real protection you can offer your users is just how resistant to attack your stored password hashes are. There are two factors that go into password hash strength:

1. **The hashing algorithm**. As slow as possible, and ideally designed to be *especially* slow on GPUs for reasons that will become painfully obvious about 5 paragraphs from now.
2. **The work factor** or **number of iterations**. Set this as high as possible, without opening yourself up to a possible denial of service attack.

   I've seen guidance that said you should set the overall work factor high enough that hashing a password takes at least 8ms on the target platform. It turns out Sam Saffron, one of my Discourse co-founders, made a good call back in 2013 when he selected the

NIST recommendation of **PBKDF2-HMAC-SHA256** and **64k iterations**. We measured, and that indeed takes roughly 8ms using our existing Ruby login code on our current (fairly high end, Skylake 4.0 Ghz) servers.

But that was 4 years ago. Exactly how secure are our password hashes in the database today? Or 4 years from now, or 10 years from now? We're building open source software for the long haul, and we need to be sure we are making reasonable decisions that protect everyone. So in the spirit of designing for evil, it's time to put on our Darth Helmet and play the bad guy – **let's crack our own hashes!**

We're gonna use the biggest, baddest single GPU out there at the moment, the GTX 1080 Ti. As a point of reference, for PBKDF2-HMAC-SHA256 the 1080 achieves 1180 kH/s, whereas the 1080 Ti achieves 1640 kH/s. In a *single* video card generation the attack hash rate has increased nearly 40 percent. Ponder that.

First, a tiny hello world test to see if things are working. I downloaded hashcat. I logged into our demo at try.discourse.org and created a new account with the password `0234567890`; I checked the database, and this generated the following values in the hash and salt database columns for that new user:

hash	`93LlpbKZKficWfV9jjQNOSp39MT0pDPtYx7/gBLl5jw=`
salt	`ZWVhZWQ4YjZmODU4Mzc0M2E2ZDRlNjBkNjY3YzE2ODA=`

Hashcat requires the following input file format: one line per hash, with the hash type, number of iterations, salt and hash (base64 encoded) separated by colons:

```
type   iter  salt
hash
sha256:64000:ZWVhZWQ4YjZmODU4Mzc0M2E2ZDRlNjBkNjY3YzE2ODA=:93LlpbKZKficWfV9jjQNOSp39MT
```

Let's hashcat it up and see if it works:

```
./h64 -a 3 -m 10900 .\one-hash.txt 0234567?d?d?d
```

Note that this is an intentionally tiny amount of work, it's only guessing three digits. And sure enough, we cracked it fast! See the password there on the end? We got it.

```
sha256:64000:ZWVhZWQ4YjZmODU4Mzc0M2E2ZDRlNjBkNjY3YzE2ODA=:93LlpbKZKficWfV9jjQNOSp39MT
```

Now that we know it works, let's get down to business. But we'll start easy. How long does it take to brute force attack **the easiest possible Discourse password, 8 numbers** – that's "only" 10^8 combinations, a little over one hundred million.

```
Hash.Type........: PBKDF2-HMAC-SHA256
Time.Estimated...: Fri Jun 02 00:15:37 2017 (1
hour, 0 mins)
Guess.Mask.......: ?d?d?d?d?d?d?d?d [8]
```

Even with a top of the line GPU that's … OK, I guess. Remember this is just one hash we're testing against, so you'd need one hour per row (user) in the table. And I have more bad news for you: Discourse hasn't allowed 8 character passwords for quite some time now. How long does it take if we try longer numeric passwords?

```
?d?d?d?d?d?d?d?d?d [9]
Fri Jun 02 10:34:42 2017 (11 hours, 18 mins)

?d?d?d?d?d?d?d?d?d?d [10]
Tue Jun 06 17:25:19 2017 (4 days, 18 hours)

?d?d?d?d?d?d?d?d?d?d?d [11]
Mon Jul 17 23:26:06 2017 (46 days, 0 hours)

?d?d?d?d?d?d?d?d?d?d?d?d [12]
Tue Jul 31 23:58:30 2018 (1 year, 60 days)
```

But all digit passwords are easy mode, for babies! How about some *real* passwords that use at least lowercase letters, or lowercase + uppercase + digits?

```
Guess.Mask.......: ?l?l?l?l?l?l?l?l [8]
Time.Estimated...: Mon Sep 04 10:06:00 2017 (94
days, 10 hours)

Guess.Mask.......: ?1?1?1?1?1?1?1?1 [8] (-1 =
?l?u?d)
Time.Estimated...: Sun Aug 02 09:29:48 2020 (3
years, 61 days)
```

