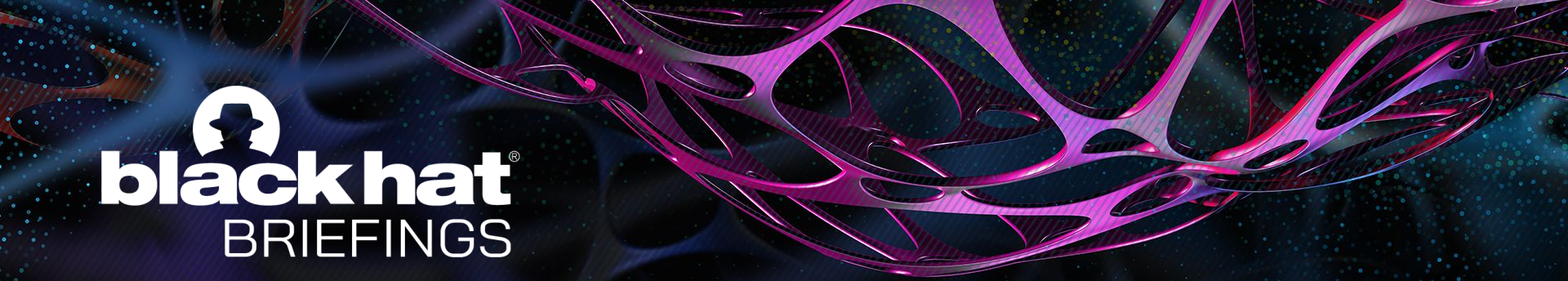


QUACK: Hindering Deserialization Attacks via Static Duck Typing

Speakers: Andreas Kellas, Neophytos Christou

whoarewe

Neophytos Christou

PhD student, Brown University

BROWN

Andreas Kellas

PhD student, Columbia University

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Collaborators: Yaniv David[1], Vasileios Kemerlis[2], Junfeng Yang[1]

Columbia University[1], Brown University[2]

We know that deserialization is dangerous …

We know that deserialization is dangerous …

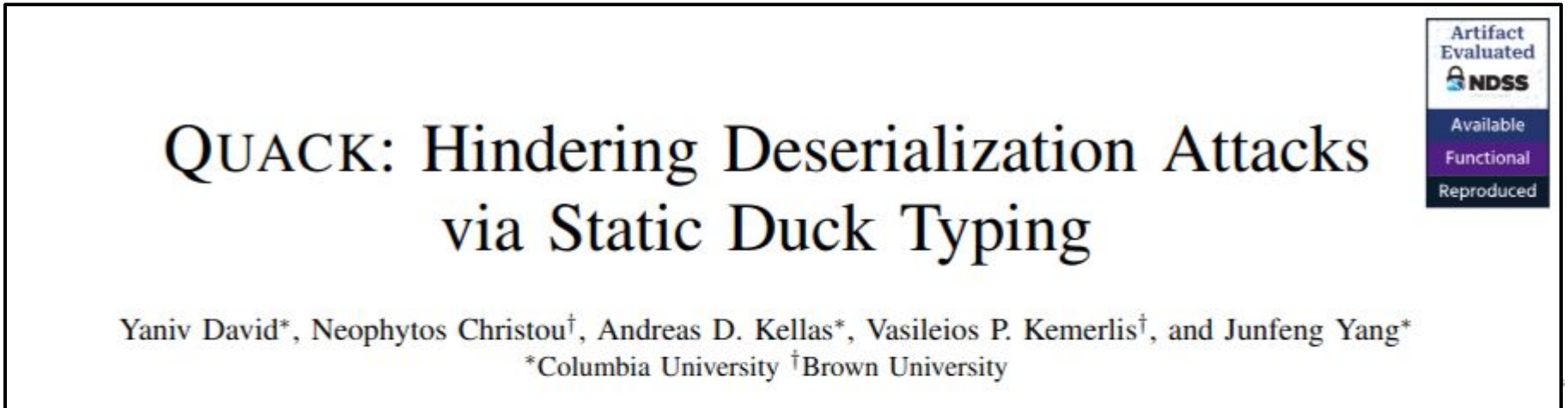
2010: “Utilizing Code Reuse/ROP in PHP Application Exploits” Stefan Esser @ BHUSA

2015: “Marshalling Pickles: how deserializing objects will ruin your day” Frohoff and Lawrence @ AppSecCali

2018: “Automated Discovery of Deserialization Gadget Chains” Ian Haken @ BHUSA

We know that deserialization is dangerous …

… so we set out to mitigate the risks



Goals

Artifact Evaluated NDSS Available Functional Reproduced

QUACK: Hindering Deserialization Attacks via Static Duck Typing

Yaniv David*, Neophytos Christou†, Andreas D. Kellas*, Vasileios P. Kemerlis†, and Junfeng Yang*
*Columbia University †Brown University

Goals

1. Introduce QUACK to the security community

Artifact Evaluated NDSS Available Functional Reproduced

QUACK: Hindering Deserialization Attacks via Static Duck Typing

Yaniv David*, Neophytos Christou†, Andreas D. Kellas*, Vasileios P. Kemerlis†, and Junfeng Yang*
*Columbia University †Brown University

Goals

1. Introduce QUACK to the security community
2. Raise awareness for the risks of deserialization

Artifact Evaluated NDSS
Available
Functional
Reproduced

QUACK: Hindering Deserialization Attacks via Static Duck Typing

Yaniv David*, Neophytos Christou†, Andreas D. Kellas*, Vasileios P. Kemerlis†, and Junfeng Yang*
*Columbia University †Brown University

