Security

JavaScript malware via images: https://blog.confiant.com/confiant-malwarebytes-uncover-steganography-based-ad-payload-that-drops-shlayer-trojan-on-mac-cd31e885c202

Learning Objectives

Better Authentication

Learn how to identify security risks

Learn to assess security needs

Learn basic methods to protect against hackers

- General security principles

A first step towards a better login

Previous Login example stored login and password in plain text

It is better to encrypt passwords

Store them in a separate file

Passwords

Many sites have some form of user validation

Common to use usernames and passwords

For sites using databases, it makes sense to store this data in the database (we'll get to this in a bit)

But!!

- Never ever ever store plain text passwords in your database
- Need to encrypt them

Encrypting Passwords

It is typical to use 1-way encryption for passwords

Not even we can decrypt them, instead, we encrypt password attempts and compare the stored password and the new one

We store the encrypted password

How to choose a hash algorithm

There are many different hashing algorithms

- all have different purposes and meet different needs

Not all hashing algorithms are good for passwords

Good candidates:

- md5 (not recommended anymore)
- sha-1
- sha-2 (sha-256, sha-512)
- whirlpool, Tiger, AES
- Blowfish

Blowfish

Advantages:

- built in to php (so are others)
- high level of security
- public domain
- no patents
- free to use
- slow

More info:

- https://en.wikipedia.org/wiki/Bcrypt
- https://en.wikipedia.org/wiki/Blowfish_(cipher)

Encryption Algorithms in PHP

Many built in:

- md5($password);
- sha1($password);
- hash('sha1', $password);
  - http://php.net/manual/en/function.hash.php
  - hash_algos() will give you more info about available hash algorithms
    - http://php.net/manual/en/function.hash-algos.php
  - very fast
  - not suitable for passwords
- crypt($password, $salt);
  - slow (which is good)

crypt function

supports 6 different encryption algorithms:

- DES, Ext-DES, MD5, SHA-256, SHA-512, Blowfish
- Note that these algorithms will work differently when using crypt than when using hash()
  - may be run multiple times making them slower and changing the output
- We don’t have to pass in to crypt which algorithm we want (directly)
  - it gets included in the salt

Salting passwords

A little aside about Rainbow tables...

Rainbow Tables

Suppose someone (the bad man) gets access to our hashed passwords

They could take an entry (a hashed password) and try all possible passwords, hashing them until they find a hash that matches

- This kills the password (reveals it)
- This is slow, but...
  - we could build a table ahead of time: Rainbow Table
  - we then just have to look for the hash (suddenly, a linear time solution appears!)

Salts help us prevent this kind of attack

Salts

A salt is just some extra data that is “added” to the password before it is encrypted

Renders rainbow tables less appealing

For example we could turn $password into:

- “Put salt on my {$password}”

Unique salts

Another option is to make a salt uniquely for each user

- “put salt on {$password} for {$user}”
- this renders rainbow tables even less useful, because a separate table for each salt (or a way way way bigger table)

We can also add pseudo-random strings to salts

- “put salt on {$password} for {$user} at” . time();
- discovering the password requires knowing the random string
- but how do we figure out the salt later?
  - commonly store the salt as well
  - can be stored with the encrypted password
  - can also hash the salt!

Let’s try it

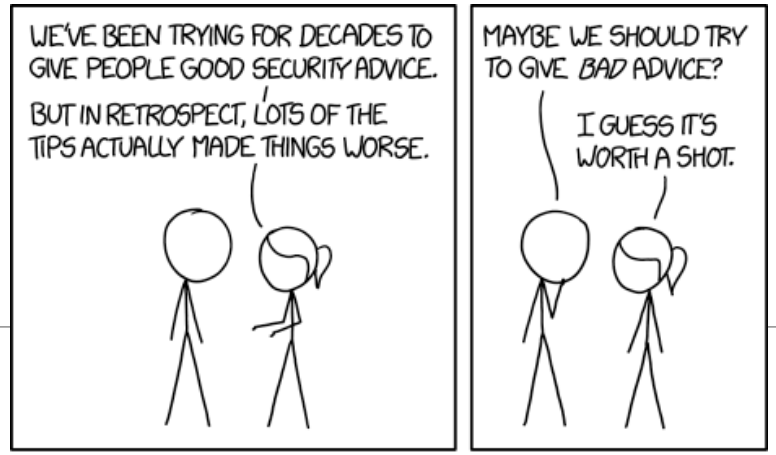
First demo we’ll play with passwords, hashing and creating salts

Second demo, we’ll generalize and create reusable function

- which should be stored somewhere secure

Salts

Why is it ok to store the salt in plaintext with the password?



