What is the best Distributed Brute Force countermeasure? [closed]

Asked 15 years, 11 months ago Modified 2 years, 8 months ago

Viewed 19k times

158

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 2 years ago.

The community reviewed whether to reopen this question 2 years ago and left it closed:

Original close reason(s) were not resolved

Improve this question

First, a little background: It is no secret that I am implementing an auth+auth system for CodeIgniter, and so far I'm winning (so to speak). But I've run into a pretty non-trivial challenge (one that most auth libraries miss entirely, but I insist on handling it properly): how to deal intelligently with **large-scale, distributed, variable-username brute-force attacks**.

I know all the usual tricks:

1. **Limiting # of failed attempts per IP/host** and denying the offenders access (e.g. Fail2Ban) - which no longer works since botnets have grown smarter
2. Combining the above with a **blacklist of known 'bad' IPs/hosts** (e.g. DenyHosts) - which relies on botnets falling for #1, which they increasingly don't
3. **IP/host whitelists** combined with traditional auth (sadly useless with dynamic IP users and the high churn on most web sites)
4. Setting a **sitewide limit** on # of failed attempts within a N minute/hour period, and throttling (suspending) all login attempts after that for a number of minutes/hours (with the problem that DoS attacking you becomes botnet child's play)
5. Mandatory **digital signatures** (public-key certificates) or RSA hardware tokens for all users with NO login/password option (without question a rock-solid solution, but only practical for closed, dedicated services)
6. Enforced **ultra-strong password schemes** (e.g. >25 nonsense characters with symbols - again, too impractical for casual users)
7. And finally, **CAPTCHAs** (which could work in most cases, but are annoying for users and virtually useless against a determined, resourceful attacker)

Now, these are just the theoretically workable ideas. There are *plenty* of rubbish ideas that blow the site wide open (e.g. to trivial DoS attacks). What I want is something better. And by better, I mean:

- It has to be secure(+) against DoS and brute-force attacks, and not introduce any new vulnerabilities that might allow a slightly sneakier bot to continue operating under the radar
- It has to be automated. If it requires a human operator to verify each login or monitor suspicious activity, it's not going to work in a real-world scenario
- It has to be feasible for mainstream web use (ie. high churn, high volume, and open registration that can be performed by non-programmers)
- It can't impede the user experience to the point where casual users will get annoyed or frustrated (and potentially abandon the site)
- It can't involve kittens, unless they are *really really secure* kittens

(+) *By 'secure', I mean at least as secure as a paranoid user's ability to keep his password secret*

So - let's hear it! *How would you do it*? Do you know of a best-practice that I haven't mentioned (oh please say you do)? I admit I do have an idea of my own (combining ideas from 3 and 4), but I'll let the true experts speak before embarrassing myself ;-)

security authentication brute-force

Share

Improve this question

Follow

asked Jan 26, 2009 at 9:37

Jens Roland

27.8k ● 15 ● 84 ● 105

16 Answers

Sorted by: Highest score (default)

70

Combining methods 3 and 4 from the original post into a kind of 'fuzzy' or dynamic whitelist, and then - and here's the trick - *not blocking non-whitelisted IPs, just throttling them to hell and back*.

> Note that this measure is *only* meant to thwart this very specific type of attack. In practice, of course, it would work in combination with other best-practices approaches to auth: fixed-username throttling, per-IP throttling, code-enforced strong password policy, unthrottled cookie login, hashing all password equivalents before saving them, never using security questions, etc.

Assumptions about the attack scenario

If an attacker is targeting variable usernames, our username throttling doesn't fire. If the attacker is using a botnet or has access to a large IP range, our IP throttling is powerless. If the attacker has pre-scraped our userlist (usually possible on open-registration web services), we can't detect an ongoing attack based on number of 'user not found' errors. And if we enforce a restrictive system-wide (all usernames, all IPs) throttling, any such attack will DoS our entire site for the duration of the attack plus the throttling period.

So we need to do something else.

The first part of the countermeasure: Whitelisting

What we can be fairly sure of, is that the attacker is not able to detect and dynamically spoof the IP addresses of several thousand of our users(+). Which makes *whitelisting* feasible. In other words: for each user, we store a list of the (hashed) IPs from where the user has previously (recently) logged in.

Thus, our whitelisting scheme will function as a locked 'front door', where a user must be connected from one of his recognized 'good' IPs in order to log in at all. A brute-force attack on this 'front door' would be practically impossible(+).

