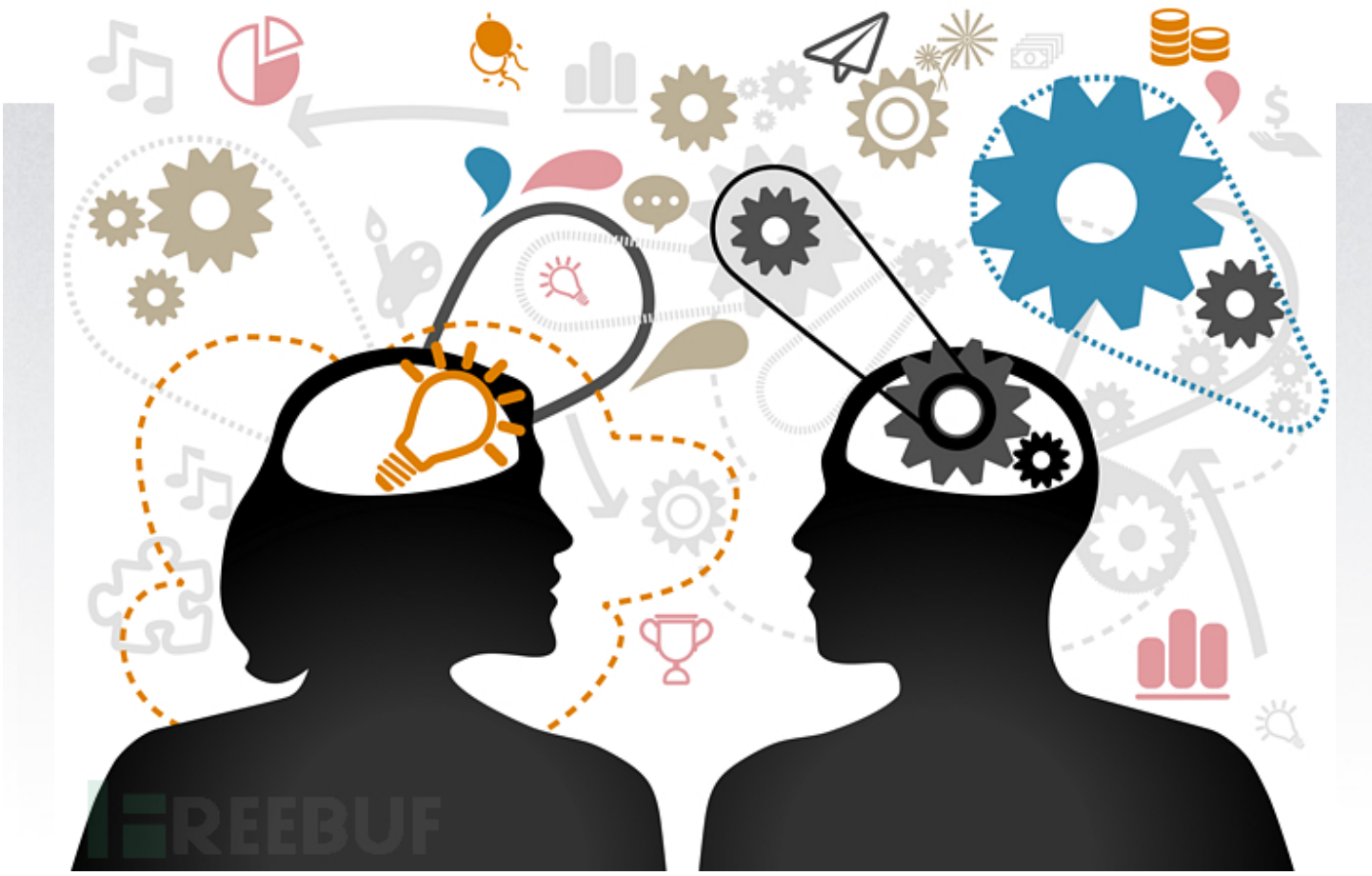


0×00 前言

首先如果你对密码学的概念以及使用并不熟悉，或者你正需要进行一些密码学的引导，那么我推荐你阅读一下这篇内容。

此前我们就曾明确的表示，即使是安全建议也应该有个保质期。因此和我们过去发布的大多数博客文章不同，上面那篇内容实际上是处在一种“随时更新”的状态：当对安全的需求变化以及新的攻击形式被发现时，我们都会做出相应的变更。