SECURITY TIPS

(PRINT OUT THIS LIST AND KEEP IT IN YOUR BANK SAFE DEPOSIT BOX.)

- DON'T CLICK LINKS TO WEBSITES
- USE PRIME NUMBERS IN YOUR PASSWORD
- CHANGE YOUR PASSWORD MANAGER MONTHLY
- HOLD YOUR BREATH WHILE CROSSING THE BORDER
- INSTALL A SECURE FONT
- USE A 2-FACTOR SMOKE DETECTOR
- CHANGE YOUR MAIDEN NAME REGULARLY
- PUT STRANGE USB DRIVES IN A BAG OF RICE OVERNIGHT
- USE SPECIAL CHARACTERS LIKE & AND %
- ONLY READ CONTENT PUBLISHED THROUGH TOR.COM
- USE A BURNER'S PHONE
- GET AN SSL CERTIFICATE AND STORE IT IN A SAFE PLACE
- IF A BORDER GUARD ASKS TO EXAMINE YOUR LAPTOP, YOU HAVE A LEGAL RIGHT TO CHALLENGE THEM TO A CHESS GAME FOR YOUR SOUL.

Why is security important?

What are hackers trying to do?
(Why do we care?)

https://commons.wikimedia.org/wiki/File:Cliche_Hacker_and_Binary_Code_(26614834084).jpg

Why security is important

Hackers may attempt to:

- steal services
- steal information
- maliciously destroy information
- use servers for other purposes
- vandalize

How much security is enough

100% security is impossible

The amount of necessary security is partly dependent on what needs to be protected

- Assess your security needs
- Reassess security needs on a regular basis
- Stay current with security practices, because things change quickly

Security Risks to Web Applications

Web applications are frequent targets of hacking

- Databases often contain personal information, which is valuable!
- PHP is very well known, especially by hackers, so it is an easy target
- PHP is easy to learn, and beginners often develop weak sites
  - hackers count on this
  - hackers test for this
- Small sites are especially vulnerable

General Security Principles

Least Privilege

Never Trust Users

Expect the Unexpected

Defense in Depth

Security through Obscurity

Map out Exposure Points

Least Privilege

Only give as much access as needed:

- users
- code
- employees/staff

Never Trust Users

How can your system tell the difference between a user and a hacker?

Users can also accidentally cause harm if you're not careful

Also applies to employees/admin!

- Do they have access to passwords or account info?
- Do they have admin login privilege?

Expect the Unexpected

Don’t just react to security problems, prevent them

Need to carefully consider:

- all the things a user may try to do
- ways your security may be circumvented

Stay on top of security alerts and news

Defense in Depth

Create layered defense

Add redundant security

- then if something fails, there is still hope

Layers:

- people
- technology
- operations
- software

Security through Obscurity

Limit access to information, even if it seems harmless

Don’t reveal details in your code

- file structure
- usernames

Blacklisting vs. Whitelisting

- define what is allowed, not what is not allowed

Map Exposure Points

What ways can data be accessed?

- URLs
- Forms
- Cookies
- Sessions

What ways is data fed out?

- HTML
- Javascript

What paths does data take?

Versioning

Security is an ongoing commitment

- address bugs as discovered
- apply bug-fixes, security patches, updates
- update everything!
- test before releasing to production

Keep both testing and deployment environments up to date

For php:

- php.net has up to date release/bugs information
- https://cve.mitre.org/
  - common vulnerabilities and exposures

PHPInfo and PHPMyAdmin

These are a common source of vulnerabilities!

Keep them secure

PHPInfo file contains precious information about your setup

- very useful when you're getting setup, but needs to be protected

PHPMyAdmin

- gives you browser access to your db
- very powerful!
- often built in
- https://httpd.apache.org/docs/current/howto/htaccess.html
- https://www.phpmyadmin.net/docs/

Validating Input

If you know data is coming in to your site, deal with it!

Carefully analyze your data expectations:

- types
  - primitive types
  - email, usernames, passwords, ...
- what data is allowed?
- what format is allowed?
- what values are allowed?