(+) unless the attacker 'owns' either the server, all our users' boxes, or the connection itself -- and in those cases, we no longer have an 'authentication' issue, we have a genuine franchise-sized pull-the-plug FUBAR situation

The second part of the countermeasure: System-wide throttling *of unrecognized IPs*

In order to make a whitelist work for an open-registration web service, where users switch computers frequently and/or connect from dynamic IP addresses, we need to keep a 'cat door' open for users connecting from unrecognized IPs. The trick is to design that door so botnets get stuck, and so legitimate users get bothered *as little as possible*.

In my scheme, this is achieved by setting a *very* restrictive maximum number of failed login attempts by unapproved IPs over, say, a 3-hour period (it may be wiser to use a shorter or longer period depending on type of service), and making that restriction **global**, ie. for all user accounts.

Even a slow (1-2 minutes between attempts) brute force would be detected and thwarted quickly and effectively using this method. Of course, a *really slow* brute force could still remain unnoticed, but too slow speeds defeat the very purpose of the brute force attack.

What I am hoping to accomplish with this throttling mechanism is that if the maximum limit is reached, our 'cat door' slams closed for a while, but our front door remains open to legitimate users connecting by usual means:

- Either by connecting from one of their recognized IPs

- Or by using a persistent login cookie (from anywhere)

The only legitimate users who would be affected during an attack - ie. while the throttling was activated - would be users without persistent login cookies who were logging in from an unknown location or with a dynamic IP. Those users would be unable to login until the throttling wore off (which could potentially take a while, if the attacker kept his botnet running despite the throttling).

To allow this small subset of users to squeeze through the otherwise-sealed cat door, even while bots were still hammering away at it, I would employ a 'backup' login form with a CAPTCHA. So that, when you display the "Sorry, but you can't login from this IP address at the moment" message, include a link that says "**secure backup login - HUMANS ONLY (*bots: no lying*)**". Joke aside, when they click that link, give them a reCAPTCHA-authenticated login form that bypasses the site-wide throttling. That way, IF they are human AND know the correct login+password (and are able to read CAPTCHAs), they will **never** be denied service, even if they are connecting from an unknown host and not using the autologin cookie.

Oh, and just to clarify: Since I do consider CAPTCHAs to be generally evil, the 'backup' login option would **only** appear *while throttling was active*.

There is no denying that a sustained attack like that would still constitute a form of DoS attack, but with the described system in place, it would only affect what I suspect to be a tiny subset of users, namely people who don't use the "remember me" cookie AND happen to be logging in while an attack is happening AND aren't logging in from any of their usual IPs AND who can't read CAPTCHAs. Only those who can say no to ALL of those criteria - specifically bots and *really unlucky* disabled people - will be turned away during a bot attack.

> ***EDIT:*** *Actully, I thought of a way to let even CAPTCHA-challenged users pass through during a 'lockdown': instead of, or as a supplement to, the backup CAPTCHA login, provide the user with an option to have a single-use, user-specific lockdown code sent to his email, that he can then use to bypass the throttling. This definitely crosses over my 'annoyance' threshold, but since it's only used as a* ***last resort*** *for a tiny subset of users, and since it still beats being locked out of your account, it would be acceptable.*

(Also, note that **none** of this happens if the attack is any less sophisticated than the nasty distributed version I've described here. If the attack is coming from just a few IPs or only hitting a few usernames, it will be thwarted much earlier, and with ***no*** site-wide consequences)

So, that is the countermeasure I will be implementing in my auth library, once I'm convinced that it's sound and that there isn't a much simpler solution that I've missed.

The fact is, there are so many subtle ways to do things wrong in security, and I'm not above making false assumptions or hopelessly flawed logic. So please, any and all feedback, criticism and improvements, subtleties etc. are highly appreciated.

Share Improve this answer Follow

edited Nov 18, 2021 at 20:48
TylerH
21.2k ● 76 ● 79 ● 110

answered Jan 27, 2009 at 7:17
Jens Roland
27.8k ● 15 ● 84 ● 105

1 Maybe you could generate a 'special' password for each user that could use if in lock-down mode (and they're connecting from new IP etc), that special password being sufficiently complicated that not possible to brute-force? – Douglas Leeder Jan 27, 2009 at 18:44

1 That could work, but only if the users remember those passwords even if they haven't used them before (these types of attack aren't commonplace, and no botmaster worth his salt would bother keeping one running for long after being throttled). The risk is too great that they simply couldn't remember. – Jens Roland Jan 28, 2009 at 0:41

