Anonymous speaks: the inside story of the HBGary hack

By Ars Staff

Published: 2011-02-16 · Archived: 2026-04-05 13:23:48 UTC

After interviews with the hackers from Anonymous who invaded HBGary Federal …

It has been an embarrassing week for security firm HBGary and its HBGary Federal offshoot. HBGary Federal CEO Aaron Barr thought he had unmasked the hacker hordes of Anonymous and was preparing to name and shame those responsible for co-ordinating the group's actions, including the denial-of-service attacks that hit MasterCard, Visa, and other perceived enemies of WikiLeaks late last year.

When Barr told one of those he believed to be an Anonymous ringleader about his forthcoming exposé, the Anonymous response was swift and humiliating. HBGary's servers were broken into, its e-mails pillaged and published to the world, its data destroyed, and its website defaced. As an added bonus, a second site owned and operated by Greg Hoglund, owner of HBGary, was taken offline and the user registration database published.

Over the last week, I've talked to some of those who participated in the HBGary hack to learn in detail how they penetrated HBGary's defenses and gave the company such a stunning black eye—and what the HBGary example means for the rest of us mere mortals who use the Internet.

Anonymous: more than kids

HBGary and HBGary Federal position themselves as experts in computer security. The companies offer both software and services to both the public and private sectors. On the software side, HBGary has a range of

computer forensics and malware analysis tools to enable the detection, isolation, and analysis of worms, viruses, and trojans. On the services side, it offers expertise in implementing intrusion detection systems and secure networking, and performs vulnerability assessment and penetration testing of systems and software. A variety of three letter agencies, including the NSA, appeared to be in regular contact with the HBGary companies, as did Interpol, and HBGary also worked with well-known security firm McAfee. At one time, even Apple expressed an interest in the company's products or services.

Greg Hoglund's rootkit.com is a respected resource for discussion and analysis of rootkits (software that tampers with operating systems at a low level to evade detection) and related technology; over the years, his site has been targeted by disgruntled hackers aggrieved that their wares have been discussed, dissected, and often disparaged as badly written bits of code.

One might think that such an esteemed organization would prove an insurmountable challenge for a bunch of disaffected kids to hack. World-renowned, government-recognized experts against Anonymous? HBGary should be able to take their efforts in stride.

Unfortunately for HBGary, neither the characterization of Anonymous nor the assumption of competence on the security company's part are accurate, as the story of how HBGary was hacked will make clear.

Anonymous is a diverse bunch: though they tend to be younger rather than older, their age group spans decades. Some may still be in school, but many others are gainfully employed office-workers, software developers, or IT support technicians, among other things. With that diversity in age and experience comes a diversity of expertise and ability.

It's true that most of the operations performed under the Anonymous branding have been relatively unsophisticated, albeit effective: the attacks made on MasterCard and others were distributed denial-of-service attacks using a modified version of the Low Orbit Ion Cannon (LOIC) load-testing tool. The modified LOIC enables the creation of large botnets that each user *opts into*: the software can be configured to take its instructions from connections to Internet relay chat (IRC) chat servers, allowing attack organizers to remotely control hundreds of slave machines and hence control large-scale attacks that can readily knock websites offline.

According to the leaked e-mails, Aaron Barr believed that HBGary's website was itself subject to a denial-of-service attack shortly after he exposed himself to someone he believed to be a top Anonymous leader. But the person I spoke to about this denied any involvement in such an attack. Which is not to say that the attack didn't happen—simply that this person didn't know about or participate in it. In any case, the Anonymous plans were more advanced than a brute force DDoS.

Time for an injection

HBGary Federal's website, hbgaryfederal.com, was powered by a content management system (CMS). CMSes are a common component of content-driven sites; they make it easy to add and update content to the site without having to mess about with HTML and making sure everything gets linked up and so on and so forth. Rather than using an off-the-shelf CMS (of which there are many, used in the many blogs and news sites that exist on the Web), HBGary—for reasons best known to its staff—decided to commission a custom CMS system from a third-party developer.

Unfortunately for HBGary, this third-party CMS was poorly written. In fact, it had what can only be described as a pretty gaping bug in it. A standard, off-the-shelf CMS would be no panacea in this regard—security flaws crop up in all of them from time to time—but it would have the advantage of many thousands of users and regular bugfixes, resulting in a much lesser chance of extant security flaws.

