Wifi:

Sheraton Conference

Pass: phptek2018

Twitter:

#phptek

Rate the Talks

https://joind.in/event/phptek-2018

PHP[TEK] 2018

Thanks to Our Sponsors

AUTOMATTIC

RED HAT® OPENSHIFT

RingCentral®

MailChimp

Square

Magento®

Evolution of PHP Security

php[tek] 2018 · Eric Mann

Today's Agenda

- Introductions and Setup
- Authentication
- Credentials Management
- Encryption
- Lunch?
- Session Management
- Data Validation / Sanitization
- Long-term Trust
- Server Hardening
- Miscellany / Questions?

TOZNY

Module 0

INTRODUCTIONS AND SETUP

TOZNY

Introduction

- Who am I?
- Who are *you?*
- What are you hoping to learn this week?

- Project overview
- Project requirements
- PHP, SQLite, (Docker, maybe)

TOZNY

Setup

- Clone the project repository

```
git clone https://github.com/ericmann/contacts.git php-security
```

- Run the "smoke test" script in the repo to test your system

```
cd php-security && php smoke.php
cd php-security && ./dockphp smoke.php
```

TOZNY

Overview

- Project components in `/module` directories
- `index.php` – Basic application bootstrapping and routing
- `util.php` – Generic utilities and template preparation
- `lesson.php` – Actual code we'll be working with

Module 1

AUTHENTICATION

TOZNY

Authentication - Passwords

- Project components in `/module-1`
- The application presents a login screen to protect an /auth endpoint
- The actual processing of passwords is not yet handled
- Let's *start* by comparing a submitted password with a static string

Authentication - Passwords

- Project components in `/module-1`
- The application presents a login screen to protect an /auth endpoint
- The actual processing of passwords is not yet handled
- Let's *start* by comparing a submitted password with a static string

- We don't want the *server* to know the password … let's hash it!

TOZNY

Authentication - Passwords

- Project components in `/module-1`
- The application presents a login screen to protect an /auth endpoint
- The actual processing of passwords is not yet handled
- Let's *start* by comparing a submitted password with a static string

- We don't want the *server* to know the password … let's hash it!

- We don't want to store passwords in code. Let's use SQLite

TOZNY

Authentication – Passwords (SQLite)

- There's a users.db file in the module. Let's load that with the SQLite extension
- The single users table contains three columns:
  - User ID
  - Username
  - Password
- Instead of comparing to a string in code, let's compare to one in the database (there's one user already registered)
  - Username: admin
  - Password: thisisinsecure

Authentication – Passwords (SQLite)

- Let's work through adding password reset tokens
  - These are *like* passwords, but don't require two pieces of information
  - Let's split the tokens into two pieces to simulate usernames and passwords
- Why is this important?
  - Timing attacks
  - Potential identity impersonation

```
SELECT tokenid, userid FROM reset_tokens WHERE token = :token
```

TOZNY

Module 2

CREDENTIALS MANAGEMENT

TOZNY

Credentials Management

- Project components in `/module-2`
- The project uses a (fake) remote API that requires credentials
  - API Key: AMZ123PW0987
  - API Secret: 5ec8cc8dca0d70801dec3e4d43b58fba
- How can we use these secrets while keeping them out of source control?

- gitcrypt
- Amazon KMS

TOZNY

Module 3

ENCRYPTION

TOZNY

Symmetric Encryption

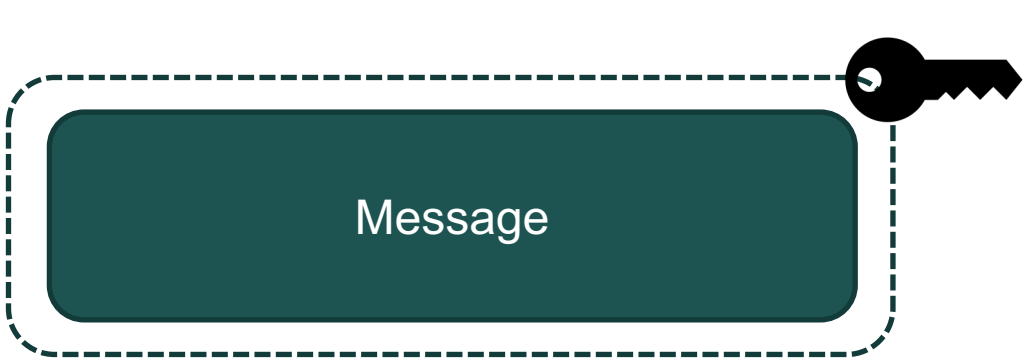
- Uses a fixed encryption key that's shared among all parties
- *Always* uses a unique, random, unpredictable nonce
- Very fast, even on large files

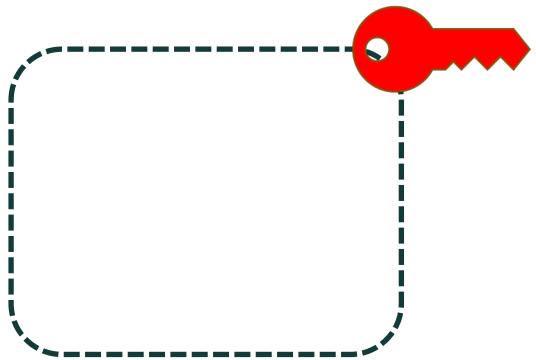
```
$message = 'This is a secret message';
$nonce = random_bytes(SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);
$key = ''; // This is generated elsewhere

$encrypted = sodium_crypto_secretbox($message, $nonce, $key);
$store = base64_encode($encrypted);
```

TOZNY

Asymmetric Encryption

- Uses two, related keys - one private and *never* shared, one public
- Public key is used to encrypt, private to decrypt
- Typically somewhat slow and only used for *small* messages
- Can be paired with symmetric encryption for speed and size





TOZNY

Asymmetric Encryption

```
$message = 'This is some secret message';
$nonce = random_bytes(SODIUM_CRYPTO_BOX_NONCEBYTES);
$sender_key = ''; // Private key of the sender
$recipient_key = ''; // Public key of the recipient
$keypair = sodium_crypto_box_keypair_from_secretkey_and_publickey(
    $sender_key,
    $recipient_key
);

$encrypted = sodium_crypto_box($message, $nonce, $keypair);
$store = base64_encode($encrypted);
```

TOZNY

Encryption Walkthrough

- Project components in `/module-3`
- Use one of the two encryption techniques (your choice) to encrypt a secret message stored in a file (secret.txt) on filesystem
- Use the same technique to read that file back from disk on the /auth page and present it to the user

Data Tokenization

- Project components in `/module-3-2`
- Encrypted data is *not* searchable
  - Encrypting an entire database would make queries impossible!
- To search, we need to *tokenize* a field and search on that token
- A user database has been provided with four columns
  - User ID
  - Username
  - Password
  - Email
- Update the implementation to *encrypt* the email address at rest (and transparently *decrypt* it after retrieval)
- Add a fifth column with a *hashed* version of the email so we can "find user by email address"

Lunch

TOZNY

Wifi:

Sheraton Conference

Pass: phptek2018

Twitter:

#phptek

Rate the Talks

https://joind.in/event/phptek-2018

PHP[TEK] 2018

Module 4

SESSION MANAGEMENT

TOZNY

Session Management

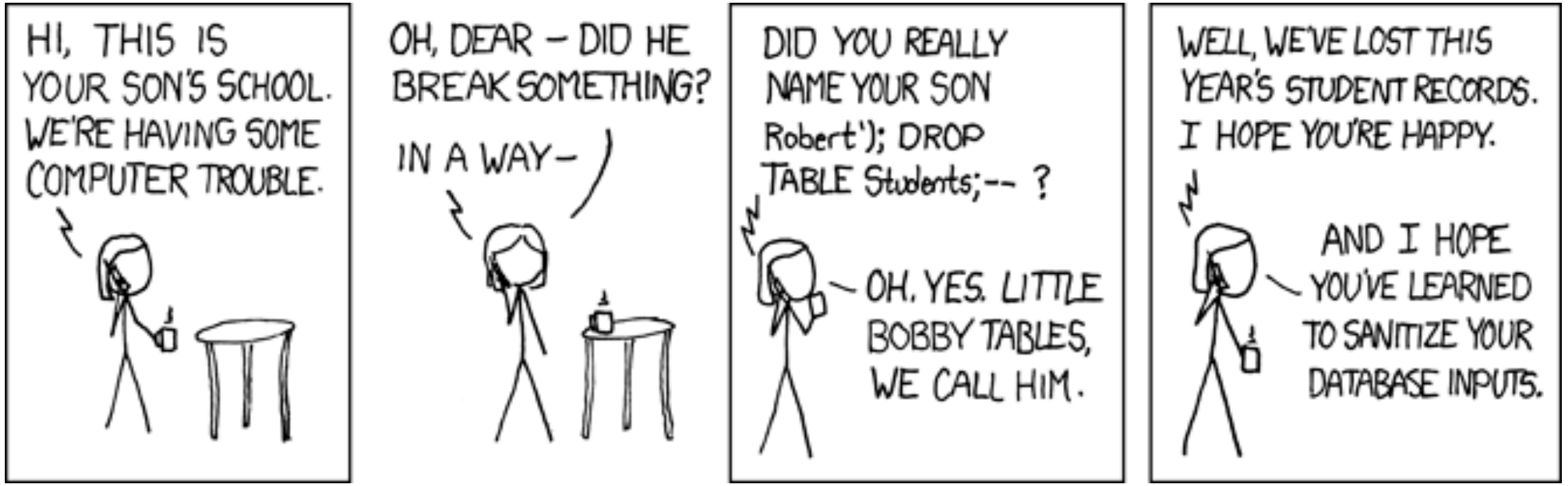
- Storing session data in userland is a bad idea
  - You *should not* trust data presented by users
- Session data is stored, by default, serialized in the filesystem
  - This can be viewed or manipulated by other processes on the server
  - We want to protect session data the same way we protected email addresses
- Sessions are not queryable, so we do not need tokenization
- Your task: add *encryption* to existing sessions to protect sensitive data!
- (If we have time) We *could* store sessions in SQLite using a custom SessionHandlerInterface

Module 5

DATA VALIDATION / SANITIZATION

TOZNY

Little Bobby Tables …



TOZNY

PHP PDO

- The PDO (PHP Data Objects) extension makes database access easy
- It handles connections, queries, errors, and even data sanitization
  - PDO can *parameterize your inputs* to prevent malicious user input from executing rogue queries
  - No more Bobby Tables!

PHP PDO

```
$handle = new \SQLite3('contacts.db');

$handle->exec(sprintf("INSERT INTO contacts (name, email) VALUES ('%s', '%s')",
$name, $email));

$handle->close();
```

```
$handle = new \PDO('sqlite:contacts.db');

$statement = $handle->prepare(("INSERT INTO contacts (name, email) VALUES
(':name', ':email')");

$statement->execute([':name' => $name, ':email' => $email]);
```

TOZNY

PHP PDO

- Project components in `/module-5`
- Up to now, I've handled any PDO connections for you
- This module is the same as our user lookup from lesson 3
  - This time *you* need to handle the query creation
  - Complete the @TODOs in the code, using existing queries as a guide
  - What *other* queries might you write for additional databases?