2 However, one method that could definitely work, is to provide a 'send me a lockdown code' link to those users, allowing them to get an email containing a single-use, user-specific token that would allow them to login, bypassing the throttling. – Jens Roland Jan 28, 2009 at 0:43

1 @Abtin: Good idea, except that would be 'entering the arms race' -- ie. starting a 'who can outsmart whom' with the people who create password lists for dictionary attacks. I

think a better way would be to enforce a strong password policy so there *are* no weak passwords – Jens Roland Jan 29, 2009 at 5:02

1 @OrestisP.: You are missing the point of the distributed attack - # of invalid attempts from each IP is minimal, so per-IP blocking can't work. Also, the question specifically describes an automated brute force attack, so 1) the attacker is not human, but rather a botnet of zombie machines (who can't use the captcha login); and 2) the brute force nature of the attack requires a very high number of login attempts to ensure success, which means farming the captcha solving out to a sweat shop somewhere isn't feasible (albeit possible if the attacker is well-funded and determined enough). – Jens Roland Sep 28, 2013 at 17:17

17

A few simple steps:

Blacklist certain common usernames, and use them as a honeypot. Admin, guest, etc... Don't let anyone create accounts with these names, so if someone does try to log them in you know it's someone doing something they shouldn't.

Make sure anyone who has real power on the site has a secure password. Require admins/ moderators to have longer passwords with a mix of letters, numbers and symbols. Reject trivially simple passwords from regular users with an explanation.

One of the simplest things you can do is tell people when someone tried to log into their account, and give them a link to report the incident if it wasn't them. A simple message when they log in like "Someone tried to log into your account at 4:20AM Wednesday blah blah. Click here if this wasn't you." It lets you keep some statistics on attacks. You can step up monitoring and security measures if you see that there's a sudden increase in fraudulent accesses.

Share Improve this answer Follow

answered Jan 28, 2009 at 1:04

patros
7,799 ● 3 ● 29 ● 37

Fine thoughts. I was definitely planning to implementing an automatic password policy that varies dynamically with the user's privilege level. The honeypot idea might work for some types of attack, but if the attack is distributed, blocking the IPs that fall for it won't be effective. – Jens Roland Jan 28, 2009 at 4:10

With respect to the 'Last attempted login time', that is a good strategy for power users (which I bet is why SO does it), but it has two weaknesses: (a) it doesn't address the problem of intrusion, it only reports that it may have happened, and (b), most user's just don't remember/care – Jens Roland Jan 28, 2009 at 4:14

---

1 Yup, the honeypot and user reporting are more about information gathering. They may provide some valuable metrics to let you know if/when a slow brute force attack is happening. – patros Jan 28, 2009 at 8:10

---

2 For the honeypot, wouldn't treating *any* non-existent username as suspicious be better than just using a fixed list of known-bad usernames? You'd want to avoid locking out users who mistyped their username and didn't notice the typo while retrying their password several times, but I still think there are ways it could be valuable. You could even avoid

some "false positives" by building a large bloom filter or similar data structure with variants of valid usernames, first names, last names, email names, etc. as users are added.
– R.. GitHub STOP HELPING ICE Jan 23, 2013 at 4:42

11

If I understand the MO of brute force attacks properly, then one or more usernames are tried continuously.

There are two suggestions which I don't think I've seen yet here:

- I always thought that the standard practice was to have a short delay (a second or so) after each wrong login for every user. This deters brute-force, but I don't know how long a one second delay would keep a dictionary attack at bay. (dictionary of 10,000 words == 10,000 seconds == about 3 hours. Hmm. Not good enough.)
- instead of a site-wide slow down, why not a user-name throttle. The throttle becomes increasingly harsh with each wrong attempt (up to a limit, I guess so the real user can still login)

Edit: In response to comments on a username throttle: this is a username specific throttle without regard to the source of the attack.

If the username is throttled, then even a coordinated username attack (multi IP, single guess per IP, same username) would be caught. Individual usernames are protected by the throttle, even if the attackers are free to try another user/pass during the timeout.

From an attackers point of view, during the timeout you may be able to take a first time guess at 100 passwords, and quickly discover one wrong password per account. You may only be able to make a 50 second guesses for the same time period.

From a user account point of view, it still takes the same average number of guesses to break the password, even if the guesses are coming from multiple sources.

For the attackers, at best, it will be the same effort to break 100 accounts as it would 1 account, but since you're not throttling on a site wide basis, you can ramp up the throttle quite quickly.