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

We Know Deserialization is Dangerous

InvoiceShelf <= 1.3.0 - PHP Deserialization

CVE-2024-55556

CVE-2023-30534: Insecure Deserialization in Cacti prior to 1.2.25

Fastly Security Research Team
Fastly Security Research Team, Fastly

Matthew Mathur
Senior Security Researcher, Fastly

MARCH 20, 2024 THREAT INTELLIGENCE

UNPATCHED PHP DESERIALIZATION VULNERABILITY IN ARTICA PROXY

Description

ClipBucket V5 provides open source video hosting with PHP. ClipBucket-v5 Version 2.0 to Version 5.5.1 Revision 199 are vulnerable to PHP Deserialization vulnerability. The vulnerability exists in upload/photo_upload.php within the decode_key function. User inputs were supplied

PHP deserialization attacks and a new gadget chain in Laravel

Posted Tue 13 February 2024
Author Mathieu Farrell

We Know Deserialization is Dangerous

```
unserialize(string $data, array $options = []): mixed
```

unserialize() takes a single serialized variable and converts it back into a PHP value.

Warning Do not pass untrusted user input to **unserialize()** regardless of the **options** value of allowed_classes. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

If you need to unserialize externally-stored serialized data, consider using hash_hmac() for data validation. Make sure data is not modified by anyone but you.

PHP Object Injection (POI)

```php
$obj = unserialize(/* Attacker controlled string */);
```

PHP Object Injection (POI)

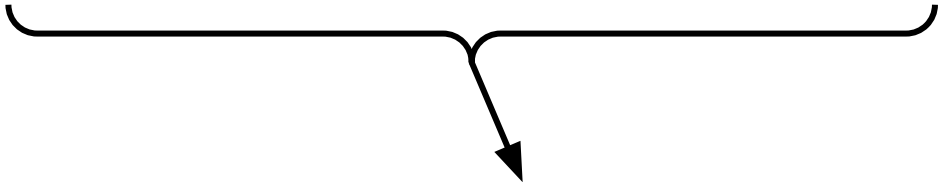
```php
class App {
    public $name;
    public function run() { /* … */ }
}

$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Programmer expects **$obj** to be an **App...**

PHP Object Injection (POI)

```
O:3:"App":1:{s:4:"name";s:8:"MyWebApp";}
```



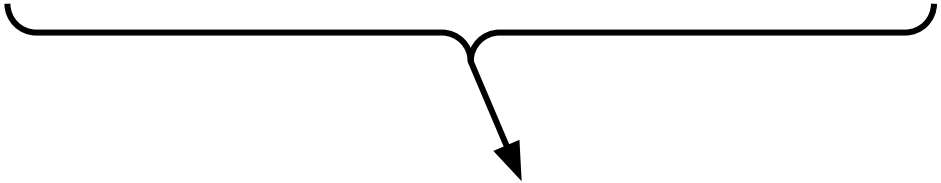
```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

```
App Object
(
  [name] => MyWebApp
)
```

Programmer expects **$obj** to be an **App**...

PHP Object Injection (POI)

```
O:3:"App":1:{s:4:"name";s:8:"MyWebAppBlackHat";}
```



```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

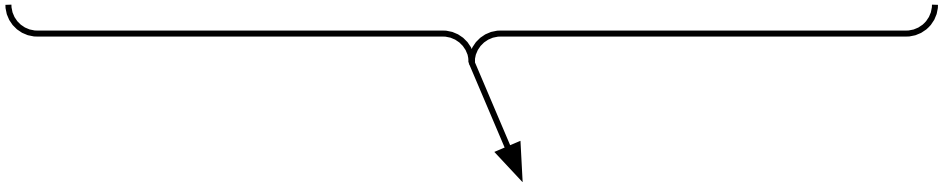
```
App Object
(
    [name] => BlackHat
)
```

Programmer expects **$obj** to be an **App**...

... but the attacker controls its *properties*...

PHP Object Injection (POI)

```
O:3:"AppFoo":1:{s:3:"namebar";s:3:"MyWebAppBAZ";}
```



```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

```
Foo Object
(
    [bar] => BAZ
)
```

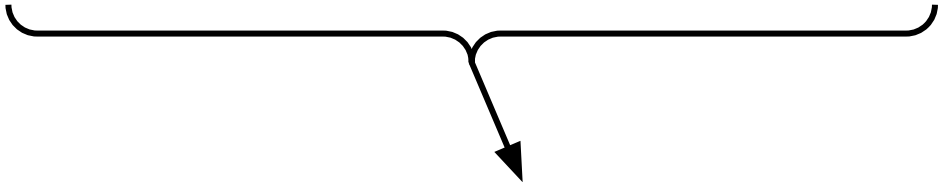
Programmer expects **$obj** to be an **App**...

... but the attacker controls its *properties*...

