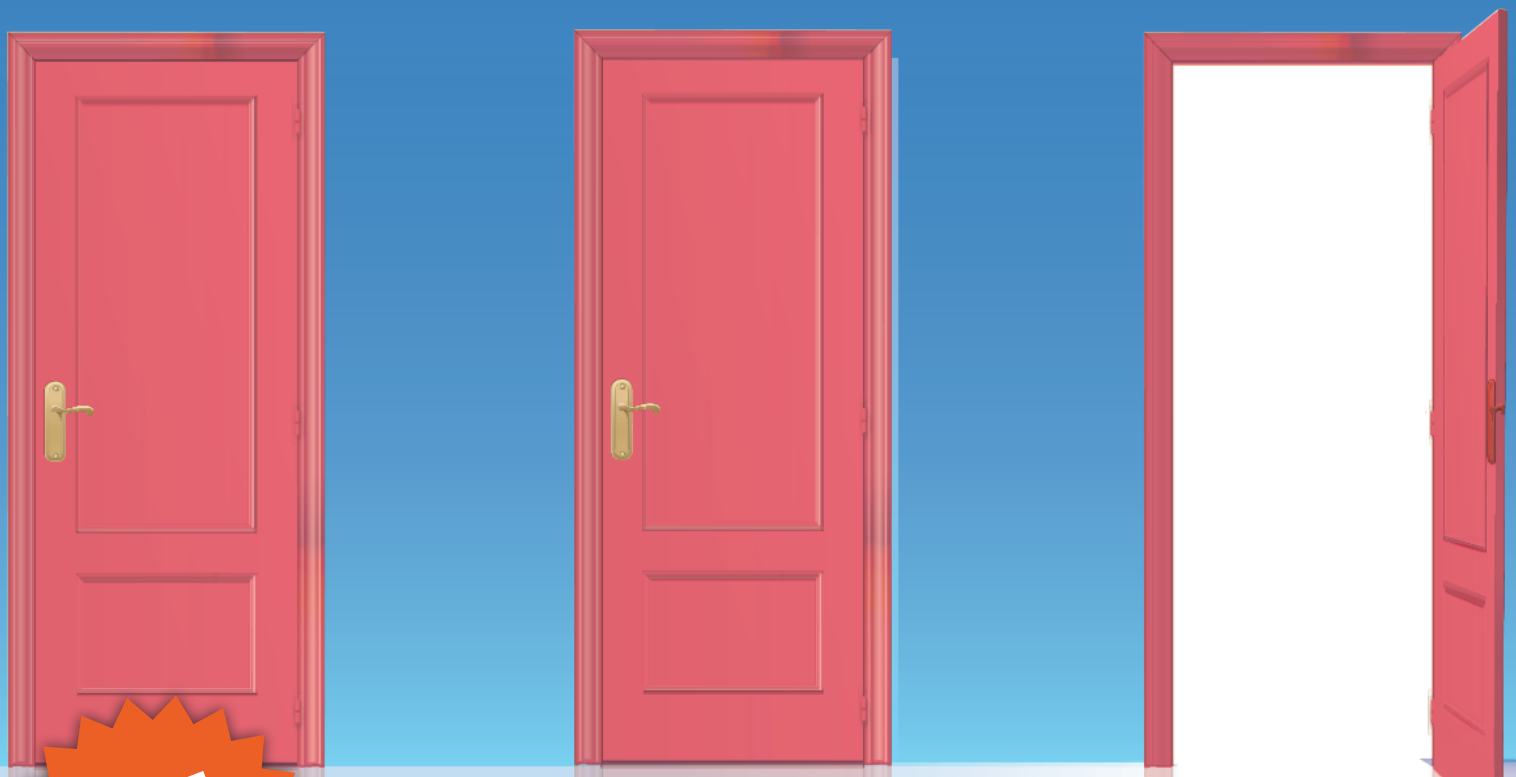
www.phparch.com

January 2016
VOLUME 15 - ISSUE 1

php[architect]

APIs

Doorways to Your Apps



FREE Article!

Discovering API Platform

Build API's on Existing Web Applications with Apigility

ALSO INSIDE

Writing Better CLI Tools with Symfony Components

Raspberry Pi with PHP on Top

Leveling Up:
Finding the Solution to the Problem

Security Corner:
Passwords are Dead, Long Live Passwords!

Education Station:
Date and Time Handling with Carbon

Community Corner:
Interview with Jeffrey Carouth

finally{}:
Different Paths for the P's in LAMP

Passwords are Dead, Long Live Passwords!