The custom solution on HBGary's site, alas, appeared to lack this kind of support. And if HBGary conducted any kind of vulnerability assessment of the software—which is, after all, one of the services the company offers—then its assessment overlooked a substantial flaw.

The hbgaryfederal.com CMS was susceptible to a kind of attack called SQL injection. In common with other CMSes, the hbgaryfederal.com CMS stores its data in an SQL database, retrieving data from that database with suitable queries. Some queries are fixed—an integral part of the CMS application itself. Others, however, need parameters. For example, a query to retrieve an article from the CMS will generally need a parameter corresponding to the article ID number. These parameters are, in turn, generally passed from the Web front-end to the CMS.

SQL injection is possible when the code that deals with these parameters is faulty. Many applications join the parameters from the Web front-end with hard-coded queries, then pass the whole concatenated lot to the database. Often, they do this without verifying the validity of those parameters. This exposes the systems to SQL injection. Attackers can pass in specially crafted parameters that cause the database to execute queries of the attackers' own choosing.

The exact URL used to break into hbgaryfederal.com was `http://www.hbgaryfederal.com/pages.php?pageNav=2&page=27`. The URL has two parameters named pageNav and page, set to the values 2 and 27, respectively. One or other or both of these was handled incorrectly by the CMS, allowing the hackers to retrieve data from the database that they shouldn't have been able to get.

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.

A little help from my friends

Contained within Greg's mail were two bits of useful information. One: the root password to the machine running Greg's rootkit.com site was either "88j4bb3rw0cky88" or "88Scr3am3r88". Two: Jussi Jaakonaho, "Chief Security Specialist" at Nokia, had root access. Vandalizing the website stored on the machine was now within reach.

The attackers just needed a little bit more information: they needed a regular, non-root user account to log in with, because as a standard security procedure, direct ssh access with the root account is disabled. Armed with the two pieces of knowledge above, and with Greg's e-mail account in their control, the social engineers set about their task. The e-mail correspondence tells the whole story:

> ```
> From: Greg
> To: Jussi
> Subject: need to ssh into rootkit
> im in europe and need to ssh into the server. can you drop open up
> ```

> firewall and allow ssh through port 59022 or something vague?
> and is our root password still 88j4bb3rw0cky88 or did we change to
> 88Scr3am3r88 ?
> thanks
>
> ———————————————————-
>
> From: Jussi
> To: Greg
> Subject: Re: need to ssh into rootkit
> hi, do you have public ip? or should i just drop fw?
> and it is w0cky - tho no remote root access allowed
>
> ———————————————————-
>
> From: Greg
> To: Jussi
> Subject: Re: need to ssh into rootkit
> no i dont have the public ip with me at the moment because im ready
> for a small meeting and im in a rush.
> if anything just reset my password to changeme123 and give me public
> ip and ill ssh in and reset my pw.
>
> ———————————————————-
>
> From: Jussi
> To: Greg
> Subject: Re: need to ssh into rootkit
> ok,
> it should now accept from anywhere to 47152 as ssh. i am doing
> testing so that it works for sure.
> your password is changeme123
>
> i am online so just shoot me if you need something.
>
> in europe, but not in finland? :-)
>
> _jussi
>
> ———————————————————-
>
> From: Greg
> To: Jussi
> Subject: Re: need to ssh into rootkit

> ```
> if i can squeeze out time maybe we can catch up.. ill be in germany
> for a little bit.
>
> anyway I can't ssh into rootkit. you sure the ips still
> 65.74.181.141?
>
> thanks
> ```
>
> _____
>
> ```
> From: Jussi
> To: Greg
> Subject: Re: need to ssh into rootkit
> does it work now?
> ```
>
> _____
>
> ```
> From: Greg
> To: Jussi
> Subject: Re: need to ssh into rootkit
> yes jussi thanks
>
> did you reset the user greg or?
> ```
>
> _____
>
> ```
> From: Jussi
> To: Greg
> Subject: Re: need to ssh into rootkit
> nope. your account is named as hoglund
> ```
>
> _____
>
> ```
> From: Greg
> To: Jussi
> Subject: Re: need to ssh into rootkit
> yup im logged in thanks ill email you in a few, im backed up
>
> thanks
> ```

Thanks indeed. To be fair to Jussi, the fake Greg appeared to know the root password and, well, the e-mails were coming from Greg's own e-mail address. But over the course of a few e-mails it was clear that "Greg" had forgotten both his username *and* his password. And Jussi handed them to him on a platter.