... and its whole *type*

PHP Object Injection (POI)

```
O:3:"AppFoo":1:{s:3:"name";O:3:"Bar":1:{...}};
```



```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

```
Foo Object
(
    [name] => Bar Object
              (...)
)
```

Programmer expects **$obj** to be an **App**...

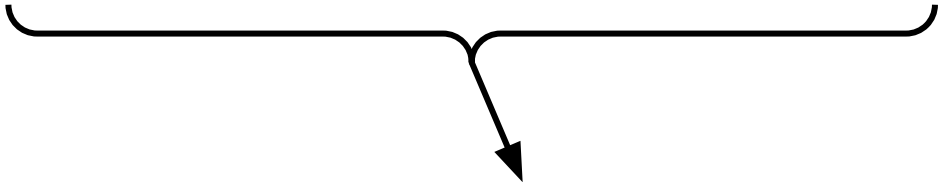
... but the attacker controls its *properties*...

... and its whole *type*

They can create *nested* objects of *any loaded class*

PHP Object Injection (POI)

```
O:3:"AppFoo":1:{s:3:"name";O:3:"Bar":1:{...}};
```



```
$obj = unserialize(/* Attacker controlled string */);
$obj→run();
```

Attacker's goal:
Change the control flow to execute malicious functionality

Programmer expects **$obj** to be an **App**...

... but the attacker controls its *properties*...

... and its whole *type*

They can create *nested* objects of *any loaded class*

Manipulating Control Flow

```php
class CommandExecutor {
    public $command;
    public function run() { system($this->command); }
}
```

```php
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Manipulating Control Flow

```
class CommandExecutor {
    public $command;
    public function run() { system($this→command); }
}
```

Attacker's goal:
Change the control flow to
execute malicious functionality

```
$obj = unserialize(/* Attacker controlled string */);
$obj→run();
```

Manipulating Control Flow

```php
class CommandExecutor {
    public $command;
    public function run() { system($this->command); }
}
```

Attacker's goal:
Change the control flow to
execute malicious functionality

```
O:15:"CommandExecutor":1:{s:7:"command";s:13:"echo "pwned!"";}
```

```php
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Manipulating Control Flow

```
class CommandExecutor {
    public $command;
    public function __wakeup() { system($this->command); }
}
```

Attacker's goal:
Change the control flow to
execute malicious functionality

```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Manipulating Control Flow

```php
class CommandExecutor {
    public $command;
    public function __wakeup() { system($this->command); }
}
```

Attacker's goal:
Change the control flow to
execute malicious functionality

```php
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Magic methods execute *automatically* in certain circumstances

Manipulating Control Flow

```
class CommandExecutor {
    public $command;
    public function __wakeup() { system($this->command); }
}
```

Attacker's goal:
Change the control flow to
execute malicious functionality

```
O:15:"CommandExecutor":1:{s:7:"command";s:13:"echo "pwned!"";}
```

```
$obj = unserialize(/* Attacker controlled string */);
$obj->run();
```

Magic methods execute *automatically* in certain circumstances

Manipulating Control Flow

```
class CommandExecutor {
    public $command;
    public function __wakeup() { /* … */ }
}
```

```
$obj = unserialize(/* Attacker controlled string */);
$obj→run();
```

Attacker's goal:
Change the control flow to
execute malicious functionality

What if the attacker can't find one method that achieves both?

Property Oriented Programming (POP)

Exploit Idea:

Property Oriented Programming (POP)

Exploit Idea:

1. Change initial control flow by creating a top-level object with an appropriate magic method

Property Oriented Programming (POP)

Exploit Idea:

1. Change initial control flow by creating a top-level object with an appropriate magic method

2. Execute malicious functionality by chaining method calls
   - Recursively set object properties to other objects

Property Oriented Programming (POP)

Exploit Idea:

1. Change initial control flow by creating a top-level object with an appropriate magic method

2. Execute malicious functionality by chaining method calls
   - Recursively set object properties to other objects

"Utilizing Code Reuse/ROP in PHP Application Exploits"
Stefan Esser @ BHUSA 2010

Property Oriented Programming

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class graph {
  function init( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Oriented Programming

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class graph {
  function init( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

Property Oriented Programming

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class graph {
  function init( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

Property Oriented Programming

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class graph {
  function init( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Orient

```
lock Object (
    [key] => tree Object (
        [children] => recordset_walk Object (
            [record] => echo `whoami`
            [callback] => system
        )
    )
)
```

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Orient

```
lock Object (
    [key] => tree Object (
        [children] => recordset_walk Object (
            [record] => echo `whoami`
            [callback] => system
        )
    )
)
```

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Orient

```
lock Object (
    [key] => tree Object (
        [children] => recordset_walk Object (
            [record] => echo `whoami`
            [callback] => system
        )
    )
)
```

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  public $children;
  function __toString( ) {
    foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Orient

```
lock Object (
    [key] => tree Object (
        [children] => recordset_walk Object (
            [record] => echo whoami
            [callback] => system
        )
    )
)
```

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Property Orient

```
lock Object (
    [key] => tree Object (
        [children] => recordset_walk Object (
            [record] => echo `whoami`
            [callback] => system
        )
    )
)
```

```
class qtype_ddwtos {
  function import($input)
  {
    unserialize($input);
  }
}
```

```
class lock {
  public $key;
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  public $children;
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  public $callback;
  public $record;
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

Moodle POI Vulnerability

CVE-2018-14630 Detail

moodle before versions 3.5.2, 3.4.5, 3.3.8, 3.1.14 is vulnerable to an XML import of ddwtos could lead to intentional remote code execution. When importing legacy 'drag and drop into text' (ddwtos) type quiz questions, it was possible to inject and execute PHP code from within the imported questions, either intentionally or by importing questions from an untrusted source.