Extra refinements:

- detect IPs that are guessing multiple accounts - 408 Request Timeout
- detect IPs that are guessing the same account - 408 Request Timeout after a large (say 100) number of guesses.

UI ideas (may not be suitable in this context), which may also refine the above:

- if you are in control of the password setting, then showing the user how strong their password is encourages them to pick a better one.

- if you are in control of the login *page*, after a small (say 10) number of guesses of a single username, offer a CAPTCHA.

Share Improve this answer Follow

edited Jan 26, 2009 at 23:34

answered Jan 26, 2009 at 15:44

jamesh
20.1k ● 14 ● 58 ● 96

A username throttle plus an IP throttle is fine against fixed-username or fixed-IP attacks, and they do make traditional dictionary attacks infeasible. But if the attacker constantly changes usernames, he will slip by without triggering a username throttle. That's what I want to counter
– Jens Roland Jan 26, 2009 at 21:23

3 Thanks for the edit, jamesh. Now we're talking. I love the idea of the 408. However, even with strict username throttling, a botnet attacking multiple users would still work. And checking the top 5000 passwords against one user is LESS likely to succeed than checking THE top 1 password on 5000 users
– Jens Roland Jan 27, 2009 at 2:43

1 Nothing like the birthday paradox. In a large group, many will use insecure passwords, and one is likely to use any given popular one. There will also be a fair number of people like me who aren't going to be caught by such an attack.
– David Thornley Jan 28, 2009 at 22:57

3 Actually, I may have to re-check the math on my previous statement. Once you have ruled out the top N most common passwords, the probability of the user having password # (N+1) may increase enough to even out the difference.

Although the curve is probably steep enough for that not to be the case – Jens Roland Jan 30, 2009 at 6:35

10

There are three factors of authentication:

1. A user **knows** something (ie, a password)
2. A user **has** something (ie, a key fob)
3. A user **is** something (ie, retina scan)

Usually, websites only enforce policy #1. Even most banks only enforce policy 1. They instead rely on a "knows something else" approach to two-factor authentication. (IE: A user knows their password and their mother's maiden name.) If you are able, a way to add in a second factor of authentication is not too difficult.

If you can generate around 256 characters of randomness, you could structure that in a 16×16 table, and then ask the user to give you the value in the table of cell A-14, for example. When a user signs up or changes their password, give them the table and tell them to print it off and save it.

The difficulty with that approach is that when a user forgets their password, as they will, you can't just offer the standard "answer this question and put in a new password", since that's vulnerable to brute-force as well. Also, you can't reset it and email them a new one, since their email could be compromised as well. (See: Makeuseof.com and their stolen domain.)

Another idea (which involves kittens), is what BOA calls SiteKey (I believe they trademarked the name). Briefly, you have the user upload an image when they register, and when they attempt to login, ask them to pick their image out of 8 or 15 (or more) random ones. So, if a user uploads a picture of their kitten, theoretically only they know exactly which picture is theirs out of all the other kittens (or flowers or whatever). The only real vunerability this approach has is the man-in-the-middle attack.

One more idea (no kittens though), is to track IPs that users access the system with, and require them to perform additional authentication (captcha , pick a kitty, pick a key from this table) when they log in from an address they haven't before. Also, similar to GMail, allow the user to view where they have logged in from recently.

Edit, New Idea:

Another way of validating login attempts is to check whether or not the user has come from your login page. You can't check referrers, since they can be easily faked. What you need is to set a key in the _SESSION var when the user views the login page, and then check to make sure that key exists when they submit their login information. If bot does not submit from the login page, it will not be able to login. You can also facilitate this by involving javascript in the process, either by using it to set a cookie, or adding some information to the form after it has loaded. Or, you can split the form up into two different submits (ie, the user enters their username, submits, then on a new page enters their password and submit again.)

The key, in this case, is the most important aspect. A common method of generating them is some combination of the user's data, their IP, and the time it was submitted.