Later on, Jussi did appear to notice something was up:

> From: Jussi
> To: Greg
> Subject: Re: need to ssh into rootkit
> did you open something running on high port?

As with the HBGary machine, this could have been avoided if keys had been used instead of passwords. But they weren't. Rootkit.com was now compromised.

Standard practice

Once the username and password were known, defacing the site was easy. Log in as Greg, switch to root, and deface away! The attackers went one better than this, however: they dumped the user database for rootkit.com, listing the e-mail addresses and password hashes for everyone who'd ever registered on the site. And, as with the hbgaryfederal.com CMS system, the passwords were hashed with a single naive use of MD5, meaning that once again they were susceptible to rainbow table-based password cracking. So the crackable passwords were cracked, too.

So what do we have in total? A Web application with SQL injection flaws and insecure passwords. Passwords that were badly chosen. Passwords that were reused. Servers that allowed password-based authentication. Systems that weren't patched. And an astonishing willingness to hand out credentials over e-mail, even when the person being asked for them should have realized something was up.

The thing is, none of this is unusual. Quite the opposite. The Anonymous hack was not exceptional: the hackers used standard, widely known techniques to break into systems, find as much information as possible, and use that information to compromise further systems. They didn't have to, for example, use any non-public vulnerabilities or perform any carefully targeted social engineering. And because of their desire to cause significant public disruption, they did not have to go to any great lengths to hide their activity.

Nonetheless, their attack was highly effective, and it was well-executed. The desire was to cause trouble for HBGary, and that they did. Especially in the social engineering attack against Jussi, they used the right

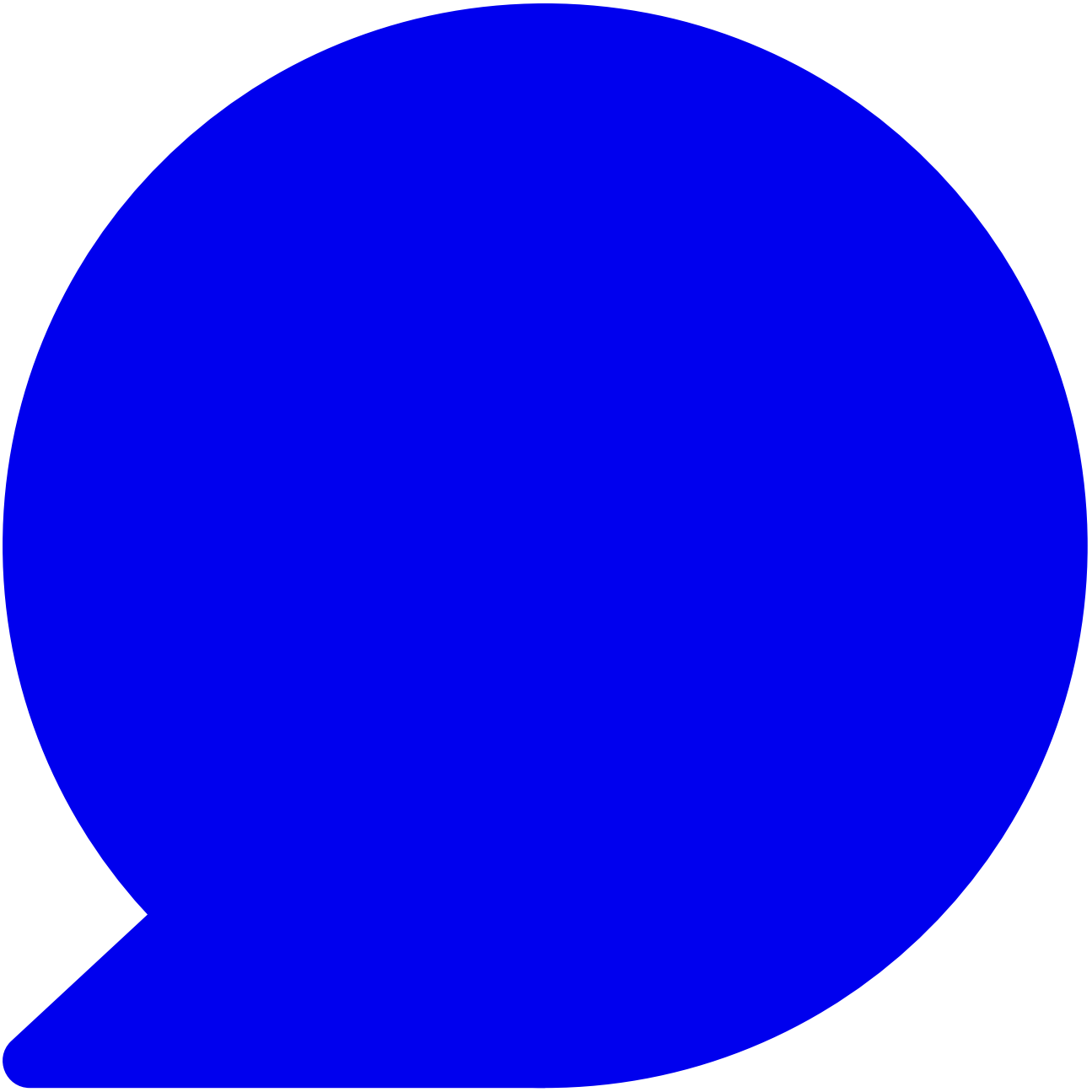
information in the right way to seem credible.