https://sec-consult.com/vulnerability-lab/advisory/remote-code-execution-php-unserialize-moodle-open-source-learning-platform-cve-2018-14630

Moodle POI Vulnerability

```
foreach ($data['#']['answer'] as $answerxml) {
    $ans = $format->import_answer($answerxml);
    $options = unserialize($ans->feedback['text']);
    $question->choices[] = array(
        'answer' => $ans->answer,
        'choicegroup' => $options->draggroup,
        'infinite' => $options->infinite,
    );
}
```

Moodle POI Vulnerability

```
foreach ($data['#']['answer'] as $answerxml) {
    $ans = $format->import_answer($answerxml);
    $options = unserialize($ans->feedback['text']);
    $question->choices[] = array(
        'answer' => $ans->answer,
        'choicegroup' => $options->draggroup,
        'infinite' => $options->infinite,
    );
}
```

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

Roadmap

Background

PHP deserialization exploits

How do PHP deserialization exploits work?

Deserialization exploit demo

How do existing defenses work?

POI Mitigations

Simple(r) representations:

Works only for simple data structures, not complex objects

Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

POI Mitigations

HMACs: Works only if the serialized object was produced by the application

If you need to unserialize externally-stored serialized data, consider using hash_hmac() for data validation. Make sure data is not modified by anyone but you.

POI Mitigations

`allowed_classes` option to `unserialize`: **works!**

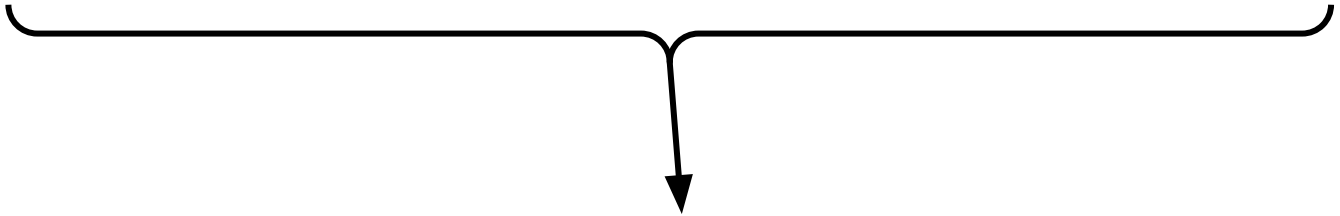
```
unserialize(string $data, array $options = [])
```

Description

Either an array of class names which should be accepted, **`false`** to accept no classes, or **`true`** to accept all classes. If this option is defined and **unserialize()** encounters an object of a class that isn't to be accepted, then the object will be instantiated as __PHP_Incomplete_Class instead. Omitting this option is the same as defining it as **`true`**: PHP will attempt to instantiate objects of any class.

Restricting Allowed Classes

```
O:15:"CommandExecutor":1:{s:7:"command";s:13:"echo "pwned!"";}
```



```php
$obj = unserialize(/* Attacker controlled string */,
    ['allowed_classes' ⇒ ['App']])
$obj→run();
```

```
PHP Fatal error:  Uncaught Error: The script tried to call a method
on an incomplete object. Please ensure that the class definition "Co
mmandExecutor" of the object you are trying to operate on was loaded
```

Restricting Allowed Classes

- **allowed_classes** is barely used
  - ~0.1% of PHP deserialization invocations in Github use it
- Why?
  - Developers are not aware of the dangers of deserialization
  - Tedious to manually deduce allowed classes for each deserialization call

Key Takeaways

Background

PHP deserialization exploits

Deserialization vulnerabilities are exploitable because attackers have access to more classes than the developer intended

Key Takeaways

Background

PHP deserialization exploits

Deserialization vulnerabilities are exploitable because attackers have access to more classes than the developer intended

Existing mitigations help, but not fully

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

QUACK's Objective

Automatically restrict each unserialize call to create *only programmer-intended classes*

QUACK: High-level Approach

```
class graph {
  function init( ) {...}
}
```

```
class base_setting {
  function get_name( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class course {
  function filter( ) {...}
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
class tree {
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

```
class lock {
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

QUACK: High-level Approach

1. Determine set of *available classes* loaded at deserialization point

```
class course {
  function filter( ) {...}
}
```

```
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
class lock {
  function __destruct( ) {
    echo “Key: $this->key”;
  }
}
```

```
class tree {
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class student_enrollment {
  function get_samples( ) {... }
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class recordset_walk {
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

QUACK: High-level Approach

```
class graph {
  function init( ) {...}
}
```

```
class base_setting {
  function get_name( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class course {
  function filter( ) {...}
}
```

```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

```
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
class tree {
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

2. Infer the set of *possible classes* based on how deserialized object is used (static duck typing)

QUACK: High-level Approach

