www.phparch.com

June 2017
VOLUME 16 - Issue 6

SECURE BY DESIGN

php[architect]

Nuclear Powered Software Security

Cybersecurity State of the Union

The Digital Speakeasy: Secure and Anonymous Access to Your Website

Free Article

ALSO INSIDE

Protocol Buffers

Education Station:
Creating Images on the Fly With Intervention Image

Community Corner:
Spurring Community with Adam Culp

Leveling Up:
Procrastination and Burnout

Artisanal:
Basic Relationships

finally{}:
Planning for the Future

php[architect]

CONTENTS

JUNE 2017
Volume 16 - Issue 6

SECURE BY DESIGN

Features

Columns

php[**architect**] is published twelve times a year by:
musketeers.me, LLC
201 Adams Avenue
Alexandria, VA 22301, USA

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

Cybersecurity State of the Union

Mark Niebergall

The cybersecurity landscape is continuously changing as new threats appear and attackers adapt. Data breaches, cyber attacks, identity theft, and scams show up regularly in the news and can have a significant negative impact to those affected by them. Keeping up with the latest cyber security trends, understanding the threats, and keeping applications secure takes an investment of time and effort.

In this article, we will review the current state of cybersecurity. Notable attacks will be highlighted, trends in attacks will be analyzed, strategies to secure projects will be identified, and PHP security-related features that can help increase application security will be covered.

Current State of Cybersecurity

Cybersecurity[1] is the "body of technologies, processes and practices designed to protect networks, computers, programs and data from attack, damage or unauthorized access" (1). Cybersecurity is vital to any organization using computers and software. Impacted organizations are found in every nation and across all industries.

In organizations of all sizes, one person may play multiple roles at the same time. As organizations mature and dedicate resources to IT, those roles become more defined with individuals focusing on specific functions. Keeping an organization secure, however, requires everyone to be mindful of security in their particular roles. Following best security practices and adapting to recent trends and growing threats can help strengthen the overall security posture of an organization, see *The alarming state of secure coding neglect*[2]. As a developer, writing secure code is critical to the success of an organization achieving cybersecurity goals.

A majority of attacks, roughly 80% (2), are from external actors. This can range from malicious hackers, script kiddies, nation states, collectives, and even customers. Attacks from external sources can come in many forms from phishing emails, to social engineering, or distributed denial-of-service (DDoS). Building out strong security at all layers of the OSI model and applying defense in depth is key to preventing these attacks from being successful.

Other attacks can come from internal sources or people associated with the organization, including disgruntled employees or someone with malicious intent. Being mindful that attacks can come not just from the outside but the inside as well can help determine where an organization may be insecure. Applying proper access controls, avoiding overreaching roles and permissions, and preventing escalation of privileges is a good place to start on strengthening security. Physical access controls should also be used to protect physical hardware (laptops, desktops, USB ports, servers, etc.) to prevent theft and unauthorized access from internal users. Hard drives used by employees to access resources should be encrypted to prevent data breaches in the case of theft or loss.

Attackers often seek to gain unauthorized access to the most valuable resources, especially data that may be of financial value to them. This can include data such as credit card information, government IDs, and other sensitive information. It is important to identify the most valuable assets to an organization and prioritize protection for those. There may be standards or regulations which apply to data used by an organization, including the worldwide Payment Card Industry Data Security Standard (PCI DSS) for credit cards. Other examples are the Health Insurance Portability and Accountability Act (HIPAA) in the United States for medical data, and Article 8 of the European Convention on Human Rights in the European Union for personal data. Be sure to research and follow local regulations covering data protection.

Notable Attacks

Over time, significant attacks have the ability to change the cybersecurity landscape by highlighting new or emerging threats. Some of the attacks which have had major impacts on cyber security are reviewed below.

KrebsOnSecurity.com: A major DDoS attack was launched in retaliation for exposure from investigations done by Brian Krebs. The DDoS was massive—665 gigabits/second—revealing an enormous botnet using the Internet of Things (IoT) which is becoming more and more prevalent. Securing IoT devices is often overlooked, and this attack highlighted the need to improve IoT security.

Target and Home Depot: Malware installed on point-of-sale (POS) systems resulted in the breach of millions of credit cards. These attacks directly resulted in an overhaul of how

1 *Cybersecurity:* http://phpa.me/what-is-cybersecurity

2 *The alarming state of secure coding neglect:* http://phpa.me/oreilly-state-secure-coding

retail stores secure credit card readers and transactions.