Most frustrating for HBGary must be the knowledge that they know what they did wrong, and they were perfectly aware of best practices; they just didn't actually *use them*. Everybody *knows* you don't use easy-to-crack passwords, but some employees did. Everybody *knows* you don't re-use passwords, but some of them did. Everybody *knows* that you should patch servers to keep them free of known security flaws, but they didn't.

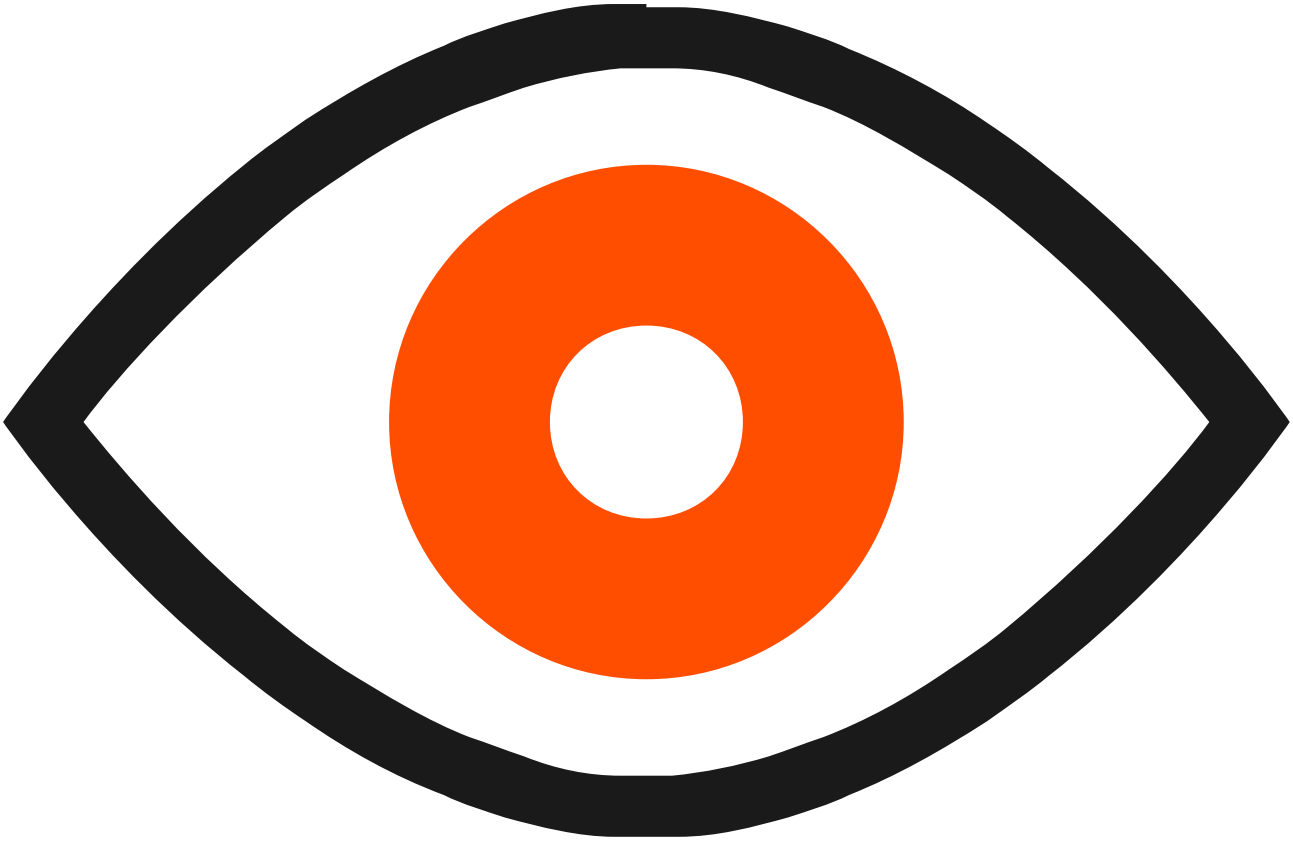
And HBGary isn't alone. Analysis of the passwords leaked from rootkit.com and Gawker shows that password re-use is extremely widespread, with something like 30 percent of users re-using their passwords. HBGary won't be the last site to suffer from SQL injection, either, and people will continue to use password authentication for secure systems because it's so much more convenient than key-based authentication.