```
class graph {
  function init( ) {...}
}
```

```
class base_setting {
  function get_name( ) {...}
}
```

```
class filter_data {
  function filter( ) {...}
}
```

```
class course {
  function filter( ) {...}
}
```

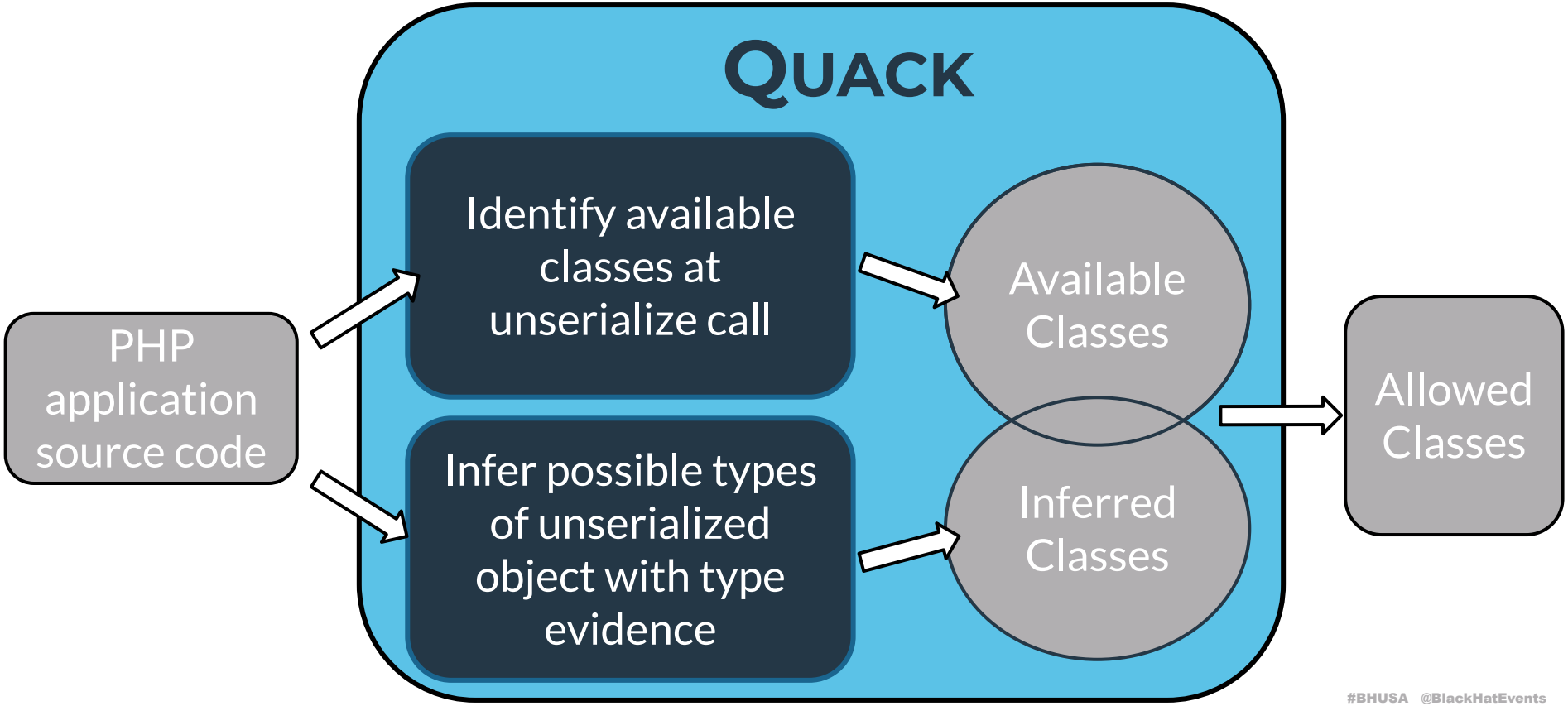
```
class qtype_ddwtos_choice {
  function choice_group( ) {...}
}
```

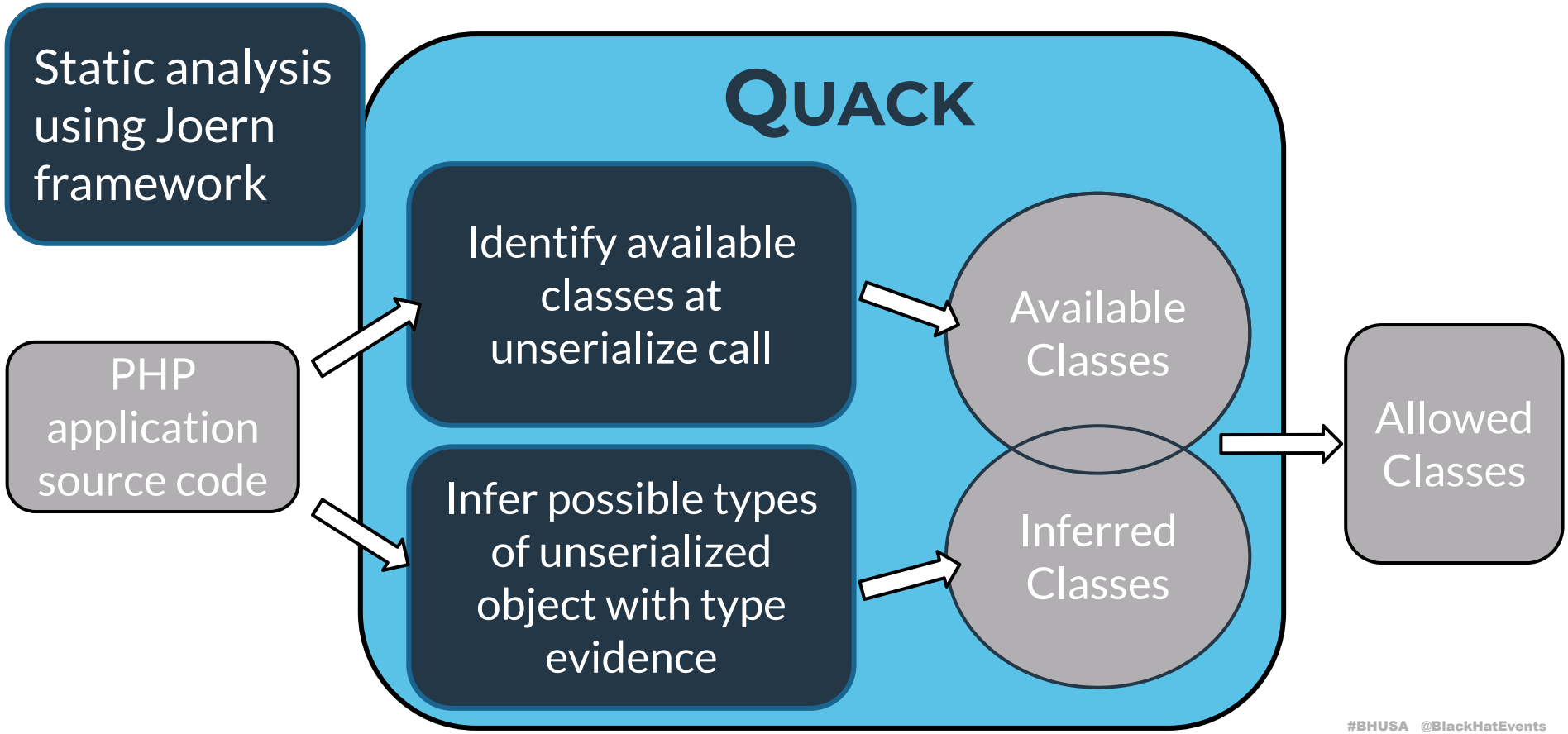
```
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
class tree {
  function __toString( ) {
   foreach ($this->children){...}
  }
}
```