> **这里我们提出一个密码安全观点：不要存储明文密码，而是存储密码的哈希值。**

事实上现在，生成安全的密码哈希值非常简单。

但这里有个问题就是，你可能希望别人可以设置一个带密码的帐号，通过这个帐号和密码，别人可以登录到你的程序中，那么这一功能要怎样才能安全的实现？

而解决这个问题也很简单——使用libsodium。它可以为大多数语言提供一个安全的密码哈希API。在1.0.8版本之前它使用的是scrypt算法，但从1.0.9版本开始，它还会提供Argon2，这是从最近的哈希密码对比中精心挑选出来的一个算法。Libsodium会提供对大多数编程语言的绑定。

> Libsodium文件
>
> Libsodium源码

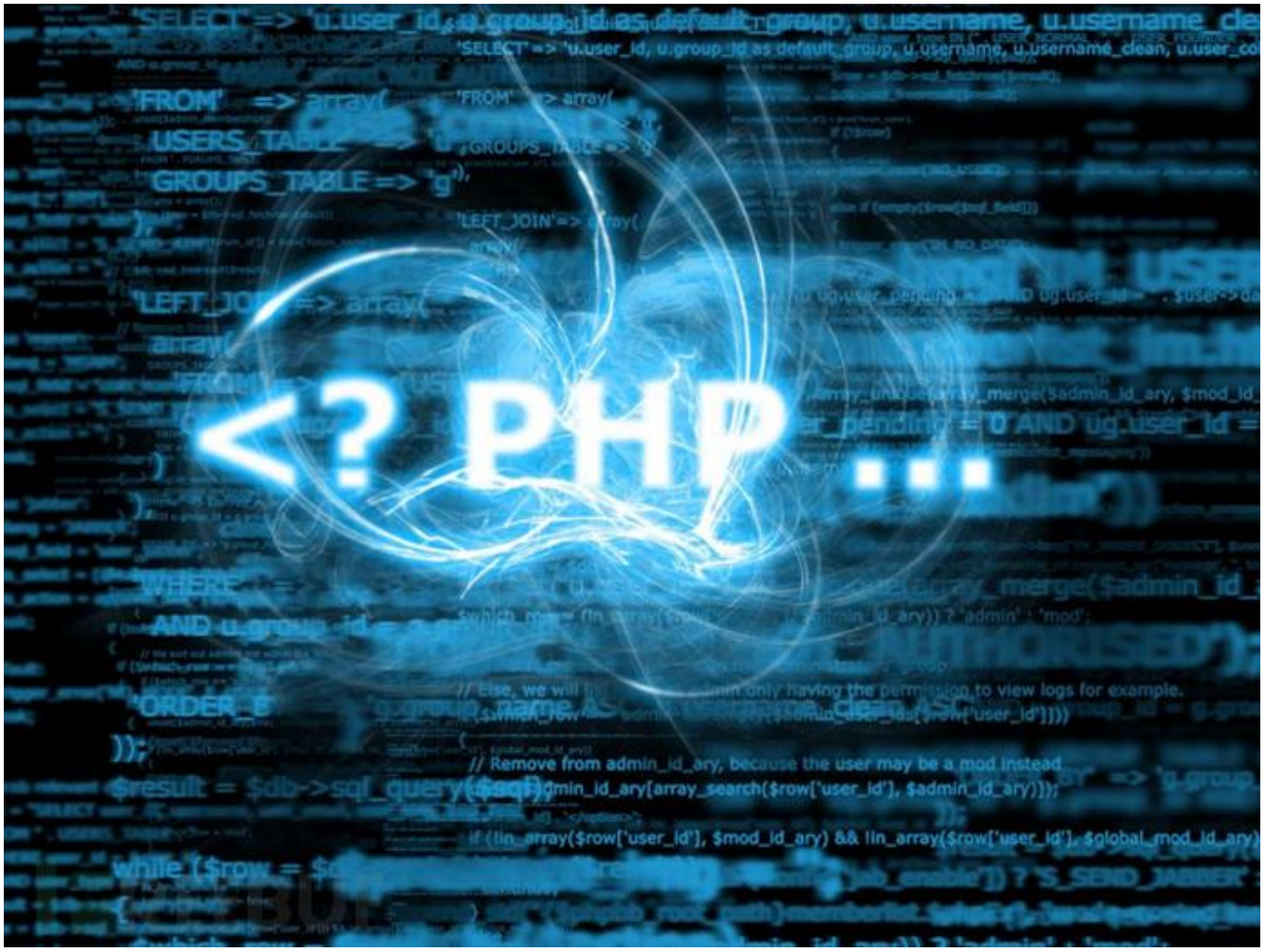
注意：这里公布了一个对Argon2i的攻击——Argon2通用密码哈希的变种表示。实际上它的影响并不严重，但它可能会导致一个新的变种（也许是Argon2x，因为它可能会使用XOR而不是覆盖内存来缓解这