A brute force try-every-single-letter-and-number attack is not looking so hot for us at this point, even with a high end GPU. But what if we divided the number by **eight** … by putting eight video cards in a single machine? That's well within the reach of a small business budget or a wealthy individual. Unfortunately, dividing 38 months by 8 isn't such a dramatic reduction in the time to attack. Instead, let's talk about nation state attacks where they have the budget to throw *thousands* of these GPUs at the problem (1.1 days), maybe even *tens of thousands* (2.7 hours), then … yes. Even allowing for 10 character password minimums, you are in serious trouble at that point.

If we want Discourse to be nation state attack resistant, clearly we'll need to do better. Hashcat has a handy benchmark mode, and here's a sorted list of the strongest (slowest) hashes that Hashcat knows about benchmarked on a rig with 8 Nvidia GTX 1080 GPUs. Of the things I recognize on that list, **bcrypt**, **scrypt** and **PBKDF2-HMAC-SHA512** stand out.

My quick hashcat results gave me some confidence that we weren't doing anything terribly wrong with the Discourse password hashes stored in the database. But I wanted to be *completely sure*, so I hired someone with a background in security and penetration testing to, under a signed NDA, try cracking the password hashes of two live and very popular Discourse sites we currently host.

> I was provided two sets of password hashes from two different Discourse communities, containing 5,909 and 6,088 hashes respectively. Both used the PBKDF2-HMAC-SHA256 algorithm with a work factor of 64k. Using hashcat, my Nvidia GTX 1080 Ti GPU generated these hashes at a rate of ~27,000/sec.
>
> Common to all discourse communities are various password requirements:
>
> - All users must have a minimum password length of 10 characters.
> - All administrators must have a minimum password length of 15 characters.
> - Users cannot use any password matching a blacklist of the 10,000 most commonly used passwords.
> - Users can choose to create a username and password or use various third party authentication mechanisms (Google, Facebook, Twitter, etc). If this option is selected, a secure random 32 character password is autogenerated. It is not possible to know whether any given password is human entered, or autogenerated.
>
> Using common password lists and masks, I cracked 39 of the 11,997 hashes in about three weeks, 25 from the [redacted] community and 14 from the [redacted] community.

This is a security researcher who commonly runs these kinds of audits, so all of the attacks used **wordlists**, along with known effective patterns and masks derived from the researcher's previous password cracking experience, instead of raw brute force. That recovered the following passwords (and one duplicate):

```
007007bond     l3tm3innow
123password    Neversaynever
1qaz2wsx3e     password1235
A3eilm2s2y     pittsburgh1
Alexander12    Playstation2
alexander18    Playstation3
belladonna2    Qwerty1234
Charlie123     Qwertyuiop1
Chocolate1     qwertyuiop1234567890
christopher8   Spartan117
Elizabeth1     springfield0
Enterprise01   Starcraft2
Freedom123     strawberry1
greengrass123  Summertime
hellothere01   Testing123
I123456789     testing1234
Iamawesome     thecakeisalie02
khristopher    Thirteen13
l1ghthouse     Welcome123
```

If we multiply this effort by 8, and double the amount of time allowed, it's conceivable that a *very* motivated attacker, or one with a sophisticated set of wordlists and masks, could eventually recover 39 × 16 = 624 passwords, or about **five percent** of the total users. That's reasonable, but higher than I would like. We absolutely plan to add a hash type table in future versions of Discourse, so we can switch to an even more secure (read: much slower) password hashing scheme in the next year or two.

```
bcrypt $2*$, Blowfish (Unix)
  20273 H/s

scrypt
  886.5 kH/s

PBKDF2-HMAC-SHA512
  542.6 kH/s

PBKDF2-HMAC-SHA256
 1646.7 kH/s
```

After this exercise, I now have a much deeper understanding of our worst case security scenario, a database compromise combined with a professional offline password hashing attack. I can also more confidently recommend and stand behind our engineering work in making Discourse secure for everyone. So if, like me, you're not entirely sure you are doing things securely, it's time to put those assumptions to the test. Don't wait around for hackers to attack you — **hacker, hack thyself!**

[advertisement] At Stack Overflow, we put developers first. We already help you find answers to your tough coding questions; now let us help you find your next job.