```
class recordset_walk {
  function current( ) {
    call_user_func($this->callback,
$this->record);
  }
}
```

3. Restrict the unserialize call to allow classes that are both *available* and *possible* (leaves only the likely intended classes)


QUACK
PHP application source code
Identify available classes at unserialize call
Infer possible types of unserialized object with type evidence
Available Classes
Inferred Classes
Allowed Classes


Static analysis using Joern framework
QUACK
Identify available classes at unserialize call
Infer possible types of unserialized object with type evidence
PHP application source code
Available Classes
Inferred Classes
Allowed Classes

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Inferring Available Classes

```
include qtype_ddwtos.php
class course {...}
```

```
class qtype_ddwtos_choice {...}
```

```
class graph {...}
```

```
include tree.php
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
include qtype_ddwtos_choice.php
include recordset_walk.php
class tree {...}
```

Available Classes

qtype_ddwtos

```
include qtype_ddwtos.php
class lock {...}
```

```
class filter_data {...}
```

```
class recordset_walk {...}
```

Inferring Available Classes

```
include qtype_ddwtos.php
class course {...}
```

```
class qtype_ddwtos_choice {...}
```

```
class graph {...}
```

```
include tree.php
class qtype_ddwtos {
  function import($input) {
    unserialize($input);
  }
}
```

```
include qtype_ddwtos_choice.php
include recordset_walk.php
class tree {...}
```

Available Classes

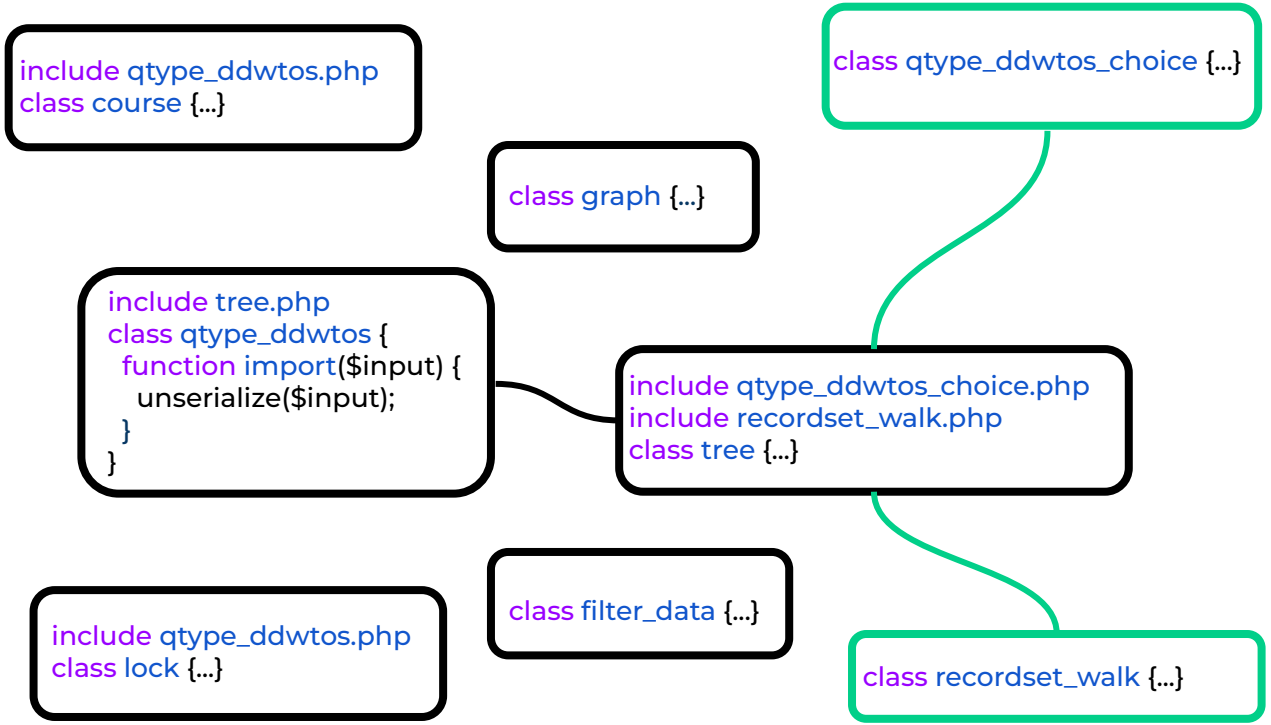
qtype_ddwtos
tree