些攻击）。

如果你因为种种原因无法在安装libsodium和你的要求之间进行调和的话，你还有其他的选择。在准备这篇博客时，我们的团队已经研究了多种编程语言的几个密码哈希库，下面就为大家利用示例代码介绍下。

目前可接受的密码哈希算法主要有：

> Argon2 密码哈希比赛的冠军
>
> bcrypt
>
> scrypt
>
> 以及密码哈希比赛的其他参赛算法 (Catena, Lyra2, Makwa, and yescrypt)
>
> PBKDF2 最糟糕的一种选择



0×01 PHP

PHP第一种方案

首先确认你使用的版本是否支持PHP。如果支持，那么PHP密码API将能够使用。如果不支持，那么可以尝试进行升级，如果还是不行，检查一下password_compat。

```
$hash = password_hash($userPassword, PASSWORD_DEFAULT);
```

昵称password_hash()使用的是加盐计算的哈希值。根据自己实际情况进行调整，如果硬件支持的话，使用最小绝对值10.12是很好的，其默认的10。

```
$hash = password_hash($userPassword, PASSWORD_DEFAULT, ['cost' => 12]);
```

事实上验证一个密码的哈希存储是很简单的：

```
if (password_verify($userPassword, $hash)) {
    // Login successful.
    if (password_needs_rehash($userPassword, PASSWORD_DEFAULT, ['cost' => 12])) {
        // Recalculate a new password_hash() and overwrite the one we stored previously
    }
}
```

在PHP7中，PASSWORD_DEFAULT仍然使用bcrypt。在未来的版本中，它可能会逐渐变为Argon2.

PHP第二种方案

如果你没办法使用libsodium（当然我们强烈建议你使用），你仍然可以用PHP中加密哈希，只需要通过来自PECL 的Dominic Black的Scrypt PHP 扩展即可。

```
#If you don't have PECL installed, get that first.
pecl install scrypt
echo "extension=scrypt.so" > /etc/php5/mods-available/scrypt.ini
php5enmod scrypt
```

接下来，把一个PHP包装捆绑在你的工程中。

```
# Hashing
$hash = \Password::hash($userProvidedPassword);
# Validation
if (\Password::check($userProvidedPassword, $hash)) {
       // Logged in successfully.
}
```

0×02 JAVA

JAVA第一种方案

在Java程序中进行安全的密码哈希，除了 libsodium还有jBCrypt，能够提供bcrypt密码哈希算法。

```
String hash = BCrypt.hashpw(userProvidedPassword, BCrypt.gensalt());
```

验证一个在Java中的bcrypt hash：

```
if (BCrypt.checkpw(userProvidedPassword, hash)) {
        // Login successful.
}
```

JAVA第二种方案

有一个java实现的scrypt，但它需要你指定参数，而不会提供一个默认值给你。

```
# Calculating a hash
int N = 16384;
int r = 8;
int p = 1;

String hashed = SCryptUtil.scrypt(passwd, N, r, p);

# Validating a hash
if (SCryptUtil.check(passwd, hashed)) {
        // Login successful
}
```

0×03 C# .NET

C#（.NET）第一种方案

我们推荐Martin Steel的BCrypt.NET fork而不是System.Security.Cryptography.Rfc2898DeriveBytes，它是PBKDF2-SHA1（我们不是说PBKDF2-SHA1不安全，但相对来说bcrypt更好一些。

```
// Calculating a hash
string hash = BCrypt.HashPassword(usersPassword, BCrypt.GenerateSalt());

// Validating a hash
if (BCrypt.Verify(usersPassword, hash)) {
       // Login successful
}
```