Share Improve this answer

Follow

edited May 9, 2014 at 12:31

Balaji Kandasamy
4,506 ● 10 ● 41 ● 58

answered Jan 27, 2009 at 0:15

davethegr8
11.6k ● 5 ● 40 ● 61

I'm sure there is more to it, but if the SiteKey idea is exactly what you mentioned, an attacker doesn't have to be a MITM, he can just run two or three login attempts for that user, and pick the image that is repeating among the random ones. Even if the set of 8-15 pictures is static for user X,
– Jens Roland Jan 27, 2009 at 2:50

(continued) it probably wouldn't be too difficult to pick the correct one, since people tend to pick predictable types of images (even images from their own Flickr albums!)
– Jens Roland Jan 27, 2009 at 2:51

---

3 Yeah, I thought of the point you brought up last night after I had gone home. I think the way to fix that is: When a user logs in and provides a correct password, display their image and some number of other random ones. When they do not provide the correct password, show some number of random
– davethegr8 Jan 27, 2009 at 17:09

---

1 images + 1, which may or may not include their own image. Also, I had another idea, see the edit in the post. But yeah,

these ideas are kinda difficult/complicated. – davethegr8 Jan 27, 2009 at 17:11

1 That "could" work, but I see a couple problems. What happens if the photo owner removes the image? How can you be sure that images returned won't be offensive to your user? How does a user remember where they clicked? (It seems difficult to forget) – davethegr8 Jan 29, 2009 at 22:53

7

I had previously answered a very similar question over at How can I throttle user login attempts in PHP. I'll reiterate the proposed solution here as I believe many of you will find it informational and useful to see some actual code. Please bare in mind that using a CAPTCHA might not be the best solution due to the increasingly accurate algorithms being used in CAPTCHA busters nowadays:

You cannot simply prevent DoS attacks by chaining throttling down to a single IP or username. Hell, you can't even really prevent rapid-fire login attempts using this method.

Why? *Because the attack can span multiple IPs and user accounts for the sake of bypassing your throttling attempts.*

I have seen posted elsewhere that ideally you should be tracking all failed login attempts across the site and associating them to a timestamp, perhaps:

```
CREATE TABLE failed_logins(
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT
PRIMARY KEY,
    username VARCHAR(16) NOT NULL,
    ip_address INT(11) UNSIGNED NOT NULL,
    attempted DATETIME NOT NULL
) engine=InnoDB charset=UTF8;
```

Decide on certain delays based on the *overall* number of failed logins in a given amount of time. You should base this on statistical data pulled from your `failed_logins` table as it will *change over time* based on the number of users and how many of them can recall (and type) their password.

```
10 failed attempts = 1 second
20 failed attempts = 2 seconds
30 failed attempts = reCaptcha
```

Query the table on every failed login attempt to find the number of failed logins for a given period of time, say 15 minutes:

```
SELECT COUNT(1) AS failed FROM failed_logins WHERE
attempted > DATE_SUB(NOW(), INTERVAL 15 minute);
```

If the number of attempts over the given period of time is over your limit, either enforce throttling or force all user's to use a captcha (i.e. reCaptcha) until the number of failed attempts over the given time period is less than the threshold.

```
// array of throttling
$throttle = array(10 => 1, 20 => 2, 30 =>
'recaptcha');

// assume query result of $sql is stored in $row
$sql = 'SELECT MAX(attempted) AS attempted FROM
failed_logins';
$latest_attempt = (int) date('U',
strtotime($row['attempted']));
// get the number of failed attempts
$sql = 'SELECT COUNT(1) AS failed FROM
failed_logins WHERE attempted > DATE_SUB(NOW(),
INTERVAL 15 minute)';
// assume the number of failed attempts was stored
in $failed_attempts
krsort($throttle);
foreach ($throttle as $attempts => $delay) {
    if ($failed_attempts > $attempts) {
        // we need to throttle based on delay
        if (is_numeric($delay)) {
            $remaining_delay = time() -
$latest_attempt - $delay;
            // output remaining delay
            echo 'You must wait ' .
$remaining_delay . ' seconds before your next
login attempt';
        } else {
            // code to display recaptcha on login
form goes here
        }
        break;
    }
}
```

Using reCaptcha at a certain threshold would ensure that an attack from multiple fronts would be *minimized* and normal site users would not experience a significant delay for legitimate failed login attempts. I can't gaurantee prevention, as it's already been expanded upon that CAPTCHA's can be busted. There are alternative solutions, perhaps a variant of "Name this animal", which could work quite well as a substitute.

Share Improve this answer Follow

edited May 23, 2017 at 11:46

Community Bot
1 ● 1

answered Jul 19, 2010 at 11:21

Corey Ballou
43.4k ● 8 ● 65 ● 77

6

I have to ask whether you've done cost-benefit analysis of this problem; it sounds like you're trying to protect yourself from an attacker who has enough web presence to guess a number of passwords, sending maybe 3-5 requests per IP (since you've dismissed IP throttling). How much (roughly) would that kind of attack cost? Is it more expensive than the value of the accounts you're trying to protect? How many gargantuan botnets want what you've got?

The answer might be no -- but if it is, I hope you're getting help from a security professional of some sort; programming skill (and StackOverflow score) do not correlate strongly to security know-how.

Share Improve this answer Follow

edited Jan 29, 2009 at 21:13

answered Jan 28, 2009 at 22:40

ojrac
13.4k ● 6 ● 39 ● 40

(You mean to say if the answer is 'no' -- ie. that the expense of a botnet attack is NOT too high in relation to the accounts) – Jens Roland Jan 29, 2009 at 5:14

But anyway, you bring up an important point. For my own uses, I don't expect any botnet operator to care in the least, but I am releasing the source code for anyone who would like decent security for their web app, and I can't know what others might be trying to protect, or who their enemies are – Jens Roland Jan 29, 2009 at 5:16

It won't be guarding national secrets no matter what (official systems need special certification, and I'm fairly sure nothing built on PHP can qualify), but all web applications need secure auth, so if I'm releasing this, it'd be incredibly irresponsible not to use best practices wherever I can – Jens Roland Jan 29, 2009 at 5:24

1 So my short answer is: I am building this because 99.9% of web sites and apps out there have appalling security (even in the big leagues: AOL, Twitter, Myspace have all been compromised before), and in most cases because they're using shoddy auth libraries. – Jens Roland Jan 29, 2009 at 5:32

Also, read the paper "To Catch A Predator" by Niels Provos et al. from the 2008 USENIX proceedings (link: usenix.org/events/sec08/tech/small.html) It is an eye opener: 2 months, one honeypot: 368,000 attacks from almost 30.000 distinct IPs, coming from more than 5,600 botnets! – Jens Roland Jan 29, 2009 at 15:22

5

To summarize Jens' scheme into a pseudo state transition diagram/rulebase:

1. user + password -> entry
2. user + !password -> denied
3. user + known_IP(user) -> front door, `// never throttle`
4. user + unknown_IP(user) -> catflap
5. (#denied > n) via catflaps(site) -> throttle catflaps(site) `// slow the bots`
6. catflap + throttle + password + captcha -> entry `// humans still welcome`
7. catflap + throttle + password + !captcha -> denied `// a correct guess from a bot`

Observations:

- Never throttle the front door. The Elbonian state police have your computer, in your house, but are unable to interrogate you. Brute force is a viable approach from your computer.
- If you provide a "Forgetten your password?" link, then your email account becomes part of the attack surface.

These observations cover a different type of attack to the ones you are trying to counter.

Share Improve this answer Follow

edited Jan 29, 2009 at 18:31

answered Jan 28, 2009 at 22:14

jamesh
20.1k ● 14 ● 58 ● 96

Absolutely the email account is part of the attack surface. I have a set of upper-bound assumptions on the security my strategy will provide, and the lowest bound is the user's own email security. If an attacker breaches a user's email, all bets are off. – Jens Roland Jan 29, 2009 at 5:07

Also, I think your state transition diagram needs a couple of details: #3 and #4 should include password; #1 and #2 should include known_IP(user) since a login always has either known or unknown IP; and #6 is 'entry despite throttle' – Jens Roland Jan 29, 2009 at 5:11

4

Looks like you are trying to defend against [slow distributed brute force](). Not that much you can do about it. We are using a PKI and no password logins. It helps, but if your clients chance workstations every once in a while, this is not very applicable.

Share Improve this answer Follow

answered Jan 26, 2009 at 10:04

raupach
3,102 ● 23 ● 30

Actually fast brute force too. I was hoping to be somewhat lenient with fixed-user brute force (throttling just 20 seconds), but on a site with 50k users, that would make variable-user *fast* brute force possible (assuming 20+ seconds to cycle through the users). And that, as they say, would suck.. – Jens Roland Jan 26, 2009 at 10:19

Well fast brute force from a single host use iptables or whatever firewall u use. – raupach Jan 26, 2009 at 14:51

I was referring to distributed fast brute force. It's rare but it's potentially very nasty – Jens Roland Jan 27, 2009 at 2:44

3

Disclaimer: I work for a two-factor company, but am not here to plug it. Here're some observations.

Cookies can be stolen with XSS and browser vulns. Users commonly change browsers or clear their cookies.

Source IP addresses are simultaneously dynamically variable and spoofable.

Captcha is useful, but doesn't authenticate a specific human.

Multiple methods can be combined successfully, but good taste is certainly in order.

Password complexity is good, anything password-based critically depends on passwords having sufficient entropy. IMHO, a strong password written down in a secure physical location is better than a weak password in memory. People know how to evaluate the security of paper documents much better than they know how to figure the effective entropy in their dog's name when used as a password for three different websites. Consider giving users the ability to print out a big or small page full of one-time use pass codes.

Security questions like "what was your high-school mascot" are mostly another lousy form of "something you know", most of them are easily guessable or outright in the public domain.

As you noted, throttling back failed login attempts is a trade-off between preventing brute-force attacks and ease of DoSing an account. Aggressive lockout policies may reflect a lack of confidence in password entropy.

I personally don't see the the benefit to enforcing password expiration on a website anyway. Attacker gets your password once, he can change it then and comply with that policy just as easily as you can. Perhaps one benefit is that the user might notice sooner if the attacker changes the account password. Even better would be if the the user were somehow notified before the attacker gained access. Messages like "N failed attempts since last login" are useful in this respect.

The best security comes from a second factor of authentication which is out-of-band relative to the first. Like you said, hardware tokens in the "something you have" are great, but many (not all) have real admin overhead associated with their distribution. I don't know of any biometric "something you are" solutions good for websites. Some two-factor solutions work with openid providers, some have PHP/Perl/Python SDKs.

Share Improve this answer

Follow

answered Jul 29, 2009 at 5:33

Marsh Ray
2,885 ● 22 ● 20

All excellent points - I couldn't agree more. The point about cookie insecurity is very valid, but without a second factor of physical tokens or one-time passwords (distributed over a secure line) you really can't protect against a vulnerable endpoint. If the user's box/browser is compromised, so are his logins. – Jens Roland Jul 13, 2010 at 8:09

1

My highest recommendation is to simply make sure that you **keep users informed** of bad login attempts to their accounts-- Users will likely take the strength of their password much more seriously if they are presented with evidence that somebody is actually trying to get into their account.

I actually caught somebody that hacked into my brother's myspace account because they had tried to get into the gmail account I setup for him and used the 'reset my password by email' feature... which went to my inbox.

Share Improve this answer

Follow

answered Jun 10, 2009 at 15:59

nvuono
3,363 ● 27 ● 27

1

1. What about requiring a one-time-password before entering their normal password? That would make it

very obvious that someone was attacking before they got many opportunities to guess the main password?

2. Keep a global count/rate of login failures - this is the indicator for an attack - during an attack be stricter about login failures e.g. ban IPs more rapidly.

Share Improve this answer Follow

edited Jan 27, 2011 at 7:08
Toon Krijthe
53.4k ● 38 ● 149 ● 202

answered Jan 26, 2009 at 10:37
Douglas Leeder
53.3k ● 9 ● 99 ● 138

1) How would you implement a one-time password on an unsecure, unauthenticated line? In other words, when does the user set these one-time passwords? 2) Yes, that's the gist of #4 on my list, the sitewide limit on failed attempts. The downside is the DoS opportunity it opens. – Jens Roland Jan 26, 2009 at 10:59

0

I don't believe there is a perfect answer but I would be inclined to approach it on a basis of trying to confound the robots if an attack is sensed.

Off the top of my mind:

Switch to an alternate login screen. It has multiple username and password blanks which really do appear but only one of them is in the right place. The field names are *RANDOM*--a session key is sent along with the login screen, the server can then find out what fields are what. Succeed or fail it's then discarded so you can't try a replay attack--if you reject the password they get a new session ID.

Any form that is submitted with data in a wrong field is assumed to be from a robot--the login fails, period, and that IP is throttled. Make sure the random field names never match the legit field names so someone using something that remembers passwords isn't mislead.

Next, how about a different sort of captcha: You have a series of questions that won't cause problems for a human. However, they are *NOT* random. When the attack starts everyone is given question #1. After an hour question #1 is discarded, never to be used again and everyone gets question #2 and so on.

The attacker can't probe to download the database to put into his robot because of the disposable nature of the questions. He has to send new instructions out to his botnet within an hour to have any ability to do anything.

Share Improve this answer Follow

answered Jan 27, 2009 at 1:17

Loren Pechtel
9,073 ● 4 ● 35 ● 46

The alternate login screen sounds like it would confuse humans more than machines, frankly. We are of course assuming that the attacker would have checked our security measures beforehand. He could have easily tweaked his scraper to find the correctly-placed fields. – Jens Roland Jan 27, 2009 at 2:59

The human-checking questions have been done before, and it's not very effective. For a human botnet operator to answer one question per hour (after which the new answer would propagate to the bots) during an attack would be quite feasible. – Jens Roland Jan 27, 2009 at 3:02

You're missing the point. The attacker can't check the in advance because it only shows the extra defenses when an attack shows up. – Loren Pechtel Jan 27, 2009 at 5:34

Sure, the human could see what the question was--but he has to communicate that to all his bots. That's a communications path that makes it easier to bring down the botnet. – Loren Pechtel Jan 27, 2009 at 5:35

I don't think I am missing the point. I don't mean he would have run an attack previously to check our security measures, I mean he would have read this thread and checked the (open) source code to check for weknesses :) – Jens Roland Jan 27, 2009 at 7:20

0

Since several folks included CAPTCHA as a fallback human mechanism, I'm adding an earlier StackOverflow question and thread on CAPTCHA's effectiveness.

Has reCaptcha been cracked / hacked / OCR’d / defeated / broken?

Using CAPTCHA doesn't limit improvements from your throttling and other suggestions, but I think the number of answers that include CAPTCHA as a fallback should consider the human-based methods available to people looking to break security.

Share Improve this answer Follow

edited May 23, 2017 at 11:54
Community Bot
1 ● 1

answered Mar 5, 2009 at 22:25
Matthew Glidden
382 ● 2 ● 6 ● 15

0

You could also throttle based on the strength of a users password.

When a user registers or changes their password you calculate a strength rating for their password, say between 1 and 10.

Something like "password" scores a 1 whereas "c6eqapRepe7et*Awr@ch" might score a 9 or 10 and the higher the score the longer it takes for throttling to kick in.

Share Improve this answer Follow

answered Jan 27, 2011 at 4:11
Joseph W
187 ● 7

---

3 I understand the idea, but that would indirectly leak information about the password, letting an attacker know whether a password is worth hacking or not. That may seem

a bit theoretical, but many users reuse passwords, so if I want to break into Strong_Throttling_Website.com I can simply attack (privileged) accounts at random until I find a user, 'Freddy', who has a weak password (i.e. early throttling), then go to Less_Secure_Website.edu and do an easy dictionary attack on Freddy's account there. It's a little involved, but certainly doable in practice. – Jens Roland Jan 27, 2011 at 13:41

0

The first answer I've usually heard when asking this question is to change ports, but forget about that and just disable IPv4. If you only allow clients from IPv6 networks you'r no longer pray for simple network scanning and attackers will resort to DNS lookups. Don't run on the same address as your Apache(AAAA)/Sendmail(MX->AAAA)/what have you given out to everybody(AAAA). Make sure your zone can't be xferd, wait you'r allowing your zone to be downloaded by anybody?

If the bots find your server setup new hostnames, just prepend some gibberish to your hostnames, and change your address. Leave the old names and even setup **honeypot names for the bot net to timeout on.

** Test your reverse(PTR) records(under ip6.arpa.) to see if they can be used to zero in on /4's that have records VS /4s that don't. I.E. Typically ip6.arpa would have ~32 "."s in an address but trying with the last few missing might elude the network blocks that have records VS others that don't. If you take that further it becomes possible to skip large portions of the address space.

In the worst case users will have to setup an IPv6 tunnel, it's not like they'd have to go as far as VPNing into a DMZ... Though one wonders why that's not the first option.

Also Kerberos is cool, but IMHO LDAP blows(What's technically wrong with NISPlus? I've read that Sun decided that users wanted LDAP and because of this they dropped NIS+). Kerberos does work fine without LDAP or NIS, just have to manage users on a host by host basis. Using Kerberos gives you an easy to use, if not automated, PKI.

Share Improve this answer Follow

answered Jan 12, 2015 at 7:55

Mike Mestnik
323 ● 6 ● 14

0

Bit late here but I was thinking, assuming a hard case - the attacker uses a lot of random IPs, random user names and a random password selected from say a list of the 10,000 most popular.

One thing you could do, especially if the system seems under attack in that there are a lot of wrong password attempts on the system and especially if the password is low entropy is to ask a secondary question like what are your parents first names, for example. If an attacker hits a million accounts trying the password 'password1' there's a good chance they'll get a lot but their odds of also getting the names right would reduce successes dramatically.

Share Improve this answer

Follow

answered Nov 23, 2015 at 23:52

Tim 333

952 ● 1 ● 8 ● 9

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.