```
include qtype_ddwtos.php
class lock {...}
```

```
class filter_data {...}
```

```
class recordset_walk {...}
```

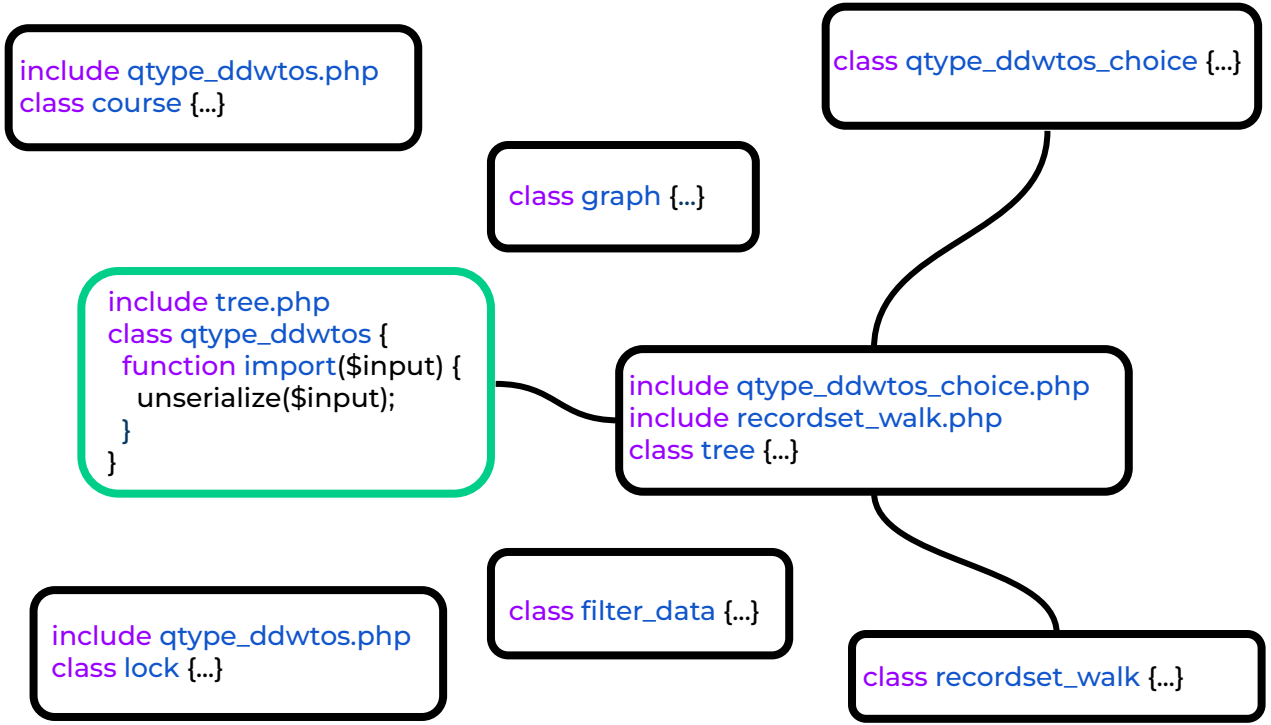
Inferring Available Classes



Available Classes

qtype_ddwtos
tree
qtype_ddwtos_choice
recordset_walk

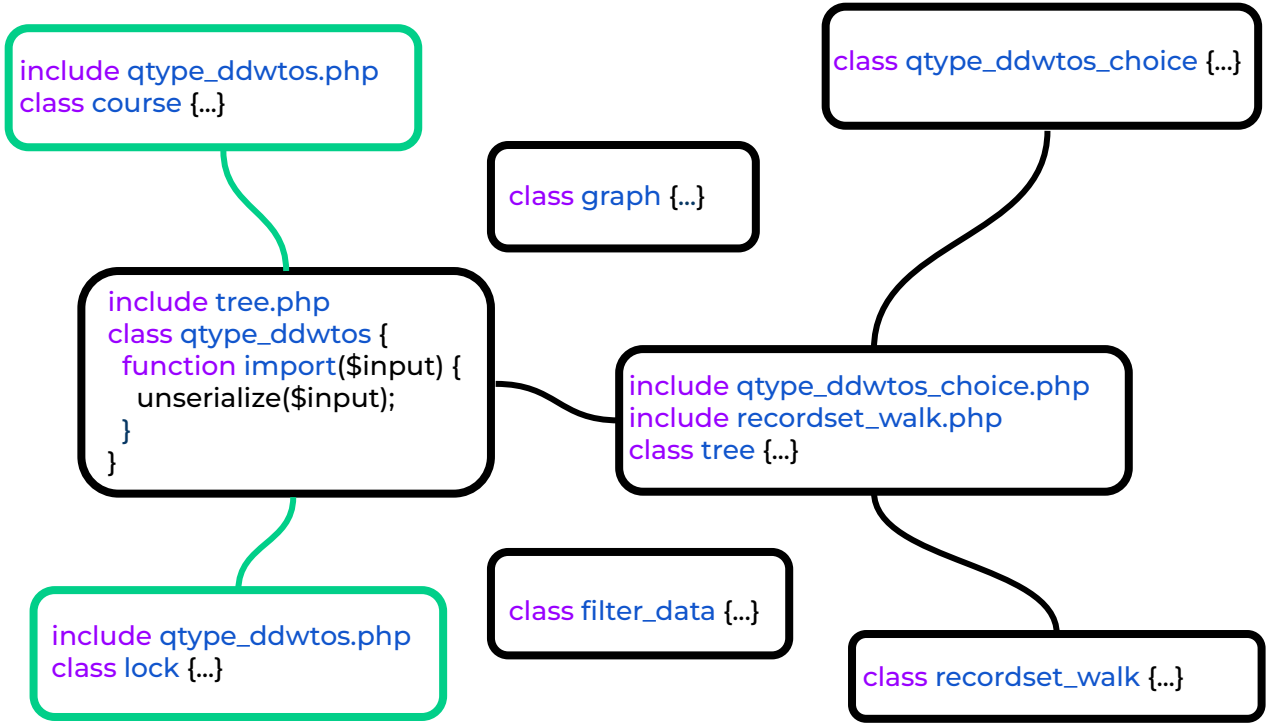
Inferring Available Classes



Available Classes

qtype_ddwtos
tree
qtype_ddwtos_choice
recordset_walk

Inferring Available Classes



Available Classes

qtype_ddwtos
tree
qtype_ddwtos_choice
recordset_walk
course
lock

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Duck Typing