C#（.NET）第二种方案

这里同样有一个Scrypt 包 在NuGET中。

```
// This is necessary:
ScryptEncoder encoder = new ScryptEncoder();
// Calculating a hash
SecureString hashedPassword = encoder.Encode(usersPassword);
// Validating a hash

if (encoder.Compare(usersPassword, hashedPassword)) {
       // Login successful
}
```

0×04 Ruby

Ruby 第一种方案

按照笔者一如既往的规律，这里是一个对bcrypt密码哈希的 Ruby gem。

```
require "bcrypt"

# Calculating a hash
my_password = BCrypt::Password.create(usersPassword)
# Validating a hash
if my_password == usersPassword
   # Login successful
```

要注意的是截止本文发布，该库仍然没有遵循加密编码的最佳实践。因此在他们抽出时间打补丁前都需要考虑到这一点。

Ruby 第二种方案

在Ruby中进行Scrpyt哈希

这也是一个对bcrypt密码哈希的 Ruby gem。

```
require "scrypt"

# Calculating a hash
password = SCrypt::Password.create(usersPassword)
# Validating a hash
if password == usersPassword
   # Login successful
```

0×05 Python

Python 第一种方案

好了你猜的没错还是使用bcrypt python 包（ GitHub ）

```
import bcrypt
import hmac

# Calculating a hash

password = b"correct horse battery staple"
hashed = bcrypt.hashpw(password, bcrypt.gensalt())
# Validating a hash (don't use ==)
if (hmac.compare_digest(bcrypt.hashpw(password, hashed), hashed)):
        # Login successful
```

Python开发人员通常更喜欢passlib (Bitbucket)，尽管它的API命名并不正确。（ ” 加密” 而不是 “hash” ）：

```
from passlib.hash import bcrypt

# Calculating a hash
hash = bcrypt.encrypt(usersPassword, rounds=12)
```

```
# Validating a hash
if bcrypt.verify(usersPassword, hash):
        # Login successful
```

Python 第二种方案

目前我们发现除了libsodium以外只有[django-scrypt package](#).可以较为健全的实现。其他的我们还在寻找中。

0×06 Node.js

Node.js 第一种方案

在Node.js上bcrypt（[https://www.npmjs.com/package/bcrypt](#)）有两种安全的实现方式，尽管bcrypt看起来是首选。

在下面的例子中，我们大量的使用了Promises来简化错误处理：

```
var Promise = require("bluebird");
var bcrypt = Promise.promisifyAll(require("bcrypt"));

function addBcryptType(err) {
        // Compensate for `bcrypt` not using identifiable error types
        err.type = "bcryptError";
        throw err;
}
```

```
// Calculating a hash:
Promise.try(function() {
	return bcrypt.hashAsync(usersPassword, 10).catch(addBcryptType);
}).then(function(hash) {
	// Store hash in your password DB.
});

// Validating a hash:
// Load hash from your password DB.
Promise.try(function() {
	return bcrypt.compareAsync(usersPassword, hash).catch(addBcryptType);
}).then(function(valid) {
	if (valid) {
		// Login successful
	} else {
		// Login wrong
	}
});

// You would handle errors something like this, but only at the top-most point where it mak
Promise.try(function() {
	// Generate or compare a hash here
}).then(function(result) {
	// ... some other stuff ...
}).catch({type: "bcryptError"}, function(err) {
	// Something went wrong with bcrypt
});
```

Node.js 第二种方案

我们推荐 scrypt for humans，这是一个对开发人员很友好的 node-scrypt 包装器，非常容易使用。

```
var Promise = require("bluebird");
var scrypt = require("scrypt-for-humans");

// Calculating a hash:
Promise.try(function() {
   return scrypt.hash(usersPassword);
}).then(function(hash) {
   // Store hash for long term use
```

```
});

// Validating a hash:
Promise.try(function() {
   return scrypt.verifyHash(usersPassword, hash);
}).then(function() {
   // Login successful
}).catch(scrypt.PasswordError, function(err) {
   // Login failed
});
```

***原文地址：paragonie，东二门陈冠希/编译 0xroot后期整理，转载请注明来自FreeBuf黑客与极客（FreeBuf.COM）**