Yahoo!: Over 1 billion user accounts were breached, and the incidents were not made public for years. The company was being sold to Verizon when the breach was disclosed, which resulted in a $350 million USD loss. This took a big hit on high-level executives and changed how they prioritize cybersecurity and handle incidents.

Mt. Gox: In 2014, the company discovered it had lost around 850,000 bitcoins valued at $450,000,000 USD. This PHP shop had practiced poor software development practices, including no version control, inadequate testing, and improper change management. The company filed for bankruptcy and closed its doors, see *The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster*[3].

San Francisco Rail: A ransomware attack took ticket machines offline. The IT staff for the organization was able to restore systems because they were prepared.

These notable attacks have changed the way organizations prepare for and handle security incidents. Having plans to recover, backups, incident handling, and using secure coding practices can help prevent attacks from being successful and reduce the negative impact of successful attacks.

Trends in Attacks

As attackers adapt to changes in technology, we see a shift in the trends of attacks occurring. Two key reports help track current patterns in attacks. Every year Verizon puts out the Data Breach Investigations Report[4], which includes charts and details about sources of attacks, attack methods, targets, impacts to organizations, and more. Every few years OWASP releases an OWASP Top 10 list of the most critical web application security risks. Regularly reviewing these reports can help understand what the current landscape looks like and helps keep up with trends.

From the Verizon Data Breach Investigations report, we see a trend of increasing frequency of hacking, malware, and social engineering attacks. The primary reason is financial gain. For web applications, hacking is commonly used to breach systems by using stolen credentials and known security holes. Once the system has been breached, the attackers make off with the data they are after and try to prevent detection. With credit cards and identity theft, the information can be resold for money, and back doors left open can be sold off or used to continue collecting information until the breach is detected and closed off.

From the OWASP Top 10 list, we can see the most commonly exploited vulnerabilities used by attackers to achieve their goals. At the time of writing this article, the 2017 list is in release candidate stage and scheduled for release in July or August. The list is:

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Broken Access Control
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Insufficient Attack Protection
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Underprotected APIs

This list can be used in conjunction with an application review and audit to ensure those items are addressed. For example, an application susceptible to SQL injection attacks should be corrected urgently. Make sure authentication and sessions are handled correctly. Review the access control model and fix any issues found, including toxic roles, privilege escalation, and insufficient policies. Review system and application configurations to ensure secure settings are used and sensitive information is not included in version control systems. Resource requests for data and functionality should only be granted for authorized and authenticated subjects. Applications should use secure coding practices to prevent XSS and CSRF attacks. Libraries used should be updated to include the latest security patches. APIs should be locked down with authentication to ensure only authorized requests are allowed and fulfilled. While this may seem like a lot of areas to address, doing so will significantly improve the security in a web application.

Strategies to Secure Projects

Many approaches can be used to secure projects. Below are a few strategies to help secure web applications.

Threat Analysis

It is important to understand the most valuable assets to an organization. According to SANS, *An Overview of Threat and Risk Assessment*[5], a threat analysis tries to answer these questions:

1. What needs to be protected?
2. Who/What are the threats and vulnerabilities?
3. What are the implications if they were damaged or lost?
4. What is the value to the organization?
5. What can be done to minimize exposure to the loss or damage?

Reviewing and answering these questions can help direct where to start securing systems and data. If a particular application or data set is most important and an organization relies on them being secure to stay operating then that is a good place to start. Identifying specific areas to work on and to improve can help notch up the level of

3 *The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster:* http://phpa.me/wired-mt-gox

4 *Data Breach Investigations Report:* http://phpa.me/verizon-breach-report-2016

5 *An Overview of Threat and Risk Assessment:* http://phpa.me/sans-threat-assessment

security and help protect what data and systems matter the most.

Cryptography

Cryptography is the process of writing and reading secret messages. In the computer world, cryptography is applied to data. Data can be in transit and at rest. Symmetric-key, or secret-key, cryptography uses a shared private key. Asymmetric-key, or public-key, cryptography uses public and private keys to encrypt and decrypt data.

Many attacks target sensitive information, including credit cards, bank accounts, and government issued IDs. Cryptography provides a way to secure the data from attackers. By correctly encrypting sensitive information with strong encryption, data stolen during a breach is unusable to the attacker by making it much harder, if not impossible, to disclose sensitive data or personal information. It is critical to keep private keys and passwords confidential to prevent encrypted data from being decrypted by unauthorized parties. Properly encrypted data can significantly decrease the negative impact a breach has on an organization.