Give all variables default values, and ONLY change them if the alternative (user input) is valid

Approaches to handling input

Reject Known Bad

Accept Known good

Sanitization

Safe Data Handling

Semantic Checks

Accept Known Good: Common Validations

Presence/length

type

format

within a set of values

uniqueness

Presence/Length

trim whitespace

check against empty

check against min/max/exact lengths

```
function has_presence($value) {
      $trimmed_value = trim($value);
  return isset($trimmed_value) && $trimmed_value !== "";
}
```

Type

remember that all post values are strings

to check if a string is a number: is_numeric

http://php.net/manual/en/function.is-numeric.php

Format

use regular expressions to validate the format of data

pass the data and the regex to the preg_match function

Careful! Writing correct regexp is hard!!

Use anchor tags (\A xyz\) to mark the start and end

- http://php.net/manual/en/function.preg-match.php

Validating emails, URLs, :

- http://www.php.net/manual/en/function.filter-var.php
- http://php.net/manual/en/filter.filters.validate.php

Within a set of values

If you know there is a fixed set of valid values, define it and check against it

You can check inclusion or exclusion

- same same but different

Uniqueness

It is often useful to check that a value is unique before adding it to a db

This will be db dependent

Before you do this, always escape user-provided values (we will cover this shortly)

Practice

In small groups, decide how you'd validate the following inputs:

- User name

Practice

In small groups, decide how you'd validate the following inputs:

- Postal Code

Practice

In small groups, decide how you'd validate the following inputs:

- Province (assume Canada)

Practice

In small groups, decide how you'd validate the following inputs:

- Phone number

Sanitizing Data

Checking that data is valid IS NOT ENOUGH!

Even valid that is technically valid can be maliciously used

Easily the most important step is to **sanitize**

Two main techniques:

- encoding
- escaping

Cautions:

Do not write your own sanitization functions

- Use built in (tried at tested) functions

Do not remove or correct invalid data

- it may be tempting, but it easy to make a huge mess

Built in Sanitization functions in php

htmlspecialchars

htmlentities

strip_tags

urlencode

json_encode

htmlspecialchars

this function works by converting special characters to html entities

< converted to <

> converted to >

“ converted to "

http://php.net/manual/en/function.htmlspecialchars.php

htmlentities

converts all possible characters to html entities

same as htmlspecialchars except will convert all html characters with entity representations to html entities

http://php.net/manual/en/function.htmlentities.php

strip_tags

Even though we said don't remove stuff...

- This technique can be ok

Strips both PHP and HTML tags from a string

strip_tags('<h1>A heading</h1>');

- returns: A heading

You can indicate tags to ignore:

strip_tags(' wowzers! , ' ');

- returns: wowzers!

http://php.net/manual/en/function.strip-tags.php

sanitizing for sql

One (not best) approach is to use the PHP function addslashes

puts slashes before all characters that should be escaped

- ‘
- “
- \
- NULL

Better approach is to use prepared statements! ← Coming Soon!

If you’re using pdo (like we are) make sure to bind parameters

- $stmt = $dbh>prepare("SELECT name, age FROM users WHERE name = ?");
- no other escaping is needed!

If you’re using sqli you can use mysqli_real_escape_string

- http://php.net/manual/en/function.mysqli-escape-string.php

urlencode()

URL-encodes a string

converts all non-alphanumeric characters to their url representation (except _)

- % followed by 2 hex digits
- spaces converted to +

http://php.net/manual/en/function.urlencode.php

filter_var

another way to filter:

- use filter_var
- pass in appropriate filter
- http://www.php.net/manual/en/function.filter-var.php

Very useful for validation:

- email addresses (**FILTER_VALIDATE_EMAIL)**
- Ip addresses, IP4 and IP6 (**FILTER_VALIDATE_IP)**
- URLs (**FILTER_VALIDATE_URL)**
- …many more!

filter_var

another way to filter:

- use filter_var
- pass in appropriate filter
- http://www.php.net/manual/en/function.filter-var.php

Very useful for sanitization:

- email addresses (**FILTER_SANITIZE_EMAIL)**
- Numbers (**FILTER_SANITIZE_NUMBER_FLOAT, FILTER_SANITIZE_NUMBER_INT)**
- URLs (**FILTER_ FILTER_SANITIZE_URL _URL)**
- …many more!

Keeping Code Private

Only show what you need to

- html
- javascript
- ...

Organize your code

Public Code

This can be kept in publicly indexable location

accessible by web server

presentation code

calls to functions in private libraries

Private Code

not accessible to web server

credentials

- keep in separate files
- **don’t put in version control**
- set permissions

Put an index.php (or .html) file in every directory

- prevents server from returning contents of directories
- optionally, redirect to somewhere safe

Attacks

There are many different kinds of attack

We will focus on:

- Cross-site Scripting (XSS)
- SQL Injection

XSS

Cross-site Scripting is the process of injecting JavaScript code into a web page

Tricks another user into running code on their machine

Often used to steal cookies

- containing history or login state

Example:

- user types in JavaScript code as a comment on a page (in a form)
- JavaScript is saved in database and served to another user

Computerphile on XSS: https://www.youtube.com/watch?v=L5l9lSnNMxg

From web application Hacker’s Handbook

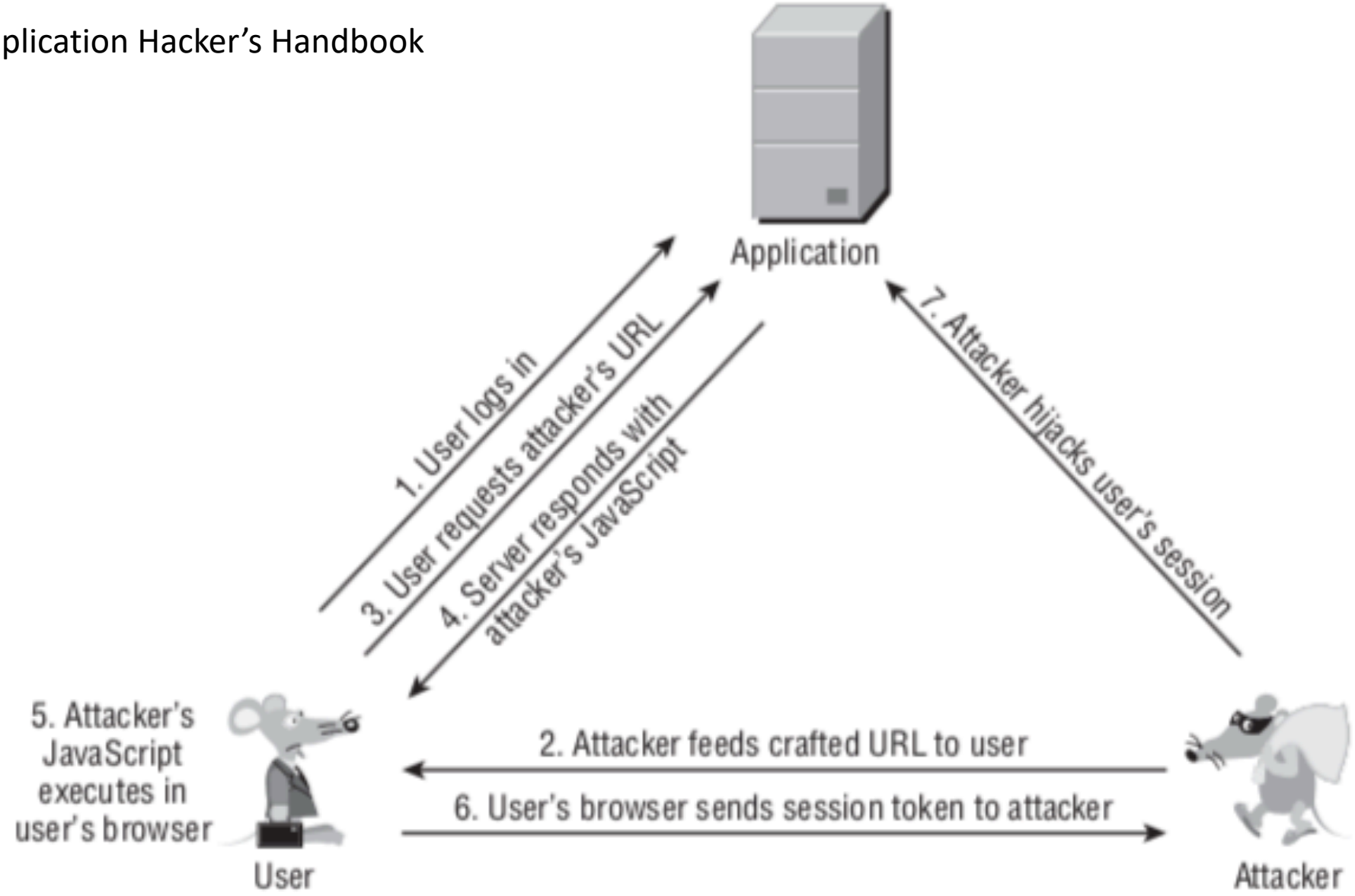


Figure 12-3: The steps involved in a reflected XSS attack

From web application Hacker's Handbook

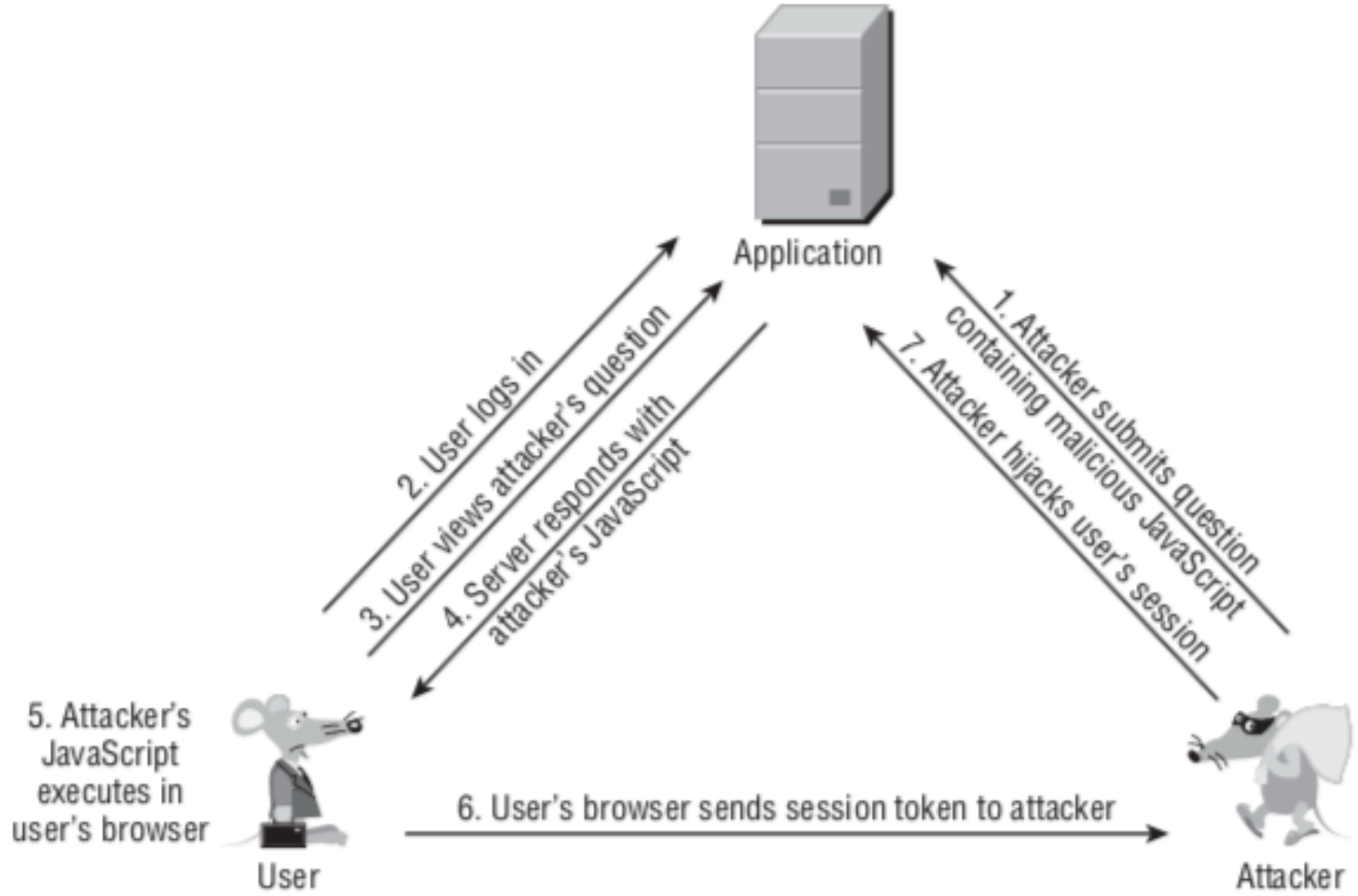


Figure 12-4: The steps involved in a stored XSS attack

Protecting for XSS

Convert characters used in html/JavaScript to character entities

- < > to < >
- use htmlspecialchars on user-entered data

Never insert untrusted data into your html

- HTML escape before inserting untrusted data into an HTML element
- escape untrusted data before inserting it into attributes
- escape untrusted data before inserting into JavaScript data values

XSS more info

For more details (lots!) see here: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Hands-on practice:

- https://www.google.com/about/appsecurity/learning/xss/
- https://xss-game.appspot.com/

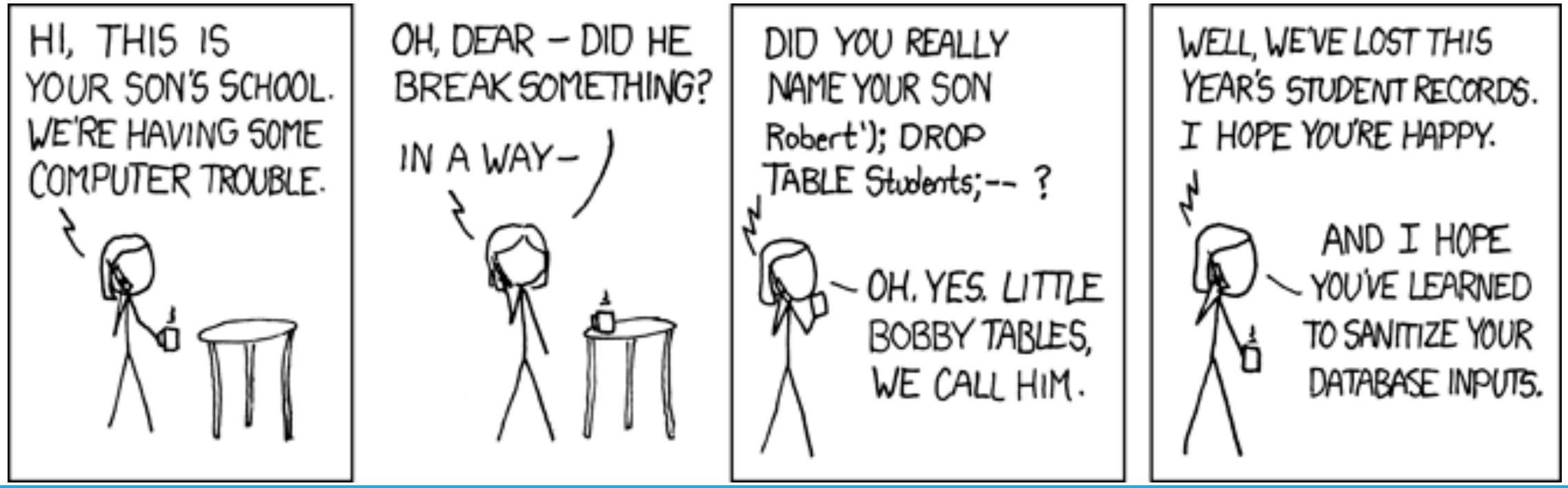
SQL Injection

HACKADAY.COM

SQL Injection Fools Speed Traps And Clears Your Record

SQL Injection

This happens when user data is directly put into an SQL statement



SQL Injection

Tricks can involve bypassing logic

- replacing username and password matching in where clause with something that is always true: 1=1

Tricks can involve inserting a whole query (nested)

Malicious code insertion: drop tables etc.

Good simple intro by Computerphile:

- https://www.youtube.com/watch?v=ciNHn38EyRc

Preventing SQL Injection

Limit privileges to application's database user

Ok: sanitize input for sql

- escape functions
- sql_prep function
- addslashes()

Better: use prepared statements

- bind values to parameters
- no more escaping needed!

Password example

In this demo we will:

- Build a file of helper function:
  - Validation
  - Hashing
  - Check password
- Let the user create an account
  - Select a user name
    - Check for uniqueness
  - Enter a password,
    - Hash and store it
- Let the user log in
  - Check against username and password in db

What does this means for your project?

Identify all point in your code where you insert user-entered data into a database
- Use prepared statements to do this

Identify all point in your code where you insert user-entered data into html/css/javascript
- Sanitize all data output to users, especially if it was user-entered

Can the user “see” details about the structure of your server?
- Don’t output errors that reveal structure of server, like this:
- Connection failed: SQLSTATE[HY000] [1045] Access denied for user 'carruths'@'192.168.18.191' (using password: YES)

Where should you place files with database access code?
- Away from other stuff
- Ideally in a non-searchable folder

Example of recent security problems

<insert company name here> leaking customer information:

- https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/