"It's a PHP unserialization vulnerability Jim, but not as we know it"

Sam Thomas

WHOAMI

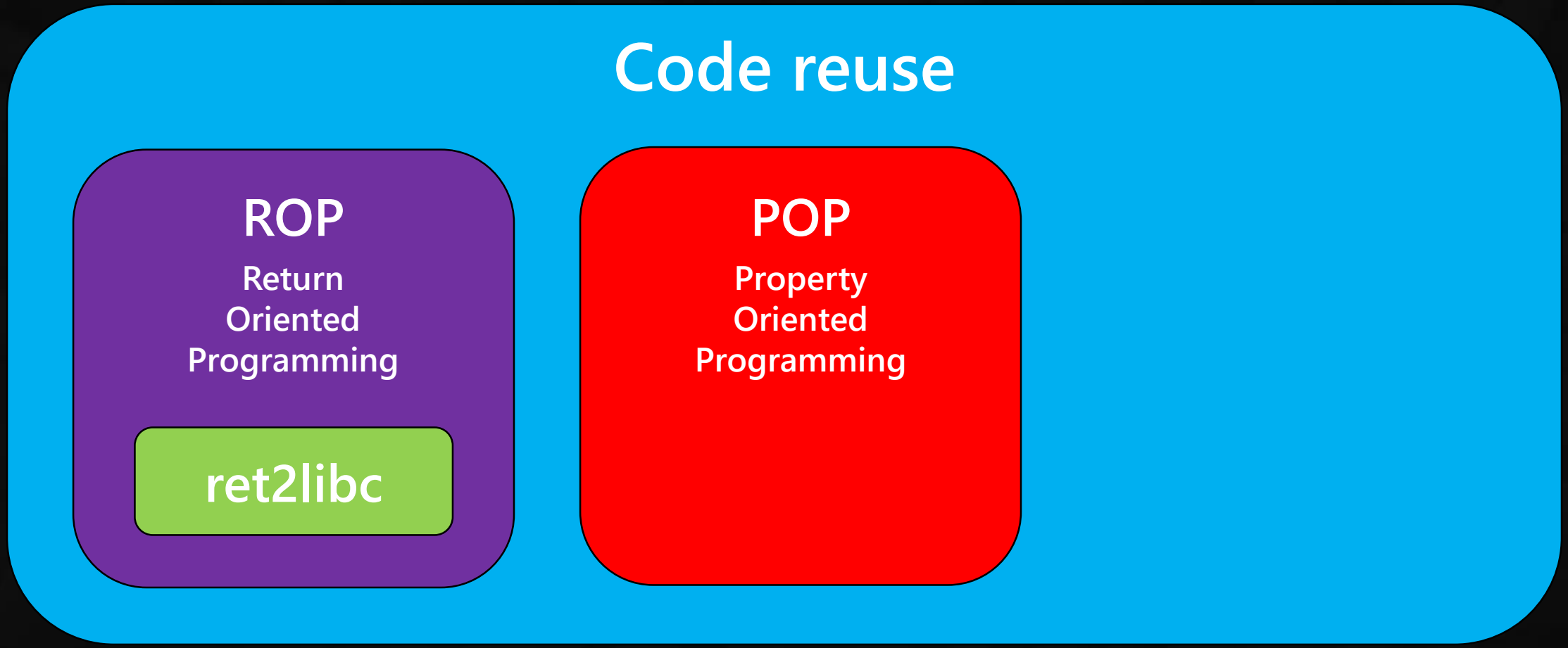
- Director of Research at Secarma Ltd
- Research / Application Assessments / Red Teaming

Introduction

[1]

[2]

Introduction



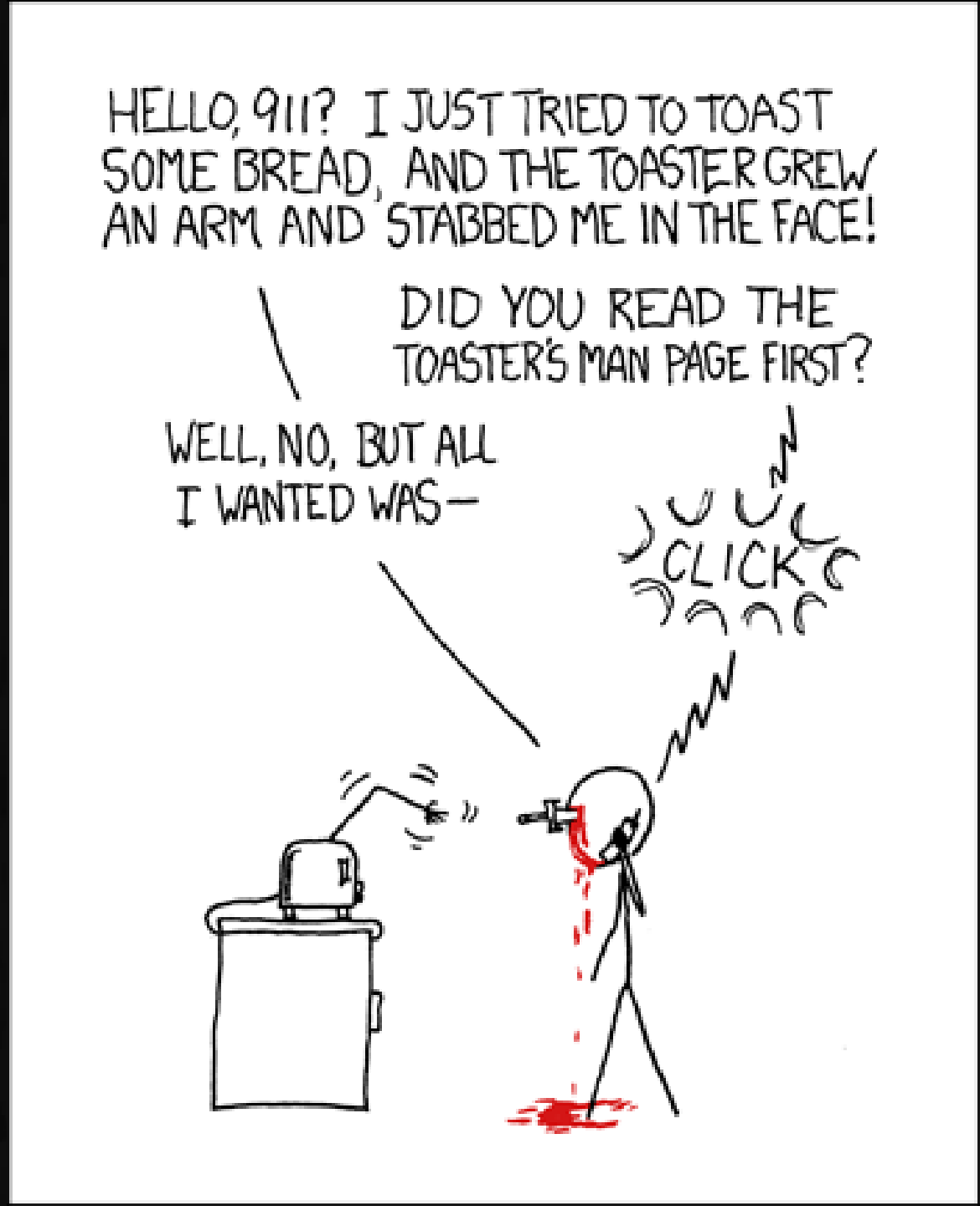
Introduction

- Unserialize is called on attacker controlled input
- Once object is unserialized from input (and when it is destroyed) certain “magic” methods are called
- In favourable circumstances properties and methods can be chained together to cause malicious actions to occur
- TL/DR = Unserialization is bad

Agenda

- Stream Wrappers
- Phar File Format
- Phar Planting
- Identifying Vulnerabilities
- PHPGGC / PHARGGC
- Case Studies
- Defence

XKCD 293 – RTFM



https://www.xkcd.com/293/

Stream Wrappers

[3]

PHP Manual › Language Reference

Change language: English

Edit Report a Bug

Supported Protocols and Wrappers

PHP comes with many built-in wrappers for various URL-style protocols for use with the filesystem functions such as fopen(), copy(), file_exists() and filesize().

file://	http://	ftp://	php://	zlib://	data://	glob://	phar://

Stream Wrappers

file://	http://	ftp://	php://	zlib://	data://	glob://	phar://

- RFI
  - include($_GET['module']);
- SSRF
  - file_get_contents($_GET['url']);
- XXE
  - <! ENTITY xxe SYSTEM "http://example.com">

Remote

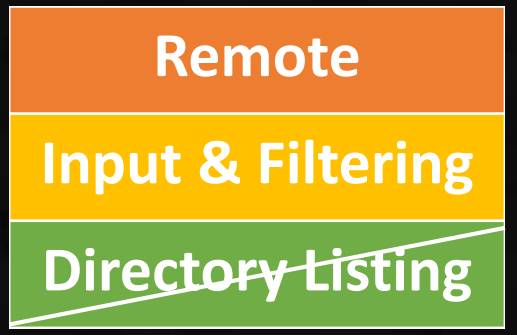
* allow_url_fopen = true

Stream Wrappers

- LFI
  - LFI -> From STDIO: php://input[4]
  - LFI -> Source code reading: php://filter/convert.base64-encode/resource=index.php[4]
- File writing
  - Similar to source code reading, if we have a file write vulnerability which writes undesirable content (e.g. “<?php die()” before our controlled value) we can base64 decode it[5]

Stream Wrappers

- Not used with normal file operations ☹



[6]

Example #1 Basic usage

```php
<?php
// Loop over all *.php files in ext/spl/examples/ directory
// and print the filename and its size
$it = new DirectoryIterator("glob://ext/spl/examples/*.php");
foreach($it as $f) {
    printf("%s: %.1FK\n", $f->getFilename(), $f->getSize()/1024);
}
?>
```

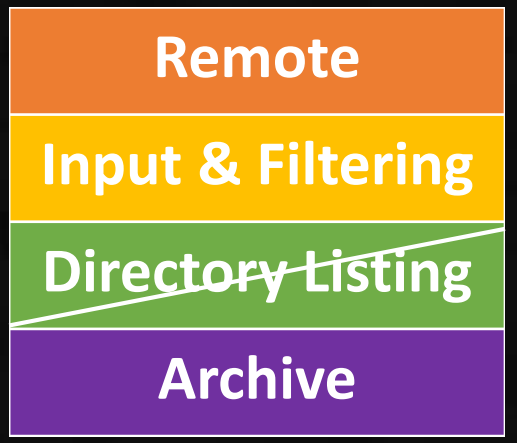
[6]

Wrapper Summary

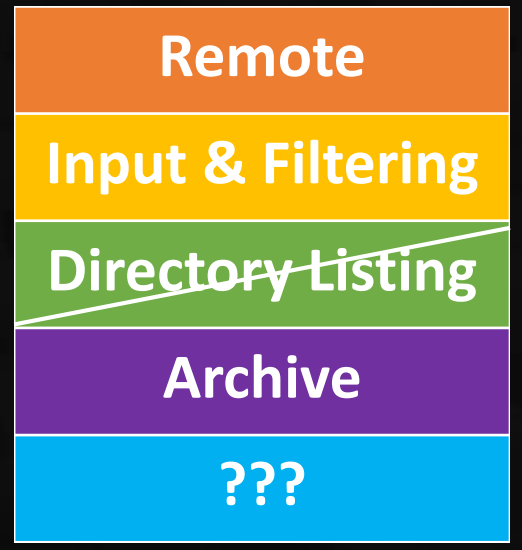
Attribute	Supported
Restricted by allow_url_fopen	No
Restricted by allow_url_include	No
Allows Reading	No
Allows Writing	No
Allows Appending	No
Allows Simultaneous Reading and Writing	No
Supports stat()	No
Supports unlink()	No
Supports rename()	No
Supports mkdir()	No
Supports rmdir()	No

Stream Wrappers

- Exploit vulnerabilities in native code?



Stream Wrappers



Ingredients of all Phar archives, independent of file format

All Phar archives contain three to four sections:

1. a stub
2. a manifest describing the contents
3. the file contents
4. [optional] a signature for verifying Phar integrity (phar file format only)

[8]

Phar file stub

A Phar's stub is a simple PHP file. The smallest possible stub follows:

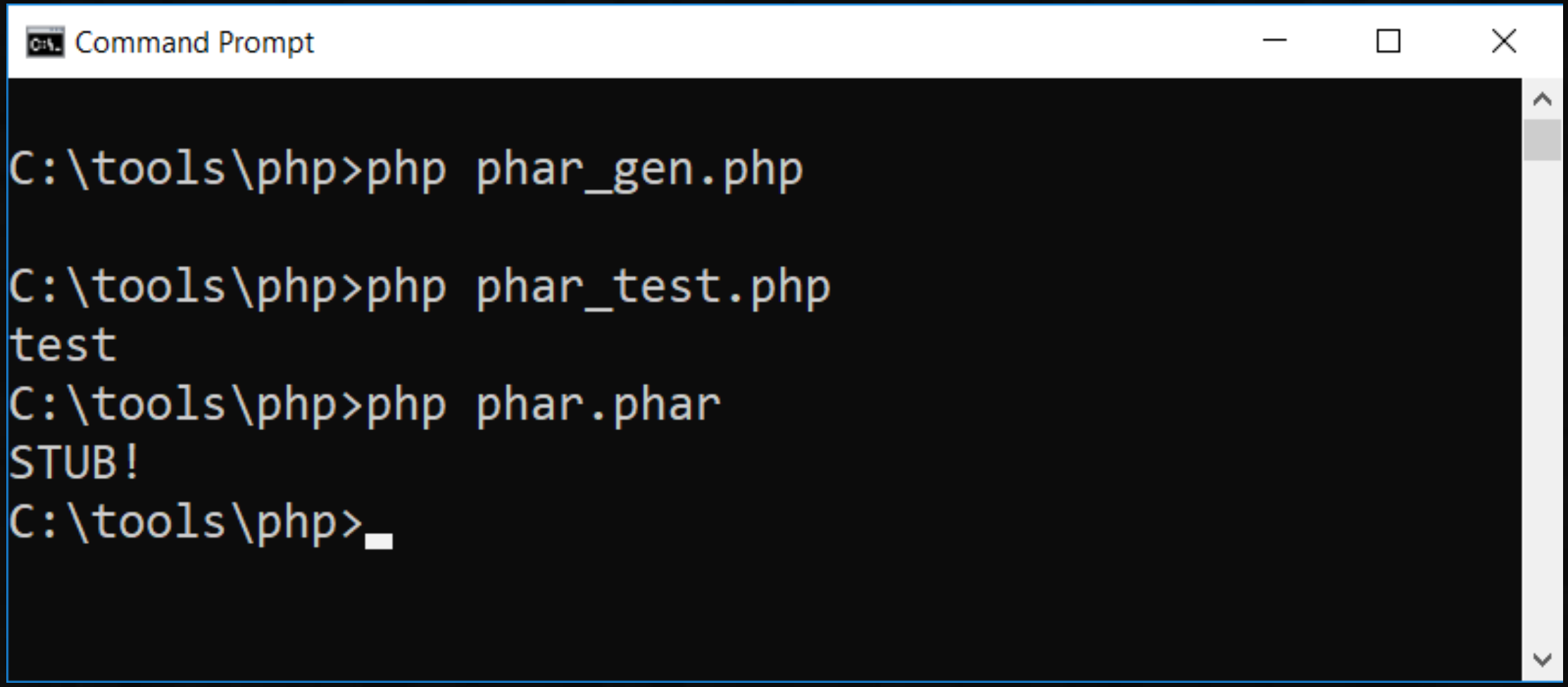
```
<?php __HALT_COMPILER();
```

phar_gen.php:

```php
<?php
    @unlink("phar.phar");
    $phar = new Phar("phar.phar");
    $phar->startBuffering();
    $phar->addFromString("test.txt","test");
    $phar->setStub("<?php echo 'STUB!'; __HALT_COMPILER(); ?>");
    $phar->stopBuffering();
?>
```

phar_test.php:

```php
<?php
    echo(file_get_contents("phar://phar.phar/test.txt"));
?>
```


Command Prompt
C:\tools\php>php phar_gen.php
C:\tools\php>php phar_test.php
test
C:\tools\php>php phar.phar
STUB!
C:\tools\php>

Ingredients of all Phar archives, independent of file format

All Phar archives contain three to four sections:

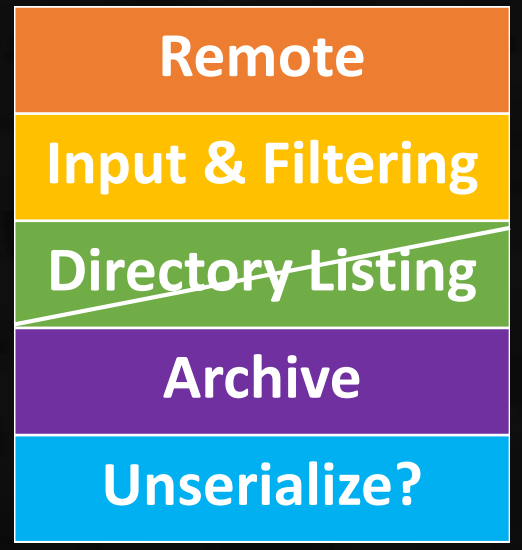
1. a stub
2. a manifest describing the contents
3. the file contents
4. [optional] a signature for verifying Phar integrity (phar file format only)

[9]

Global Phar manifest format

Size in bytes	Description
4 bytes	Length of manifest in bytes (1 MB limit)
4 bytes	Number of files in the Phar
2 bytes	API version of the Phar manifest (currently 1.0.0)
4 bytes	Global Phar bitmapped flags
4 bytes	Length of Phar alias
??	Phar alias (length based on previous)
4 bytes	Length of Phar metadata (*0* for none)
??	Serialized Phar Meta-data, stored in serialize() format
at least 24 * number of entries bytes	entries for each file

Stream Wrappers



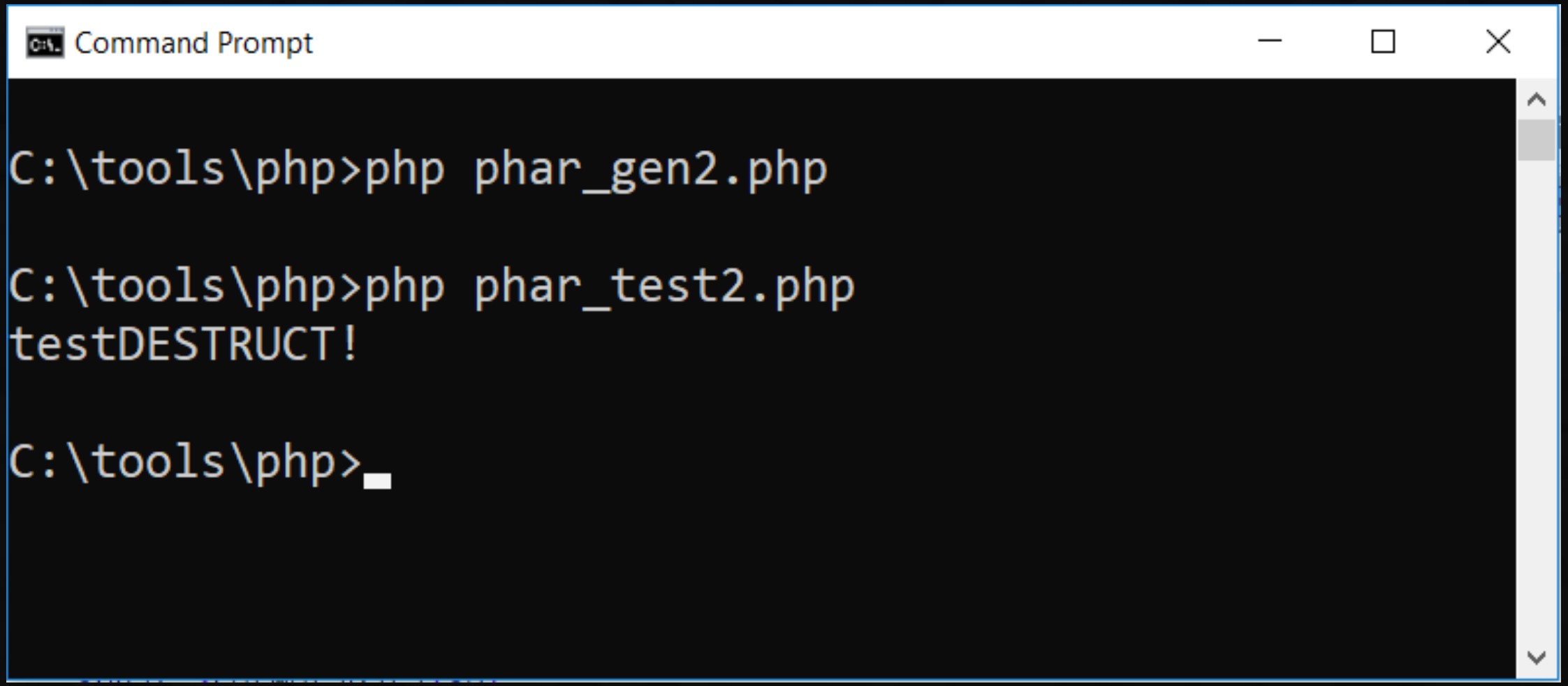
phar_gen2.php:

```php
<?php
    class TestObject
    {
    }

    @unlink("phar.phar");
    $phar = new Phar("phar.phar");
    $phar->startBuffering();
    $phar->addFromString("test.txt","test");
    $phar->setStub("<?php __HALT_COMPILER(); ?>");
    $o = new TestObject();
    $phar->setMetadata($o);
    $phar->stopBuffering();
?>
```

phar_test2.php:

```php
<?php
    class TestObject
    {
        function __destruct()
        {
            echo "DESTRUCT!\n";
        }
    }
    echo(file_get_contents("phar://phar.phar/test.txt"));
?>
```

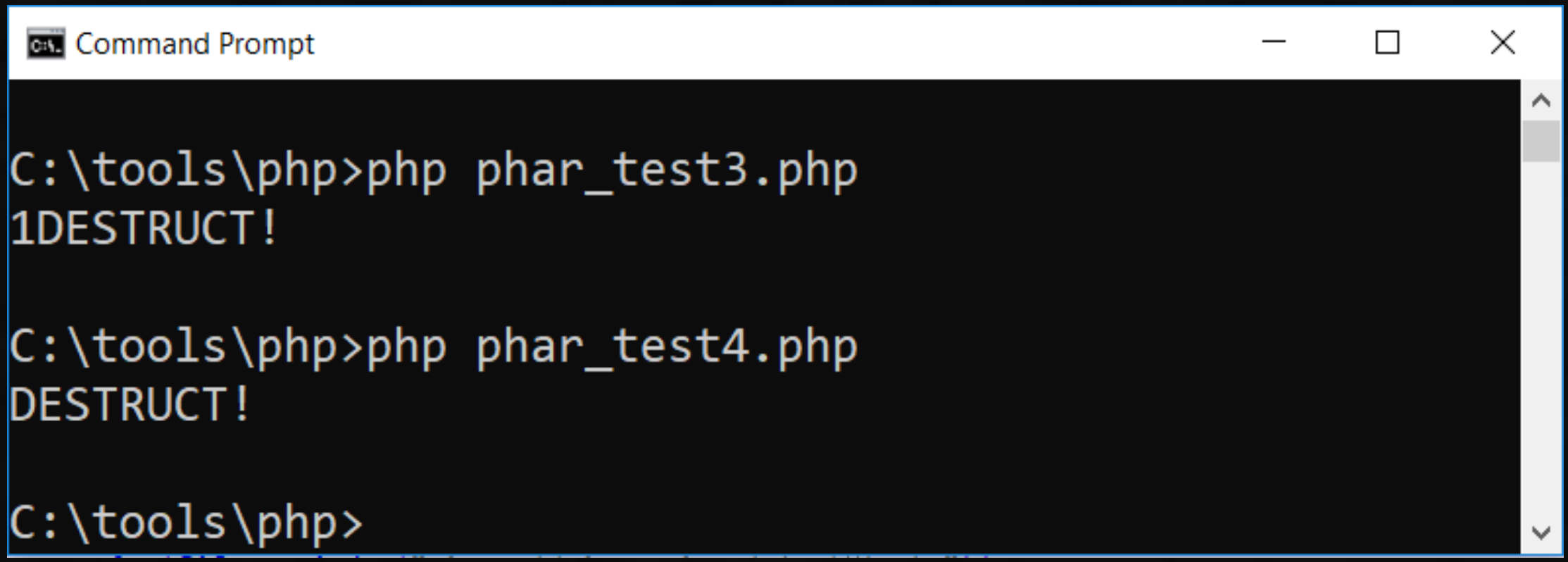

Command Prompt
C:\tools\php>php phar_gen2.php
C:\tools\php>php phar_test2.php
testDESTRUCT!
C:\tools\php>

phar_test3.php:

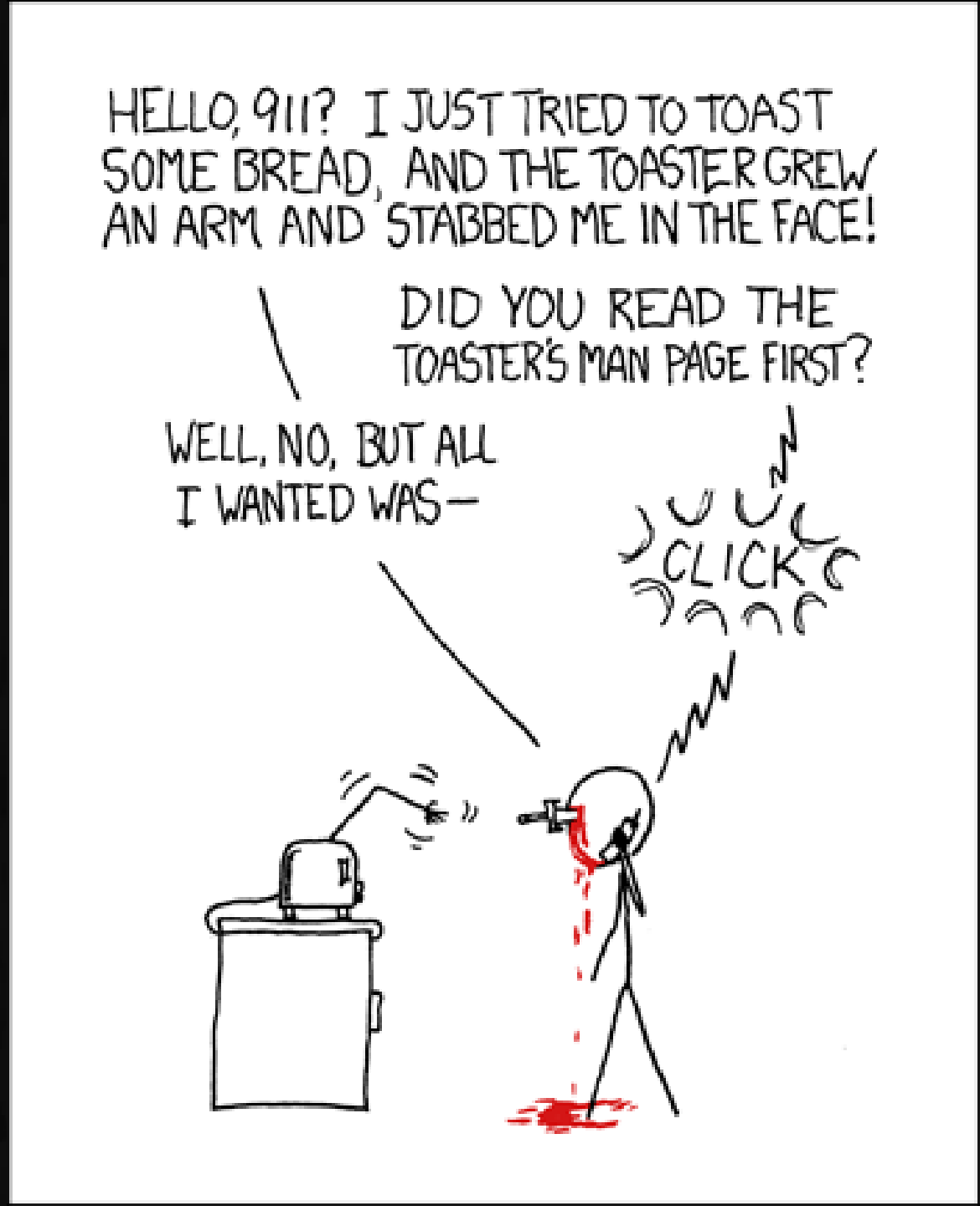
```php
<?php
    class TestObject
    {
        function __destruct()
        {
            echo "DESTRUCT!\n";
        }
    }
    echo(file_exists("phar://phar.phar/test.txt"));
?>
```

phar_test4.php:

```php
<?php
    class TestObject
    {
        function __destruct()
        {
            echo "DESTRUCT!\n";
        }
    }
    echo(file_exists("phar://phar.phar/¯\_(ツ)_/¯"));
?>
```


Command Prompt
C:\tools\php>php phar_test3.php
1DESTRUCT!
C:\tools\php>php phar_test4.php
DESTRUCT!
C:\tools\php>

XKCD 293 – RTFM



https://www.xkcd.com/293/

Basic Attack Methodology

- Cause a valid phar archive (containing malicious serialised meta-data) to exist on the local file system
- Cause a file operation to reference this archive via the “phar://” stream wrapper

Difference from “unserialize()”

- Only “__destruct” or “__wakeup” will work as initial trigger
- “__destruct” chains are executed in a context where the current working directory is “/” (no relative paths)

Ingredients of all Phar archives, independent of file format

All Phar archives contain three to four sections:

1. a stub
2. a manifest describing the contents
3. the file contents
4. [optional] a signature for verifying Phar integrity (phar file format only)

Ingredients of all Phar archives, independent of file format

All Phar archives contain three to four sections:

1. a stub
2. a manifest describing the contents
3. the file contents
4. [optional] a signature for verifying Phar integrity (phar file format only)

Phar File Format

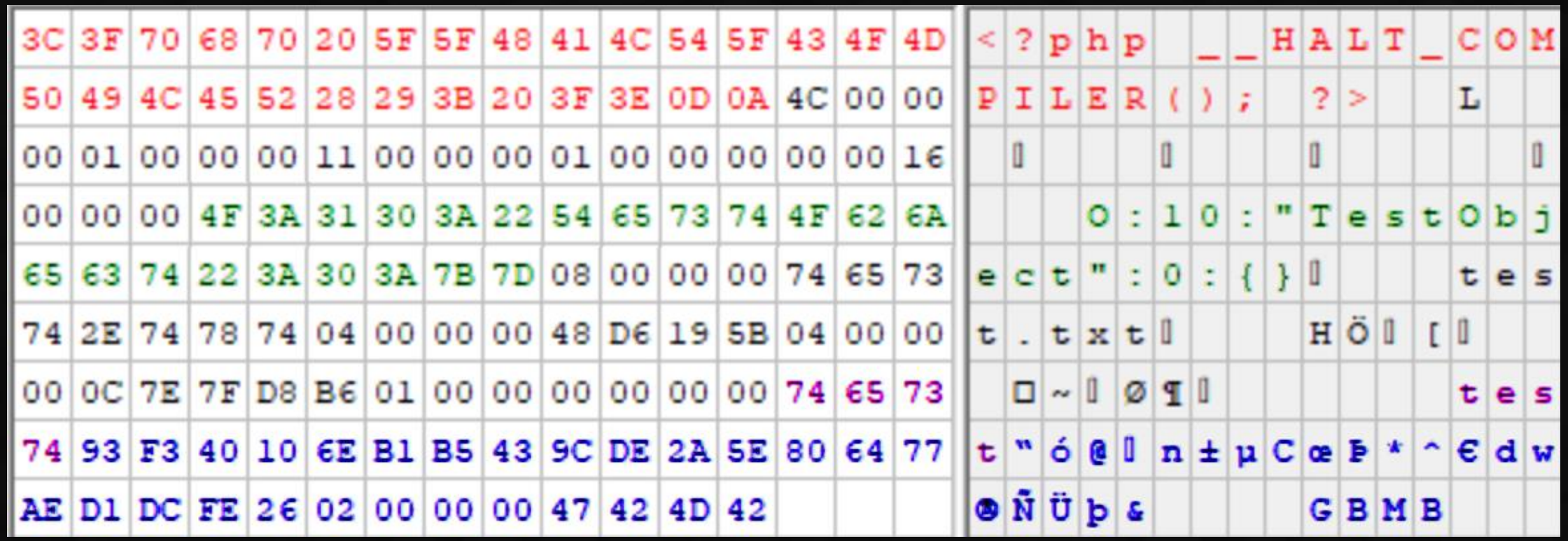
- 3 valid formats
  - Phar
  - Tar
  - Zip
- Must contain
  - Stub
  - Manifest (Serialized metadata)
  - File contents
  - Signature

Phar File Format

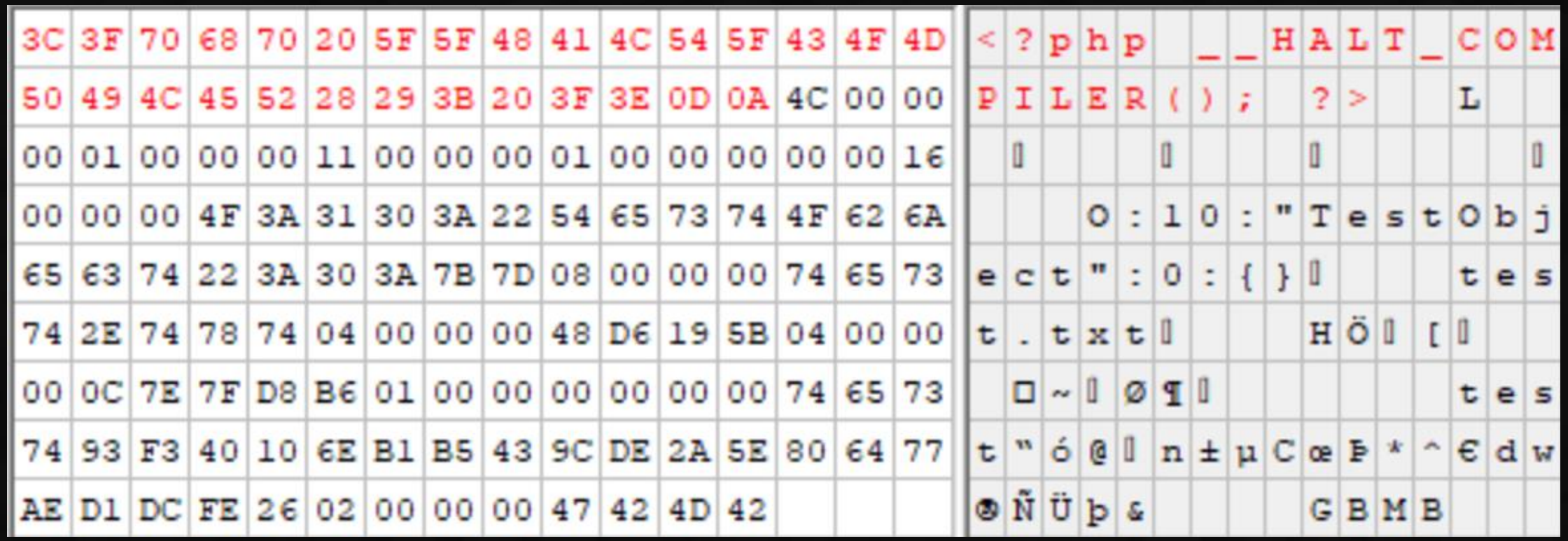
- 3 valid formats
  - Phar
  - Tar
  - Zip
- Must contain
  - Stub
  - Manifest (Serialized metadata)
  - File contents
  - Signature

Phar File Format

- 3 valid formats
  - Phar
  - Tar
  - Zip
- Must contain
  - Stub
  - Manifest (Serialized metadata)
  - File contents
  - Signature



stub	manifest	metadata	contents	signature



3C	3F	70	68	70	20	5F	5F	48	41	4C	54	5F	43	4F	4D
50	49	4C	45	52	28	29	3B	20	3F	3E	0D	0A	4C	00	00
00	01	00	00	00	11	00	00	00	01	00	00	00	00	00	16
00	00	00	4F	3A	31	30	3A	22	54	65	73	74	4F	62	6A
65	63	74	22	3A	30	3A	7B	7D	08	00	00	00	74	65	73
74	2E	74	78	74	04	00	00	00	48	D6	19	5B	04	00	00
00	0C	7E	7F	D8	B6	01	00	00	00	00	00	00	74	65	73
74	93	F3	40	10	6E	B1	B5	43	9C	DE	2A	5E	80	64	77
AE	D1	DC	FE	26	02	00	00	00	47	42	4D	42			

<	?	p	h	p		_	_	H	A	L	T	_	C	O	M
P	I	L	E	R	(	)	;		?	>			L		
	▯				▯				▯						▯
			O	:	1	0	:	"	T	e	s	t	O	b	j
e	c	t	"	:	0	:	{	}	▯				t	e	s
t	.	t	x	t	▯				H	Ö	▯	[	▯		
	□	~	▯	Ø	¶	▯							t	e	s
t	"	ó	@	▯	n	±	µ	C	œ	Þ	*	^	€	d	w
®	Ñ	Ü	þ	&					G	B	M	B			

stub	manifest	metadata	contents	signature

Phar File Format

- By inserting data into the stub we can fake most file formats
- To create a valid Phar archive we must completely control the end of the file, nothing can follow the signature, and the signature must match the contents of the archive

Phar File Format

- 3 valid formats
  - Phar
  - Tar
  - Zip
- Must contain
  - Stub
  - Manifest (Serialized metadata)
  - File contents
  - Signature

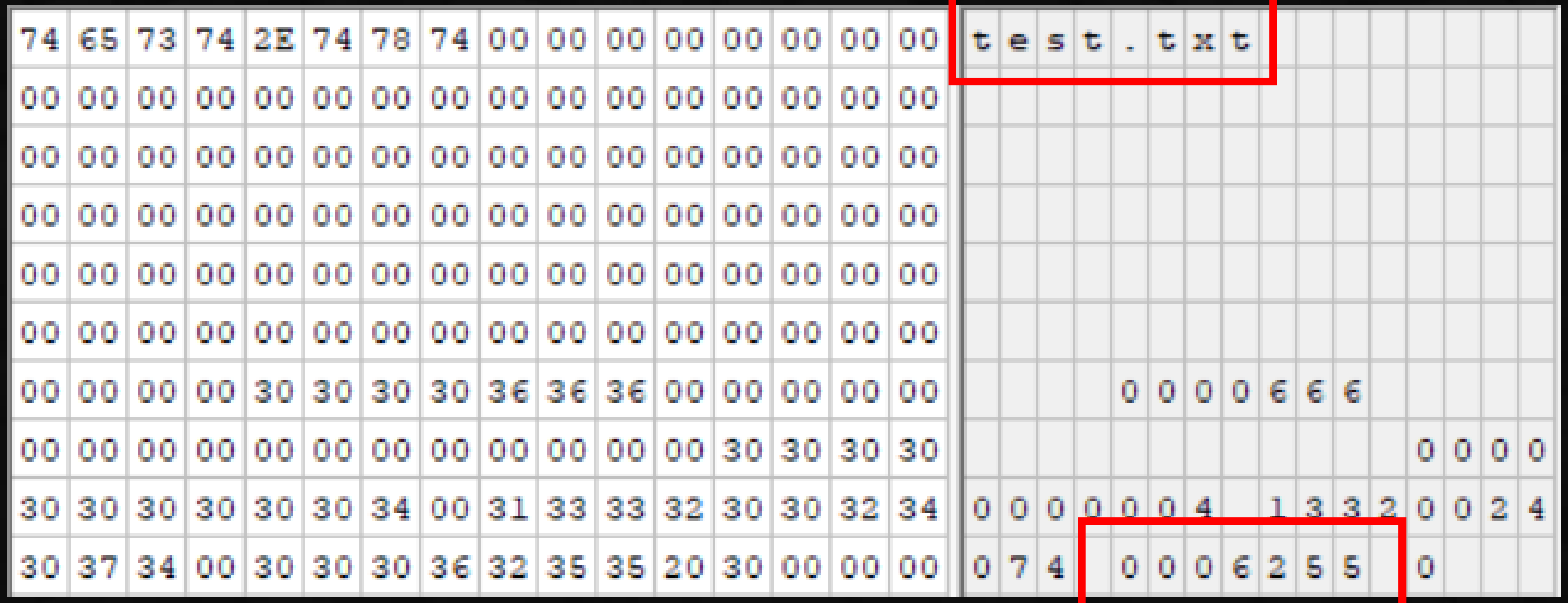
Phar/Tar File Format

- File sizes rounded up to nearest 512 byte size
- Each file preceded by 512 byte header
- First 100 bytes are filename
- 4 byte checksum for file contents
- The end of an archive is marked by at least two consecutive zero-filled records. (Anything after this is ignored)

Phar/Tar File Format

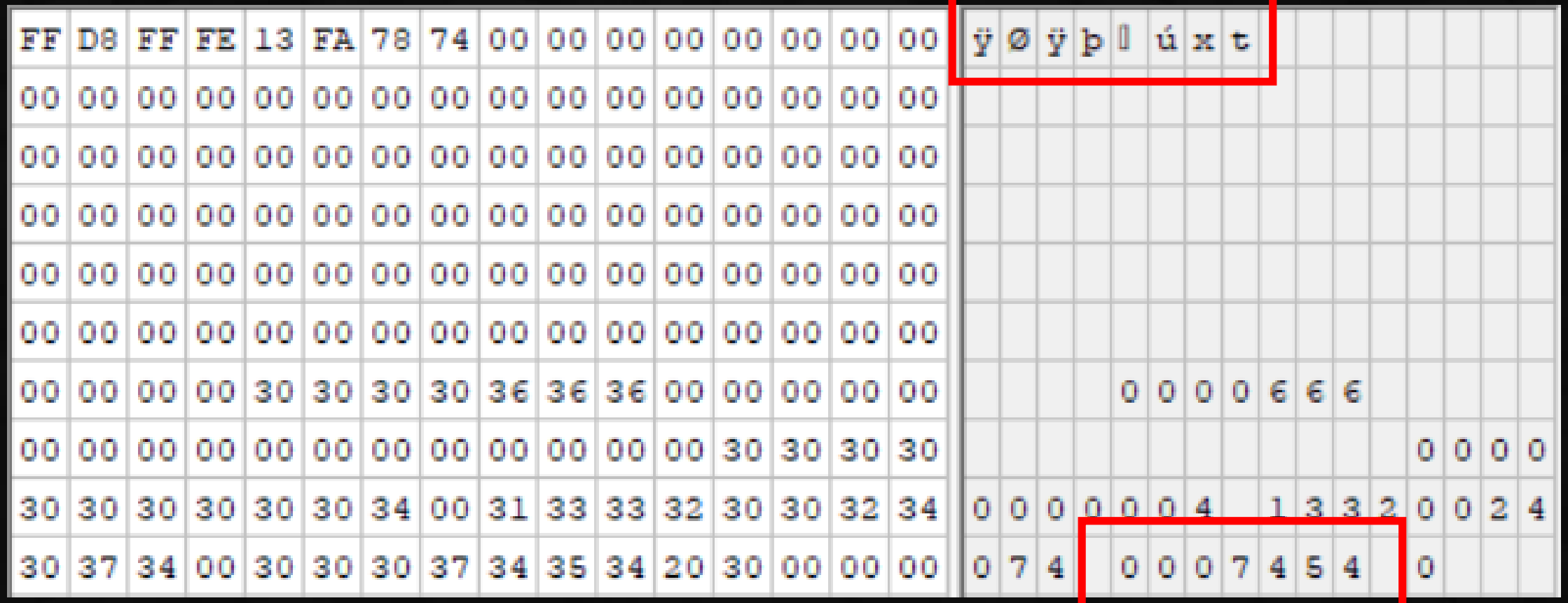
- File sizes rounded up to nearest 512 byte size
- Each file preceded by 512 byte header
- First 100 bytes are filename
- 4 byte checksum for file contents
- The end of an archive is marked by at least two consecutive zero-filled records. (Anything after this is ignored)

Filename



Checksum

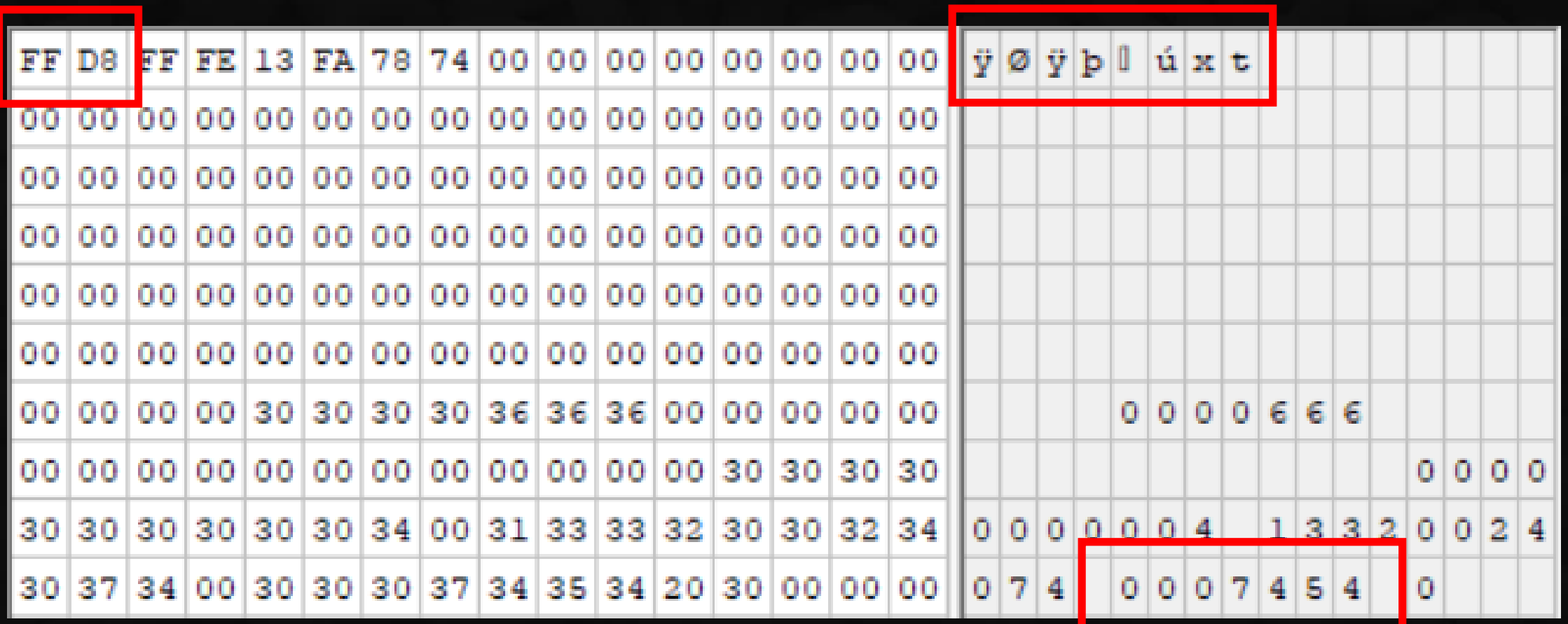
Filename



Checksum

JPEG header

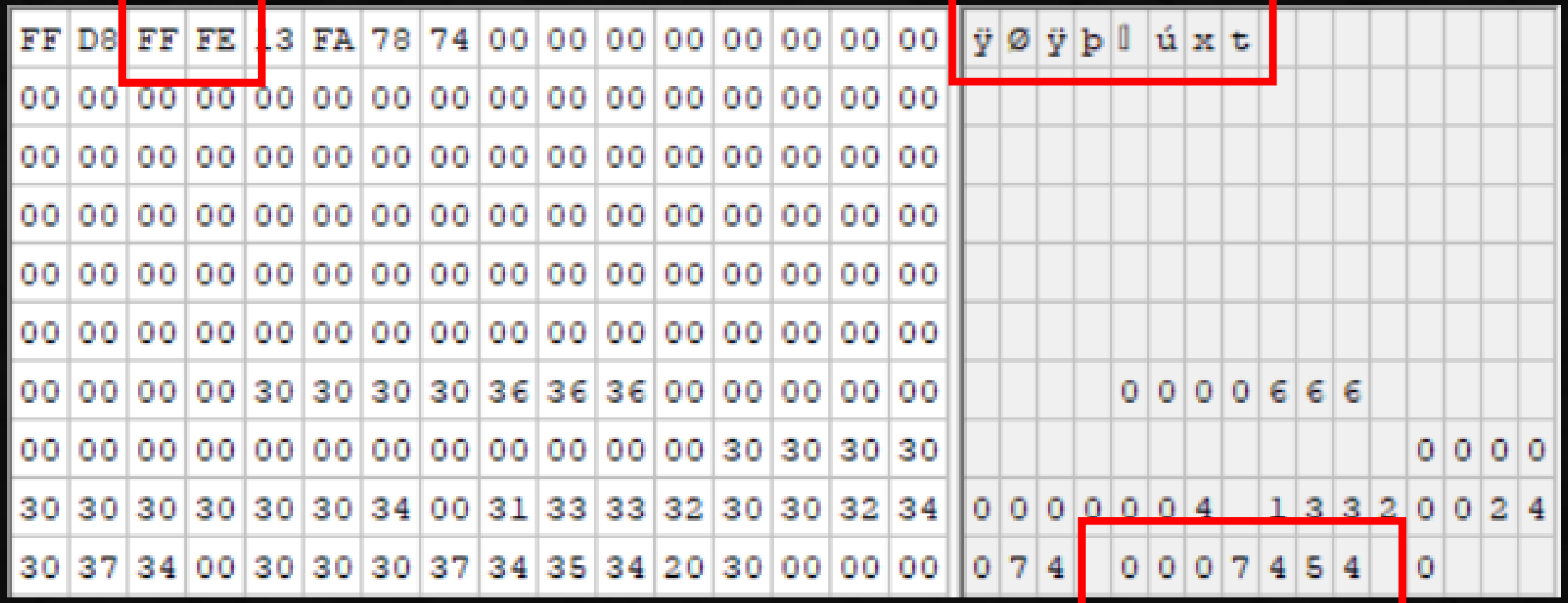
Filename



Checksum

Comment

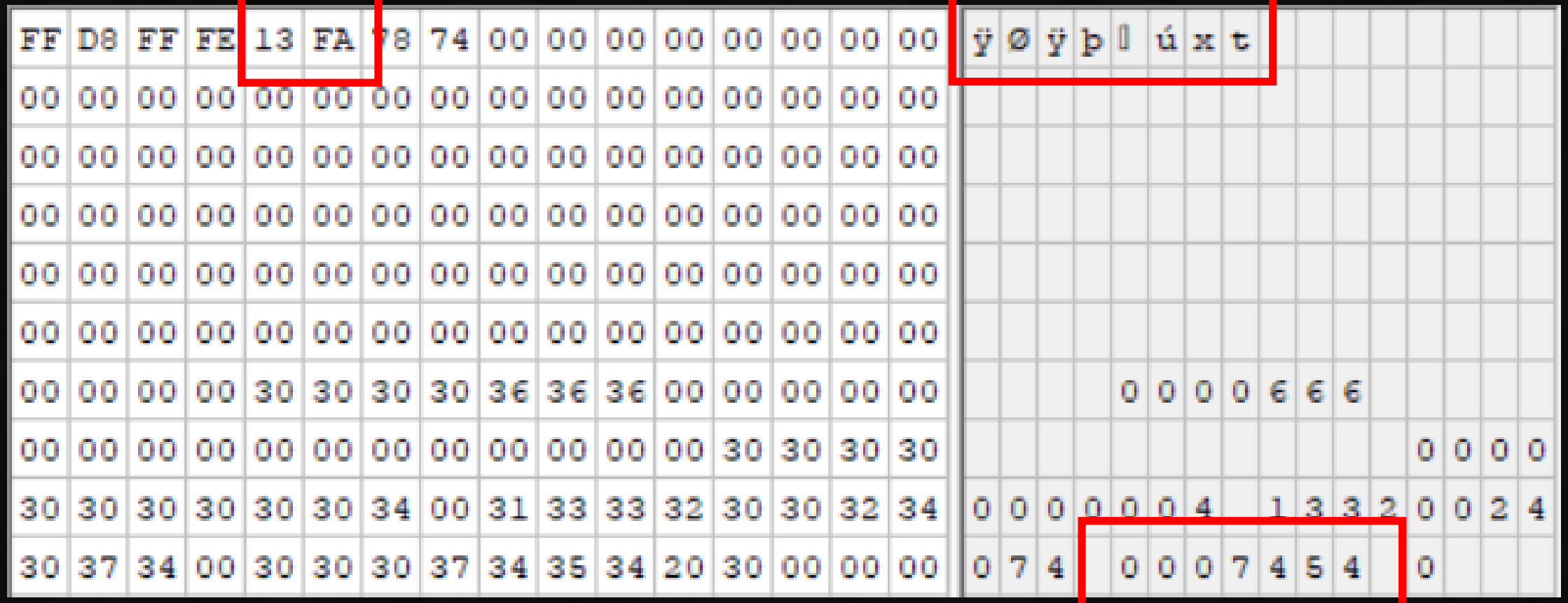
Filename



Checksum

Comment length

Filename



Checksum

Quick Polyglot Demo

Phar Planting

- Upload fake image / polyglot
- Upload temporary file (phpinfo() trick[10]/ brute force[11])
- Session File
  - $_SESSION[‘foo’] = $_POST[‘bar’]
- phar:///proc/self/fd/0 ?
- Log files ?

Identifying Vulnerabilities

- XXE & SSRF well understood
  - NONET libxml option does not block “phar://”
- Use “ftp://” if OOB (direct or DNS) possible
  - Supports all file operations (file_exists, file_put_contents etc..[12])
- Can use “file://” if no OOB
  - Observe behaviour with valid “file://” path

PHPGGC / PHARGGC

- PHPGGC (PHP Generic Gadget Chains)
  - Awesome tool for generating PHP unserialization payloads
  - ysoserial for PHP
- PHARGGC
  - Nicks all the bits from PHPGGC to generate phar payloads
  - Either prepends a given header to the stub or generate jpeg polyglot
  - “phar.read_only” must be set to 0 to write phar archives[13]

PHPGGC / PHARGGC Payloads

- PHP 7.2 depreciates assert($string)
- Replace “assert” with “passthru”
- Consider “Composer\Autoload\includeFile” for PHP code execution
  - Remember “__destruct” chains execute without path context so unfortunately we must know the absolute path of the file to be included

Case Studies

- All examples were found through manual code analysis
- Only one could not have easily been identified with blackbox techniques
- All use a simple file upload to plant the phar archive containing our payload

Case Study A – Typo3

Reported: 9th June 2018
Fixed: 12th July 2018

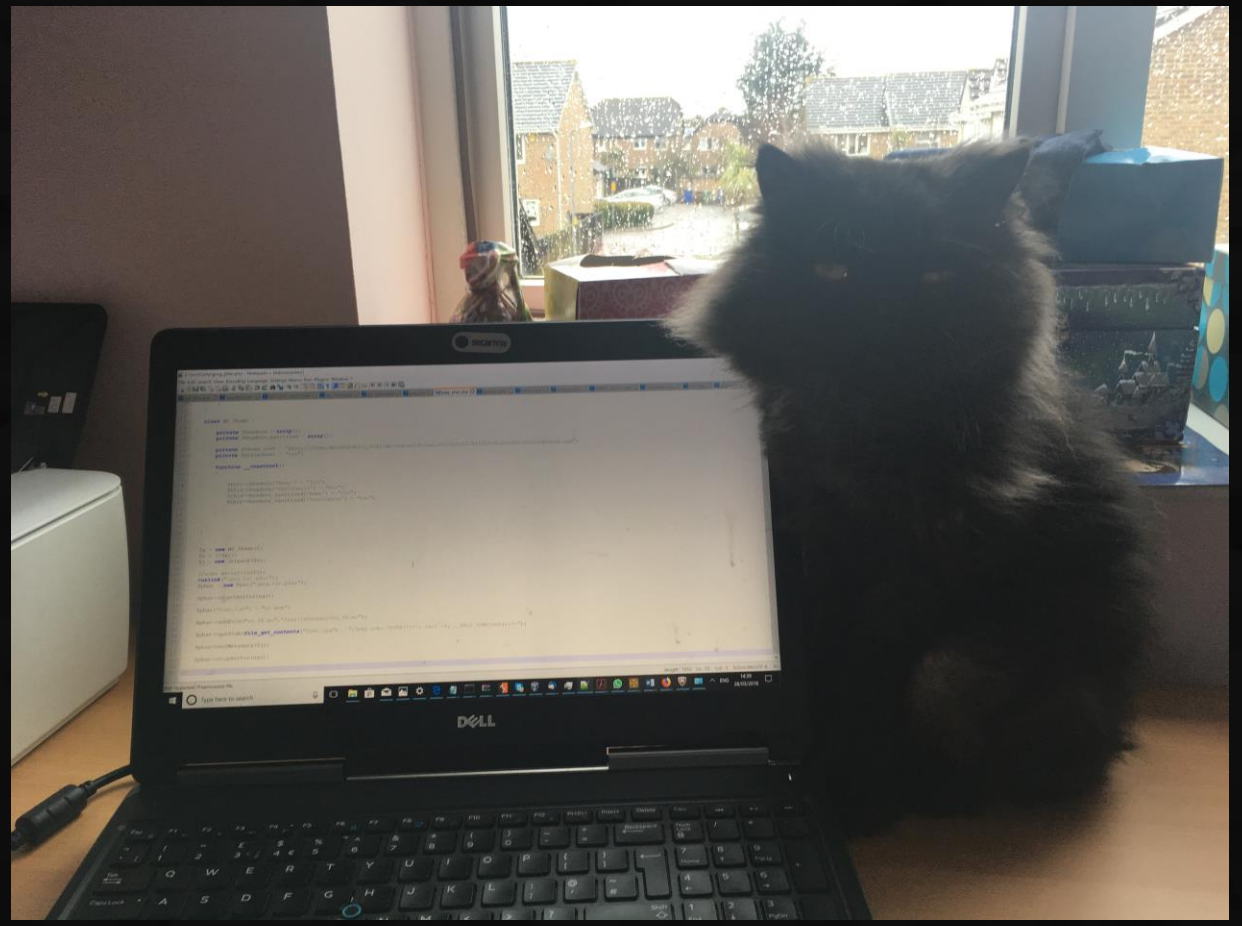
Many thanks to Oliver Hader

Case Study A – Typo3

- There is a vulnerability in link processing which leads to a call to “file_exists” with complete control of the parameter:

```
        } elseif ($containsSlash || $isLocalFile) { // file
(internal)
            $splitLinkParam = explode('?', $link_param);
            if (file_exists(rawurldecode($splitLinkParam[0])) ||
$isLocalFile) {
```

Case Study A – Typo3 – Demo Time



Case Study B - Wordpress

Reported: 28th February 2017 – Not fixed ☹

- There is a subtle vulnerability in thumbnail processing which allows an attacker to reach a “file_exists” call with control of the start of the parameter.

Case Study B - Wordpress

```
function wp_get_attachment_thumb_file( $post_id = 0 ) {
    $post_id = (int) $post_id;
    if ( !$post = get_post( $post_id ) )
        return false;
    if ( !is_array( $imagedata = wp_get_attachment_metadata( $post-
>ID ) ) )
        return false;

    $file = get_attached_file( $post->ID );

    if ( !empty($imagedata['thumb']) &&
    ($thumbfile = str_replace(basename($file), $imagedata['thumb'],
$file)) && file_exists($thumbfile) ) {
```

Case Study B - Wordpress

```
function get_attached_file( $attachment_id, $unfiltered = false ) {
    $file = get_post_meta( $attachment_id, '_wp_attached_file', true
);

    // If the file is relative, prepend upload dir.
    if ( $file && 0 !== strpos( $file, '/' ) && ! preg_match(
'|^.:\\\|', $file ) && ( ( $uploads = wp_get_upload_dir() ) && false
=== $uploads['error'] ) ) {
        $file = $uploads['basedir'] . "/$file";
    }

    if ( $unfiltered ) {
        return $file;
    }
```

Case Study B – Wordpress - Payload

- Before Wordpress 4.9 (November 2017) there existed a path from the "__toString" magic method to attacker controlled input within a call to "create_function" [14]
- Several plugins could be abused to trigger “__toString” from “__destruct”
- After Wordpress 4.9 we need a new payload…

Case Study B – Wordpress - Payload

```php
class Requests_Utility_FilteredIterator extends ArrayIterator {
    /**
     * Callback to run as a filter
     *
     * @var callable
     */
    protected $callback;
    ...
    public function current() {
        $value = parent::current();
        $value = call_user_func($this->callback, $value);
        return $value;
    }
}
```

Requests_Utility_FilteredIterator

- Array iterator which implements property defined callback
- Triggered by any call to foreach() on the object

Case Study B – Wordpress - Payload

```
 * @package          WooCommerce/Classes/Log_Handlers
 */
class WC_Log_Handler_File extends WC_Log_Handler {
    ...
    protected $handles = array();
    ...
    public function __destruct() {
        foreach ( $this->handles as $handle ) {
            if ( is_resource( $handle ) ) {
                fclose( $handle ); // @codingStandardsIgnoreLine.
            }
        }
    }
```

Case Study B – Wordpress – Demo Time

Case Study C – TCPDF (via Contao)

Reported: 24th May 2018

Fixed: imminently?

Case Study C – TCPDF (via Contao)

- TCPDF is a very common library used to render HTML into a PDF

  “used daily by millions of users and included in thousands of CMS and Web applications”
- Exposed to attackers either deliberately or through XSS
- <img> tag handler allows attacker to reach controlled call to “file_exists”
- Very typical path for SSRF

Case Study C – TCPDF (via Contao)

```
    protected function openHTMLTagHandler($dom, $key, $cell) {
        $tag = $dom[$key];
        ...
        // Opening tag
        switch($tag['value']) {
            ...
            case 'img': {
                ...
                            $this->Image($tag['attribute']['src'],
$xpos, $this->y, $iw, $ih, '', $imglink, $align, false, 300, '',
false, false, $border, false, false, true);
```

Case Study C – TCPDF (via Contao)

```
public function Image($file, $x='', $y='', …) {
        ...
        if ($file[0] === '@') {
            // image from string
            $imgdata = substr($file, 1);
        } else { // image file
            if ($file[0] === '*') {
                // image as external stream
                $file = substr($file, 1);
                $exurl = $file;
            }
            // check if is a local file
            if (!@file_exists($file)) {
```

Case Study C – TCPDF – Demo Time

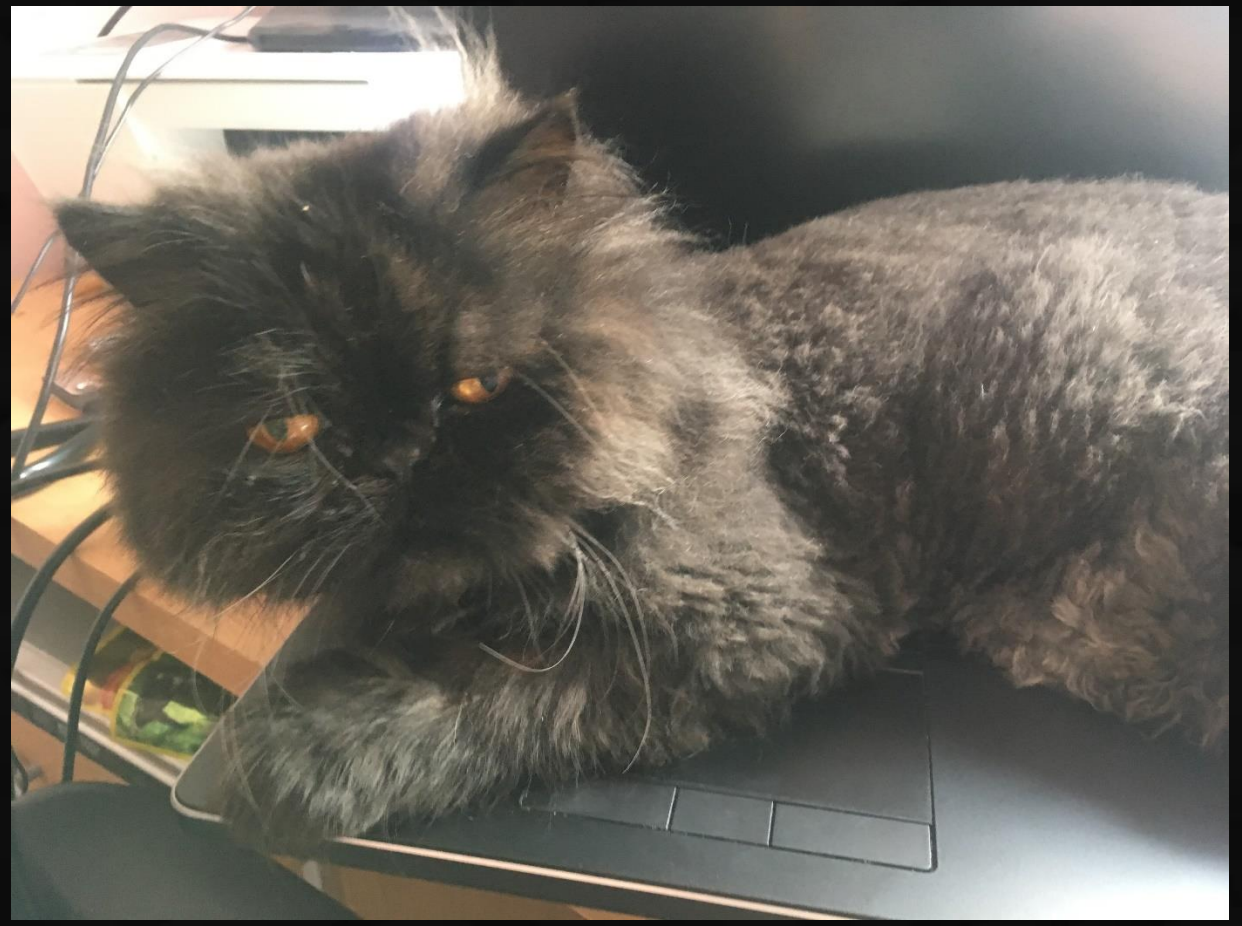
Defence

- Be very careful when passing user controlled values to *ANY* file operations
- Signature detection for Phar archives / polyglots
- Disable PHAR extension during compilation
- Hopefully PHP will change this behaviour in a future version

Take aways

- The increasing complexity (and bloat?) of typical web applications is making them more prone to code re-use attacks.
- By abusing the "phar://" stream handler a number of different vulnerability types can be used to induce unserialization in PHP.
- A class of vulnerabilities that would have previously been considered low impact information disclosure/SSRF issues can potentially be exploited to achieve code execution.
- A number of instances of this class of vulnerability can be easily identified through source code analysis or simple black box methods.

Questions?



References

[1] https://www.owasp.org/images/f/f6/POC2009-ShockingNewsInPHPExploitation.pdf
[2] https://www.owasp.org/images/9/9e/Utilizing-Code-Reuse-Or-Return-Oriented-Programming-In-PHP-Application-Exploits.pdf
[3] http://php.net/manual/en/wrappers.php
[4] https://websec.wordpress.com/2010/02/22/exploiting-php-file-inclusion-overview/
[5] https://sektioneins.de/en/advisories/advisory-032009-piwik-cookie-unserialize-vulnerability.html
[6] http://php.net/manual/en/wrappers.glob.php
[7] http://php.net/manual/en/phar.fileformat.ingredients.php
[8] http://php.net/manual/en/phar.fileformat.stub.php
[9] http://php.net/manual/en/phar.fileformat.manifestfile.php
[10] https://www.insomniasec.com/downloads/publications/LFI%20With%20PHPInfo%20Assistance.pdf
[11] https://truesecdev.wordpress.com/2016/11/09/local-file-inclusion-with-tmp-files/
[12] http://php.net/manual/en/wrappers.ftp.php
[13] http://php.net/manual/en/phar.configuration.php
[14] http://www.slideshare.net/_s_n_t/php-unserialization-vulnerabilities-what-are-we-missing