Patch Early, Patch Often

Keeping current with security patches can be a significant means of thwarting attacks. Attackers often exploit known vulnerabilities that have patches available. Applying patches as soon as possible after they are released makes your applications harder to break into. This includes updating libraries used, PHP versions, and other applicable patching to your environments. Having unit tests in place can help ensure updating versions doesn't break features and functionality within an application as these patches are applied.

PHP Security

PHP has a variety of features built-in to help improve the security of applications. Below we'll go through a few of these important features, basic syntax, when to use them, and how to leverage them to make your code more secure.

Hashes

PHP has a variety of hashing algorithms available to generate hashes. The function `hash_algos` returns an array of the available algorithms, which includes MD5, Tiger, SHA-256, and other common hashing algorithms. These can then be used in `hash($algorithm, $data)` to generate the digest for the specified data. It is important to note `hash` should not be used for passwords; password hashing will be covered later in this article. Hashing data should be used for verifying data integrity, meaning the data hasn't been altered from the original. If it had been modified, then the hash would change. Often you can find a message digest, which is the output of a hash of the message.

It is important to understand hashes are one-way, meaning you cannot take the output and determine the input. If you can determine the input from the digest, then the hashing algorithm is considered broken for cryptographic purposes, but may still be useful in other applications. For example, MD5 is considered cryptographically broken but is still commonly used to determine message digests, including in popular source control versioning. Recent research into collisions with MD5 is, however, sparking a migration away from using MD5 for digest purposes.

Password Hashing

Password hashing and checking is a very common requirement. PHP has built-in functionality to assist with this. To hash a password, simply call:

```
password_hash($password, $algorithm, $options)
```

This function uses the latest recommended hashing algorithm by default, and a cost of 10 by default, so you can simply use `password_hash($password)` if the defaults meet your requirements. Before PHP 7 there was a `salt` option, but it's deprecated because the randomly generated salt should be used to maximize security. Executing the `password_hash` function multiple times with the same input will result in different output because of the random salt being used, which means users with the same password will have different hashes.

The `password_hash` function is used in conjunction with:

```
password_verify($password, $hash)
```

The `$password` is the password entered, and the `$hash` was stored when the password was created. This provides a secure, timing-attack safe, and easy way to hash and verify passwords in PHP.

Enforce Strong Passwords

StupidPass[6] can be used to ensure strong passwords are used. It provides a series of checks against a string to see if it is long enough, has four character sets (uppercase, lowercase, numeric and special characters); is not a common password (based on lists from research); and other optional checks to determine if a provided password is sufficiently complex.

The dictionary used by StupidPass is entirely customizable, and many dictionaries exist online which can be used to keep current and catch trending catch phrases and commonly used passwords.

To install StupidPass, use:

```
composer require northox/stupid-password
```

To use StupidPass with the default dictionary and options, use:

```
$sp = new StupidPass();
$isValid = $sp->validate($passwordToTest);
```

6 *StupidPass: https://github.com/northox/stupid-password*

PHP[WORLD] 2017

Call for Speakers
Open until June 23rd

This is a conference like no other. Designed to bring together all the communities linked by the PHP programming language.

Together as the PHP community, the sum is greater than the whole.

November 15-16, 2017
Washington, D.C.

world.phparch.com

The maximum length is defaulted to 40, which can be overridden. Environmental words, like the website name, can be included in the checks as invalid passwords. The default dictionary can be expanded and updated and then used. An example of these options is:

```
$sp = new StupidPass($maxLength, $environmentalWords,
$pathToUpdatedDictionary);
$isValid = $sp->validate($passwordToTest);
```

mcrypt

PHP had mcrypt available for years, but it has been deprecated and should not be used with new code. If needed, it can be made available with PECL. Unfortunately, it was built on libmcrypt, which hasn't been maintained for about a decade. This led to an RFC by Scott Arciszewski calling for the deprecation of mcrypt, and the RFC was approved in March 2016. Any new development should not use mcrypt, and any existing code using mcrypt should be upgraded to use OpenSSL, libsodium, or another maintained and secure library to encrypt and decrypt data, depending on the use case and scenario. It's nice to know mcrypt is still available if necessary, but it is more important to know there are better solutions available for your encryption needs.

OpenSSL

OpenSSL has a wide variety of asymmetric cryptography functions and provides a variety of functionality for handling certificates, encryption, and decryption. It is actively maintained, so it gets updates and security patches as needed. The OpenSSL functions can be used with public and private keys to encrypt and decrypt data. It can also be used to generate and verify signatures, and for certificate signing requests (CSR).