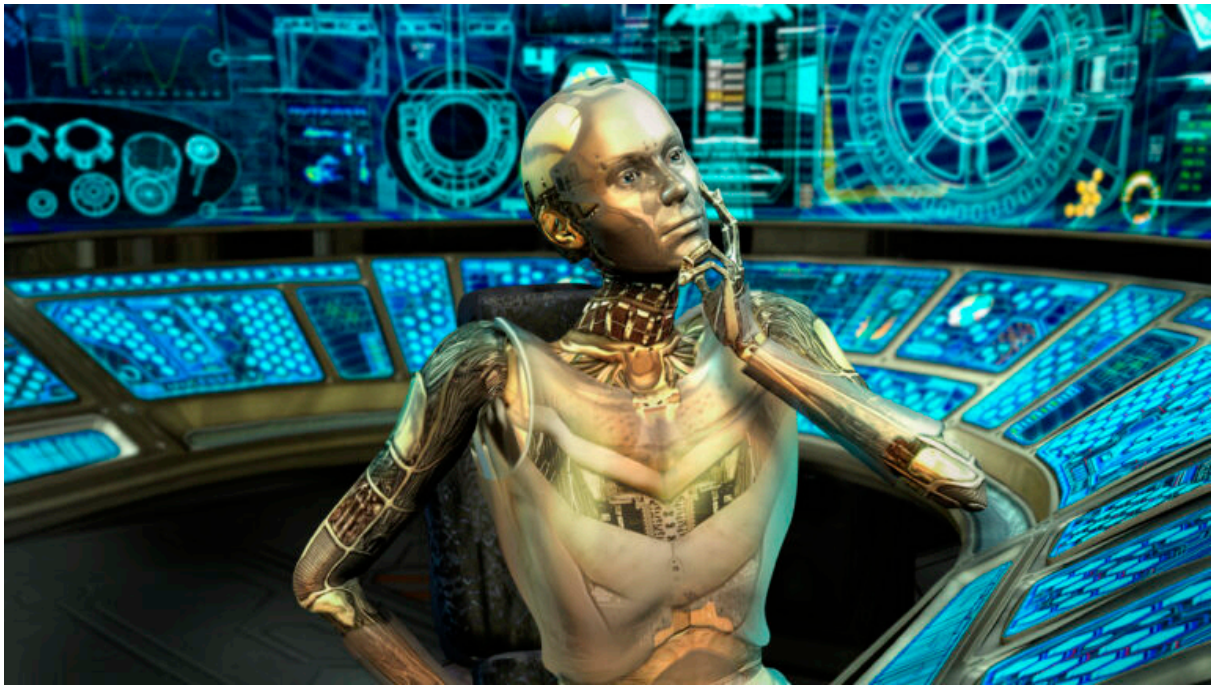
So there are clearly two lessons to be learned here. The first is that the standard advice is good advice. If all best practices had been followed then none of this would have happened. Even if the SQL injection error was still present, it wouldn't have caused the cascade of failures that followed.

The second lesson, however, is that the standard advice isn't good enough. Even recognized security experts who should know better won't follow it. What hope does that leave for the rest of us?



397 Comments



1. 
2.
3.
4.
5.

Source: https://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/