Anonymous speaks: the inside story of the HBGary hack

By Peter Bright | Published 7 months ago

Rainbow tables

Specifically, the attackers grabbed the user database from the CMS—the list of usernames, e-mail addresses, and password hashes for the HBGary employees authorized to make changes to the CMS. In spite of the rudimentary SQL injection flaw, the designers of the CMS system were not completely oblivious to security best practices; the user database did not store plain readable passwords. It stored only hashed passwords—passwords that have been mathematically processed with a hash function to yield a number from which the original password can't be deciphered.

The key part is that you can't go backwards—you can't take the hash value and convert it back into a password. With a hash algorithm, traditionally the only way to figure out the original password was to try every single possible password in turn, and see which one matched the hash value you have. So, one would try "a," then "b," then "c"... then "z," then "aa," "ab," and so on and so forth.

To make this more difficult, hash algorithms are often quite slow (deliberately), and users are encouraged to use long passwords which mix lower case, upper case, numbers, and symbols, so that these brute force attacks have to try *even more* potential passwords until they find the right one. Given the number of passwords to try, and the slowness of hash algorithms, this normally takes a very long time. Password cracking software to perform this kind of brute force attack has long been available, but its success at cracking complex passwords is low.

However, a technique first published in 2003 (itself a refinement of a technique described in 1980) gave password crackers an alternative approach. By pre-computing large sets of data and generating what are known as rainbow tables, the attackers can make a trade-off: they get much faster password cracks in return for using much more space. The rainbow table lets the password cracker pre-compute and store a large number of hash values and the passwords that generated them. An attacker can then look up the hash value that they are interested in and see if it's in the table. If it is, they can then read out the password.

To make cracking harder, good password hash implementations will use a couple of additional techniques. The first is iterative hashing: simply put, the output of the hash function is itself hashed with the hash function, and this process is repeated thousands of times. This makes the hashing process considerably slower, hindering both brute-force attacks and rainbow table generation.

The second technique is salting; a small amount of random data is added to the password before hashing it, greatly expanding the size of rainbow table that would be required to get the password.

In principle, any hash function can be used to generate rainbow tables. However, it takes more time to generate rainbow tables for slow hash functions than it does for fast ones, and hash functions that produce a short hash value require less storage than ones that produce long hash values. So in practice, only a few hash algorithms have widely available rainbow table software available. The best known and most widely supported of these is probably MD5, which is quick to compute and produces an output that is only 128 bits (16 bytes) per hash. These factors together make it particularly vulnerable to rainbow table attacks. A number of software projects exist that allow the generation or downloading of MD5 rainbow tables, and their subsequent use to crack passwords.

As luck would have it, the hbgaryfederal.com CMS used MD5. What's worse is that it used MD5 badly: there was no iterative hashing and no salting. The result was that the downloaded passwords were highly susceptible to rainbow table-based attacks, performed using a rainbow table-based password cracking website. And so this is precisely what the attackers did; they used a rainbow table cracking tool to crack the hbgaryfederal.com CMS passwords.

Even with the flawed usage of MD5, HBGary could have been safe thanks to a key limitation of rainbow tables: each

table only spans a given "pattern" for the password. So for example, some tables may support "passwords of 1-8 characters made of a mix of lower case and numbers," while other can handle only "passwords of 1-12 characters using upper case only."

A password that uses the full range of the standard 95 typeable characters (upper and lower case letters, numbers, and the standard symbols found on a keyboard) and which is unusually long (say, 14 or more characters) is unlikely to be found in a rainbow table, because the rainbow table required for such passwords will be too big and take too long to generate.

Alas, two HBGary Federal employees—CEO Aaron Barr and COO Ted Vera—used passwords that were very simple; each was just six lower case letters and two numbers. Such simple combinations are likely to be found in any respectable rainbow table, and so it was that their passwords were trivially compromised.

For a security company to use a CMS that was so flawed is remarkable. Improper handling of passwords—iterative hashing, using salts and slow algorithms—and lack of protection against SQL injection attacks are *basic errors*. Their system did not fall prey to some subtle, complex issue: it was broken into with basic, well-known techniques. And though not all the passwords were retrieved through the rainbow tables, two were, because they were so poorly chosen.

HBGary owner Penny Leavy said in a later IRC chat with Anonymous that the company responsible for implementing the CMS has since been fired.

Password problems

Still, badly chosen passwords aren't such a big deal, are they? They might have allowed someone to deface the hbgaryfederal.com website—admittedly embarrassing—but since everybody knows that you shouldn't reuse passwords across different systems, that should have been the extent of the damage, surely?

Unfortunately for HBGary Federal, it was not. Neither Aaron nor Ted followed best practices. Instead, they used the same password in a whole bunch of different places, including e-mail, Twitter accounts, and LinkedIn. For both men, the passwords allowed retrieval of e-mail. However, that was not all they revealed. Let's start with Ted's password first.

Along with its webserver, HBGary had a Linux machine, support.hbgary.com, on which many HBGary employees had shell accounts with ssh access, each with a password used to authenticate the user. One of these employees was Ted Vera, and his ssh password was identical to the cracked password he used in the CMS. This gave the hackers immediate access to the support machine.

ssh doesn't have to use passwords for authentication. Passwords are certainly common, but they're also susceptible to this kind of problem (among others). To combat this, many organizations and users, particularly those with security concerns, do not use passwords for ssh authentication. Instead, they use public key cryptography: each user has a key made up of a private part and a public part. The public part is associated with their account, and the private part is kept, well, private. ssh then uses these two keys to authenticate the user.

Since these private keys are not as easily compromised as passwords—servers don't store them, and in fact they never leave the client machine—and aren't readily re-used (one set of keys might be used to authenticate with several servers, but they can't be used to log in to a website, say), they are a much more secure option. Had they been used for HBGary's server, it would have been safe. But they weren't, so it wasn't.

Although attackers could log on to this machine, the ability to look around and break stuff was curtailed: Ted was only a regular non-superuser. Being restricted to a user account can be enormously confining on a Linux machine. It spoils all your fun; you can't read other users' data, you can't delete files you don't own, you can't cover up the evidence of your own break-in. It's a total downer for hackers.

The only way they can have some fun is to elevate privileges through exploiting a privilege escalation vulnerability. These crop up from time to time and generally exploit flaws in the operating system kernel or its system libraries to trick it into giving the user more access to the system than should be allowed. By a stroke of luck, the HBGary system was vulnerable to just such a flaw. The error was published in October last year, conveniently with a full, working exploit. By November, most distributions had patches available, and there was no good reason to be running the exploitable code in February 2011.

Exploitation of this flaw gave the Anonymous attackers full access to HBGary's system. It was then that they discovered many gigabytes of backups and research data, which they duly purged from the system.

Aaron's password yielded even more fruit. HBGary used Google Apps for its e-mail services, and for both Aaron and Ted, the password cracking provided access to their mail. But Aaron was no mere *user* of Google Apps: his account was also the *administrator* of the company's mail. With his higher access, he could reset the passwords of any mailbox and hence gain access to all the company's mail—not just his own. It's this capability that yielded access to Greg Hoglund's mail.

And what was done with Greg's mail?

A little bit of social engineering, that's what.