To create a new private key, use:

```
openssl_pkey_new( $config)
```

`$config` is an optional array which can include key bits, key type, if the key should be should be encrypted, and other options. This function returns a resource, which can then be used to decrypt data encrypted with the public key. To write the private key to file for future use, call the function:

```
openssl_pkey_export_to_file(resource $privateKey, $filename)
```

For example, if you created a private key using `openssl_pkay_new` you can write the private key to a file named `private.key`.

Now that a private key has been created, a public key can be generated using:

```
openssl_pkey_get_details($privateKey)
```

This returns an array with details about the private key. Then use the array key `key` and write it to file or whatever else needs to be done. For example, you can use `file_put_contents('public.key', $details['key'])` to create a public key file based on the private key. The public key can then be distributed to others so they can decrypt encrypted data from the holder of the private key and encrypt data which can only be decrypted using the private key. You can then clean up the private key from memory using `openssl_free_key($privateKey)`.

To use the public key to encrypt data, use:

```
openssl_public_encrypt($plaintext, &$encrypted, $publicKey)
```

This will save the plaintext message into the `$encrypted` variable, which is the encrypted version of the message. The encrypted data can then be sent to the holder of the private key, who can then decrypt the message using:

```
openssl_private_decrypt($encrypted, $plaintext, $privateKey)
```

To use the private key to encrypt data, use:

```
openssl_private_encrypt($plaintext, &$encrypted, $privateKey)
```

The encrypted data can then be sent out to be decrypted with the public key.

This process of using the private and public keys is used not only to encrypt data but also to provide authentication. Decrypting data with the public key helps others ensure data was encrypted using the private key. This means as long as the private key is kept private and not shared or leaked, then only the private key holder can send encrypted messages that were encrypted using that key and are the only ones who can read data encrypted by the public key.

libsodium

Exciting news in the PHP community is the library libsodium will be included in PHP as of PHP 7.2. Around the same time mcrypt was deprecated, libsodium was on its way to being included in the PHP core thanks to an RFC by Scott Arciszewski approved in February 2017. Arciszewski also points out this makes PHP the first programming language to have a modern cryptography library built-in. Including libsodium in the PHP core, means PHP now has a modern cryptography library which can encrypt, decrypt, hash passwords, and perform other cryptography-related operations.

The key to using libsodium is knowing how to generate random strings. This can be accomplished using:

```
$random = \Sodium\randombytes_buf($numBytes);
```

Where `$numBytes` is usually passed in using a Sodium constant like `\Sodium\CRYPTO_SECRETBOX_KEYBYTES` or `\Sodium\Sodium\CRYPTO_BOX_NONCEBYTES` or others. This can be used to generate keys for encrypting and decrypting data, and for generating a nonce, which is a random string to be used once and only once.

There is a free online book created by Paragon Initiative that is an excellent resource when using libsodium. The

libsodium book[7] includes terminology, usage, and examples for encrypting, decrypting, hashing, and other features available in libsodium.

CSPRNG

The Cryptographically Secure Pseudo-Random Number Generator, or CSPRNG, is used (as the name suggests) to generate random numbers and bytes which are safe to use with cryptography. It includes `random_bytes($length)` to generate bytes and `random_int($min, $max)` to generate integers. These should be used instead of `rand` and `mt_rand`, which are not safe for cryptographic purposes and do not produce cryptographically safe values.

CSPRNG works differently depending on which operating system is used. On Linux machines, it uses a `getrandom` system call. On Windows, it uses `CryptGenRandom`. On all other operating systems, it uses `/dev/urandom`.

Conclusion

Cybersecurity is not only important today, but will become more important over time. As applications become more complex and more data becomes available online, the likelihood of being attacked only increases. Keeping current with critical security trends and how to prevent successful attacks can help keep a project secure and can be vital to the overall success of an organization. Leveraging PHP features for cryptography, using the reports and tools described in this article, and being mindful of web application vulnerabilities can make a monumental difference in improving the security in projects being worked on.

Mark Niebergall is a passionate security-minded PHP software engineer with over a decade of hands-on experience working on PHP projects. He is the Utah PHP Users Group President and often speaks at user groups and conferences. Mark has a Masters degree in MIS, minor in CS, is SSCP and CSSLP certified, and volunteers for (ISC)2 exam development. Mark is an expert drone pilot, casual gamer, and proudly teaching his kids how to push buttons and use technology.

7 *libsodium book:* https://paragonie.com/book/pecl-libsodium

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[**architect**]

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe