PHundamental Security

Ecosystem Review, Coding Secure with PHP, and Best Practices

OWASP NYC AppSec 2008

September 24th, 2008

New York PHP Community

November 25th, 2008

Hans Zaunere, Managing Member

Overview

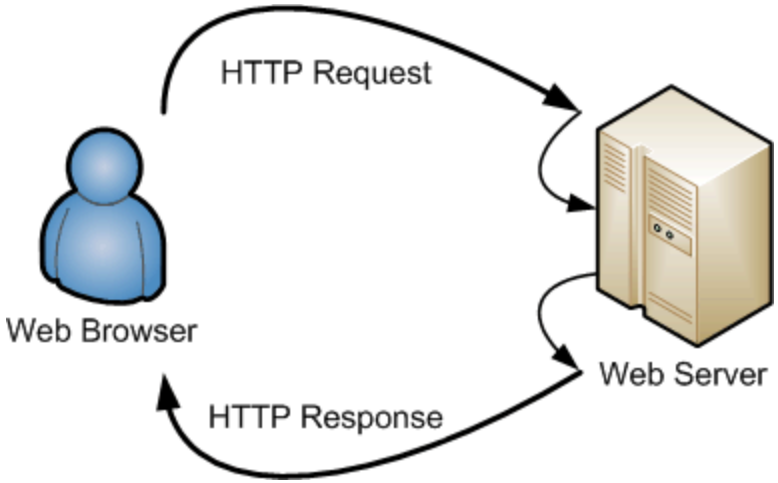
- Introduction
- The Security Ecosystem
- Security Pressure Points in PHP
- Best Practices
- “It’s the System, Stupid!”
- Top 5 Best Practices
- Conclusions

Introduction

PHP is the **P**HP: **H**ypertext **P**reprocessor

- www.nyphp.org
  - www.nyphp.org/phundamentals/
- This is not **YASIXSKOTDT**
  - Yet-Another-SQL-Injection-XSS-Script-Kid-Of-The-Day-Talk
- Guru Stefan Esser recently presented an excellent talk:
  - http://www.suspekt.org/2008/09/18/slides-from-my-lesser-known-security-problems-in-php-applications-talk-at-zendcon/
  - Numerous other excellent cut-paste resources for these ubiquitous attacks
  - Ubiquitous means they can happen in any language

Look Familiar?



HTTP – The Great Equalizer

The Security Ecosystem

- Security fundamentals are common across the board
- Different environments have different requirements
  - Desktop applications are different from web/internet applications
- Web/Internet apps have a huge number of touch points
  - PHP isn't responsible for all of them – in fact, not most
  - The Developer/Enterprise is - in **ALL** cases
- Different languages handle in different ways
  - .NET, Java, Python, PHP all have their idiosyncrasies
- PHP is no different... except...

"More internet applications speak PHP than any other"

The PHP Ecosystem

- PHP gets a bad rap
  - Low point of entry and great flexibility

“Greatest strength and biggest weakness”

- And there’ve been some mistakes
  - Weak default configuration
  - The infamous **magic_*** of PHP
  - PHP Group [rightfully] argues: “What’s a security flaw?”

“It's easy to shoot yourself in the foot with C. In C++ it's harder to shoot yourself in the foot, but when you do, you blow off your whole leg.”

Bjarne Stroustrup, Inventor of C++

Security Points-of-Entry

Three Zones of Responsibility

- PHP is effectively a wrapper around libraries and data sources
  - Many external dependencies and touch points
- There are many zones of responsibility
  - A language is not responsible for them – a developer is
  - A language should not go out of its way to save the developer
    - Frameworks/foundations can be used for this

Security Points-of-Entry

Three Zones of Responsibility

1. Developer
   - Poorly written code by amateurs
   - **Primary cause** for the security ecosystem around PHP
   - Easy to pick up for those with no programming background
   - Laziness - letting PHP do its **magic_***
   - Doing things quick-n-dirty
   - Too forgiving

“Program Smart”

Security Points-of-Entry

Three Zones of Responsibility

2. Extensions and external libraries
   - PHP's greatest asset
   - Sometimes library binding is faulty
     - There could be better extension certification, and it's getting better
   - Sometimes the external library has faults, or behaves in an unforeseen way when in a web environment – possible in any environment
   - Know what extensions you're using, use the minimal number of extensions, and be aware of the environment they were originally designed for.

"Know Thy Extensions"

Security Points-of-Entry

Three Zones of Responsibility

3. PHP Core – “PHP”
   - This is PHP
   - Secunia: 19 advisories between 2003-2008 – **1 in 2008**
     Java: 38+ between 2003-2008
     Ruby: 11+ between 2003-2008

“The List Goes On – PHP is Not Alone”

   - Often safe_* and magic_* related
     – Functions designed to protect developers from ignoring best practices.
     – Or deal with shared environment where incorrect security expectations are prevalent.

“More internet applications speak PHP than any other”

Best Practices

Or, How not to blow off your whole leg

- Best practices are common to any well run enterprise environment
  - Yes, PHP has grown/is growing into this environment very quickly
- Web security is largely about your data and less about exploits in the underlying platform
  - Buffer overflows aren't so much the hot topic
  - ... and those who know, don't talk

PHP Best Practices

Or, How not to blow off your whole leg with PHP

- Installation
  - Avoid prepackaged installs, including RPMs, .deb, etc.
  - If you use them, review their default deployment
  - Installation touch points also typically include Apache/MySQL
- Configuration
  - Use php.ini-recommended
  - Better yet, take the time to know what you're doing and tune configuration files yourself

PHP Best Practices

Be Fashionable – Style and Design

- Don't make PHP guess what you mean
  - Be explicit with variables and types
  - Don't abuse scope – know where your variables come from
  - Avoid **magic_*** and implicitness – BE EXPLICIT
- Keep code small, organized and maintainable
  - Keep code/logic chunks small
  - Use OOP techniques to enforce code execution paths
  - Use includes to keep things organized
- Don't use super-globals directly – wrap for protection

"Be aggressive – B.E. aggressive"

PHP Best Practices

Know Your Data – Love Your Data

- It's always about data
- One of PHP's greatest strengths – loosely typed
  - ... and you guessed it – biggest weaknesses
  - Don't make PHP guess what you mean
- Cast variables, know their types and the data you expect
  - Let PHP do its magic only when you want it to – not by chance

PHP Best Practices

It's 10pm – Do You Know Where Your Data Is?

- Keep tabs on your data's path, lifecycle and type
  - Know where it's come from, what it's doing, and where it's going
  - Filter/escape/cast and throw exceptions every step of the way
- Input validation, output validation, CASTING
- Don't be lazy – be explicit – use OOP

"Casting isn't just for movie producers"

“It’s the System, Stupid”

Networks, Systems, and Databases, Oh My

- No system has a single security pressure point
- Put PHP in the same well managed enterprise environment as other technologies
- Don’t take the easy way out just because you can
- PHP/AMP respond very well to TLC

The Top 5 Best Practices

In No Particular Order

1. PHP is loosely typed and automatically converts between types (known as type casting). However, always consider variables typed, and allow type casting to happen only explicitly. In fact, explicitly type casting variables can add a significant level of data validation and security.

   http://www.php.net/manual/en/language.types.type-juggling.php

2. Remember that PHP is a dynamic scripting language, allowing execution to jump between files during runtime. However, don't abuse lazily including other files – keep all include/require statements in a centralized place and always know what files you're including.

   http://www.php.net/manual/en/function.require.php
   http://www.php.net/manual/en/function.include.php

Top 5 Best Practices

In No Particular Order

3. A typical PHP application will deal with a lot of external data, usually from unknown sources – especially someone's web browser, or worse, someone hand-crafting HTTP requests. PHP is on the front line of security. Always be aware of where your data is coming from, what you'll be doing with it, and where it's going.

   http://www.php.net/manual/en/security.php

4. PHP is very forgiving with its syntax, style and application structure. People on the web are not forgiving. Write clean, organized, and structured code, using the right tools for the job. Do not take shortcuts, depend on PHP's "**magic**" or convenience functionality, or force PHP into guessing what you want to do. You're the programmer – be specific.

   http://www.php.net/manual/en/tutorial.php

Conclusions

Goal: PHP is Just One of the Boys

- PHP is just part of the ecosystem
  - ... and there is awareness and experience on the PHP side
  - The ying/yang of PHP's history overshadows reality
  - Stand by PHP and it'll stand by you
  - Program it - don't hack it

"With great power comes great responsibility."

Spiderman's Uncle

- Web/Internet applications are deep and complex
  - Users, interoperability, data, architecture, support, compliance
  - PHising, hijacking, spam, social engineering – **BROWSERS!**

"PHP is the least of your worries"

The Top 5 Best Practices

And the #1 Fifth Best Practice!

5. PHP's greatest strength and weakness is its ease, power and flexibility. As a result, there is a lot of bad PHP code on the Internet, which is the source of PHP's bad reputation. A programming language is only as good as the programmer.
   Look at most other publicly available PHP code and if it appears sloppy, do what you know is right. If you have experience in another programming language, use it. If not, take the time to properly learn some programming theory, involve yourself with experienced programmers in the community, learn about the other components of web development, and then apply it all to the PHP syntax.
   Your applications will be more secure, more maintainable, and PHP literally becomes a joy to code in.

   Lastly, we encourage everyone – worldwide - to join the New York PHP mailing lists at http://www.nyphp.org/mailinglists.php to ask questions and seek advice.

Questions

hans.zaunere@nyphp.com

For renowned online support, New York PHP Mailing Lists are available to anyone:

http://www.nyphp.org/mailinglists.php