TOZNY

Module 6

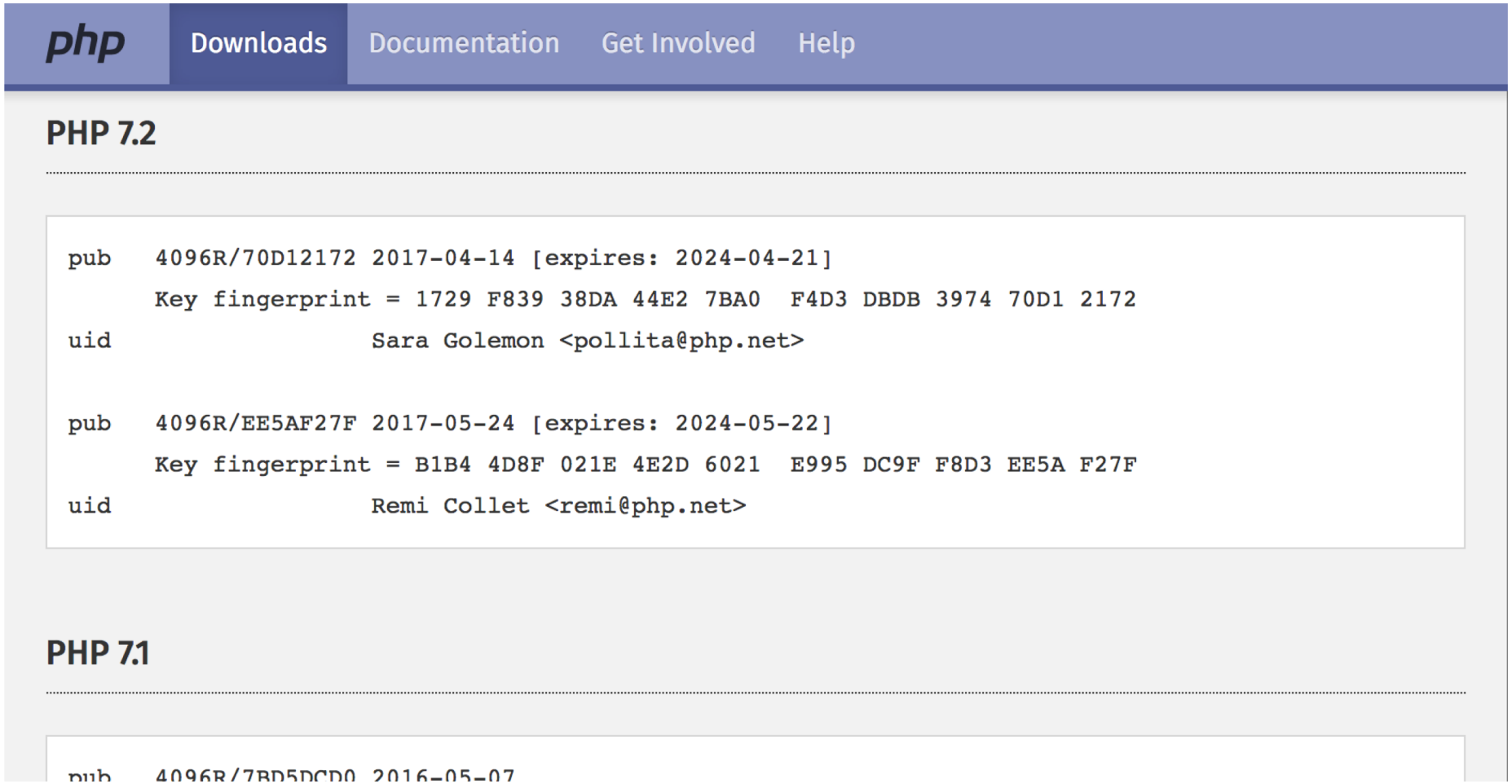
LONG-TERM TRUST

TOZNY

Signing and Verification

- Sometimes you want to verify the integrity of data outside your control
- Cryptographic signatures over the data identify a party who's vouching for the data
- Signatures leverage a public/private keypair (similar to asymmetric encryption from earlier)
- Different algorithms and systems are accepted
  - GPG with RSA is popular

Signing and Verification



TOZNY

Signing and Verification

- Sometimes you want to verify the integrity of data outside your control
- Cryptographic signatures over the data identify a party who's vouching for the data
- Signatures leverage a public/private keypair (similar to asymmetric encryption from earlier)
- Different algorithms and systems are accepted
  - GPG with RSA is popular
  - PHP ships with EdDSA via Libsodium

Signing and Verification

- Project components in `/module-6`
- There is a *signed* file in the project
- Load the file into PHP and verify the signature
- The public key I used is:

```
4af816254d721f156edb2589fddf55db06a16fe546cbc252708e287796fc16a7
```

- Create your own keypair and use it to sign a file

TOZNY

Module 7

SERVER HARDENING

TOZNY

Server Hardening

- How is PHP running on the server?
  - Use a specific user account and group
  - Ensure the user *only* has access to PHP resources on the server
  - Lock down PHP's access with `open_basedir`
- What does the project structure look like?
  - Where is the project *entrypoint?*
  - Where are other source files?
  - Is there anything sensitive in the project?
  - Is `/vendor` accessible to remote users?

TOZNY

Server Hardening

- Further resources:
  - http://securingphp.com/
  - https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet
  - https://paragonie.com/
  - Web Security 2016 – php[architect]
  - Security Principles for Web Applications – php[architect]

TOZNY

Miscellany

QUESTIONS? COMMENTS? OPEN DISCUSSION

TOZNY

Wifi:

Sheraton Conference

Pass: phptek2018

Twitter:

#phptek

Rate the Talks

https://joind.in/event/phptek-2018

PHP[TEK] 2018