Chris Cornutt

password	696969	753951	killer
12345	123123	chocolate	abcdef
12345678	batman	soccer	hannah
qwerty	trustno1	tigger	test
123456789	daniel	asdasd	alexander
1234	computer	jennifer	andrew
baseball	michael	jordan	222222
dragon	121212	abcd1234	joshua
football	charlie	trustno1	freedom
1234567	master	buster	samsung
monkey	superman	555555	asdfghj
letmein	qwertyuiop	liverpool	purple
abc123	112233	abc	ginger

Anyone who's been around web applications (or really any applications) that need to protect data or restrict access to only a certain group of users has experience with passwords. They're a de facto standard when it comes to protecting applications. Along with usernames (or email addresses, depending on the system), they're used to identify the end user and verify that they are who they claim to be. Unfortunately, there are many things wrong with them that make them one of the worst options in application protection. There are whole industries around removing or reinforcing passwords in applications, yet they're still a huge part of the security of most services out there. I'm getting ahead of myself, though. First off, let's review a little history of where passwords came from.

DisplayInfo()

Related URLs:

- PHP Password Hashing–http://php.net/book.password

Where Did Passwords Come from, Anyway?

The concept of a *password* or *passphrase* has been around about as long as there have been secrets to protect. Way back in history, this *something you know* was used for everything from sharing information between groups to allowing access to certain physical areas not open to everyone. Fast-forward to more recent times, and there's no shortage of movies and books out there in which spies use them to identify each other or messages are protected with a password only the intended recipient should know.

Passwords were first introduced into the world of computers in the early 1960s. A group at MIT in Massachusetts decided that they needed a way to segregate the time that people had to spend on the shared computing systems owned by the university. Passwords were used in their Compatible Time-Sharing System (CTSS). They even had the notion of protecting this password and not echoing it back out to the user as he or she typed. Incidentally, many Unix-based systems still do this, while most web applications use a *password* type form field that, though masking, still gives a visual indication of how long the password is.

Move forward another 10 years or so, and another password improvement came along in the form of hashed passwords. Robert Morris added this functionality to the origins of the operating system we know as Unix, which used a simpler version of the standard `crypt()` functionality to protect password contents. Even then they realized that having just a plain-text, human-readable password somewhere wasn't the best or most secure method for protecting valuable resources.

Since then, password storage methods and usage have evolved, but the heart of the usage is still the same. Passwords are still a single point of failure that's usually combined with a much more public piece of information, a username, to restrict access to portions of applications. This is the real key to the problem: they only offer a single point of protection that all too often is easily compromised, leaving the system wide open to attack.

Common Password Problems

I've already mentioned one of the major problems with passwords: the single point of failure they provide. But there are a few others that contribute to most of the password-centric vulnerabilities out there.

Password Reuse

We've all done it before. If you say you haven't, you're probably fibbing just a bit. There are so many services out there, and we're constantly signing up for more and more every day. With almost all of them using the same basic methods for authentication (again, the infamous username+password combo), it's very easy to slip into reusing the same password across multiple services.

Why is this a bad thing? Well, imagine you signed up at SuperAwesomeHosting.com with an email address for the username and a password that just happens to be the same as the one for your email account. The hosting company assures you that security is a top priority for its customers and that your information is 100% safe in its systems. One day an attacker stumbles over an unprotected development copy of the site and discovers an SQL injection vulnerability, and harvests all of the live user data—including your credentials. Now the attacker has the credentials not only for your SuperAwesomeHosting.com account but also for your email account. Think of everything they could do if they made the jump and tried to log in to your email!

As a developer, it's almost impossible to prevent this from happening, unfortunately. The only thing you can do is try to enforce good password policies and hope users don't shoot themselves in the foot.

Bad Passwords

When I give presentations about security at conferences and online training, there's one thing I say every single time: "People are terrible at passwords." As humans, there's a built-in desire to make things as simple as possible. This is the same habit that leads to password reuse. Unfortunately, it also makes us lazy about the passwords we come up with. We use things like our pet's name, the date we got married, or words right from the dictionary. Take a quick look through anyone's social media pages and you can find answers to most of these and just start guessing. Even worse, when it comes time to reset your password, most people will just tack on a `1` or `!` and call it good.

This is where password policies come in. These policies help guide users to create good passwords that will effectively protect not only their own account but also your service. There's one key thing to remember when setting up your policies: the phrase "at least." The real key to effective policies is to set them up so that you define minimum requirements and then let the user go wild from there. Sure, this can still lead to some pretty bad passwords, but there are plenty of tools out there that will measure the entropy of the password to ensure it's strong enough.

Bad Password Storage

I want to touch on one last topic that's a bit more developer-focused than the others in this (non-exhaustive) list. Users put their trust in you to keep their private information safe, including the passwords they provide. They perceive this as one of the keys to your having a secure system, so it makes sense that you'd protect this information accordingly. Unfortunately, we hear of service after service that was either storing their users' passwords in plain text or, while making an effort to "encrypt" them, only thought that an `md5` of the password value was enough.

If you're doing either if these things, stop right now and go fix your application. There are a lot of reasons that people give for storing passwords poorly, but **they're all invalid**.

Passwords should *always* be one-way hashed with a strong method prior to being stored. Any company that can send you a plain-text version of your password or that can read it back to you during a support call is without a doubt doing it wrong. There's no need for anyone other than the user themselves to know their password.

So how can I do this effectively in my application? Fortunately, it's been made super simple since PHP 5.5 with the password hashing functions. It's literally a two-line process to hash what the user gives you and verify if it's correct:

```
<?php
// To hash the password
$storeMe = password_hash($_POST['userInput'], PASSWORD_DEFAULT);

// To verify the password
if (password_verify($_POST['userPassword'], $storeMe) === true) {
   echo 'yay!';
}
```

This method currently uses an algorithm called *bcrypt* that rehashes the string provided a number of times based on the "cost" value. PHP's default cost is 10 unless you define it as an option.

The cost affects how difficult a password hash is to crack and is meant to increase as hardware becomes more capable.

And Finally

Passwords are flawed, there's no doubt about it. I mentioned tools and services that have come up around the password ecosystem and that want to help remediate some of the risk associated with using passwords—two-factor authentication and federated identity being two of the more popular options. These can do a lot to help improve the overall security stance of the application and prevent other problems from happening, but with a password at the core of it all, a lot of risk will still be involved.

If the only protection between your application and the outside world is a simple user-defined string of text, it might be time to rethink and reinforce your systems. Trust me, a password just doesn't cut it.

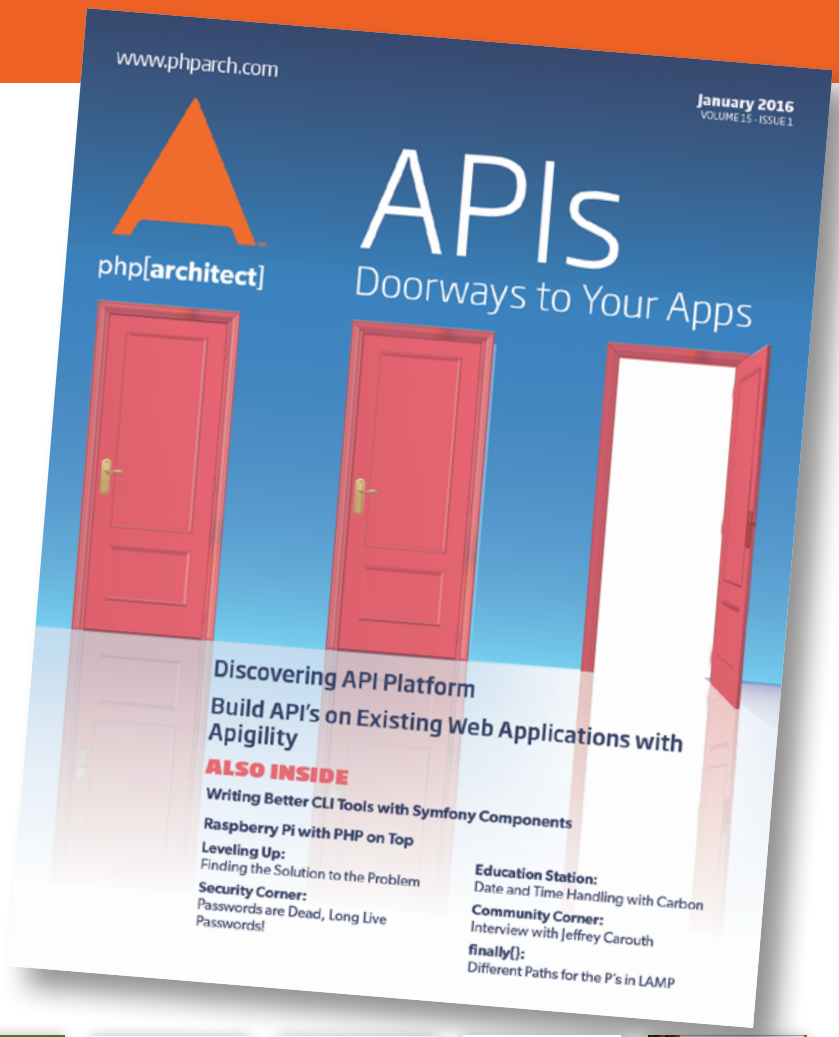
For the last 10+ years, **CHRIS** has been involved in the PHP community in one way or another. These days he's the Senior Editor of PHPDeveloper.org and lead author for Websec.io, a site dedicated to teaching developers about security and the Securing PHP ebook series. He's written for several PHP publications and has spoken at conferences in both the U.S. and Europe. He's also an organizer of the DallasPHP User Group and the Lone Star PHP Conference and works as an Application Security Engineer for Salesforce.

Twitter: @enygma

Want more articles like this one?

Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.



php[architect]

magazine

books

conferences

training

www.phparch.com

Get the complete issue for only $6!

We also offer digital and print+digital subscriptions starting at $49/year.