```
class Duck {
    public function swim() {
        echo "Duck swimming\n";
    }

    public function fly() {
        echo "Duck flying\n";
    }
}

class Whale {
    public function swim() {
        echo "Whale swimming\n";
    }
}
```

Dynamic languages: An object is of a given type if it has the methods/properties required by that type

```
$animal->swim();
$animal->fly();
```

Duck Typing

```php
class Duck {
    public function swim() {
        echo "Duck swimming\n";
    }

    public function fly() {
        echo "Duck flying\n";
    }
}

class Whale {
    public function swim() {
        echo "Whale swimming\n";
    }
}
```

Dynamic languages: An object is of a given type if it has the methods/properties required by that type

QUACK: *static* duck-typing-based *type inference rules*

```php
$animal->swim();
$animal->fly();
```

Type Inference Rules: Class Methods

```php
class Duck {
    public function swim() {}
    public function fly() {}
}

class Whale {
    public function swim() {}
}

$animal = unserialize($input);
$animal->swim();
$animal->fly();
```

Type Inference Rules: Class Methods

```php
class Duck {
    public function swim() {}
    public function fly() {}
}

class Whale {
    public function swim() {}
}

$animal = unserialize($input);
$animal->swim();
$animal->fly();
```

Type: **Duck** | **Whale**

- Reason: **swim** method
- Node: **$animal**

Type Inference Rules: Class Methods

```php
class Duck {
    public function swim() {}
    public function fly() {}
}

class Whale {
    public function swim() {}
}

$animal = unserialize($input);
$animal->swim();
$animal->fly();
```

Type: **Duck** | **Whale**
- Reason: **swim** method
- Node: **$animal**

Type: **Duck**
- Reason: **fly** method
- Node: **$animal**

Type Inference Rules: Class Methods

```php
class Duck {
    public function swim() {}
    public function fly() {}
}

class Whale {
    public function swim() {}
}

$animal = unserialize($input);
$animal->swim();
$animal->fly();
```

Type: **Duck** | **Whale**
- Reason: **swim** method
- Node: **$animal**

Type: **Duck**
- Reason: **fly** method
- Node: **$animal**

Possible classes: **Duck**

Type Inference Rules: Class Properties

```php
class Duck {
    public $feather_color;
}

class Whale {
    public $flippers;
}

$animal = unserialize($object);
echo "This duck's feathers are $animal->feather_color";
```

Type Inference Rules: Class Properties

```php
class Duck {
    public $feather_color;
}

class Whale {
    public $flippers;
}

$animal = unserialize($object);
echo "This duck's feathers are $animal->feather_color";
```

Type: **Duck**

- Reason: **feather_color** property
- Node: **$animal**

Type Inference Rules: Class Properties

```php
class Duck {
    public $feather_color;
}

class Whale {
    public $flippers;
}

$animal = unserialize($object);
echo "This duck's feathers are $animal->feather_color";
```

Type: **Duck**

- Reason: **feather_color** property
- Node: **$animal**

Possible classes: **Duck**

Type Inference Rules: Argument Type

```
class Duck {}

class Whale {}

function somefunc(Duck $duck) {}

$animal = unserialize($object);
somefunc($animal);
```

Type Inference Rules: Argument Type

```
class Duck {}

class Whale {}

function somefunc(Duck $duck) {}

$animal = unserialize($object);
somefunc($animal);
```

Type: **Duck**

- Reason: 1st argument to **somefunc**
- Node: **$animal**

Type Inference Rules: Argument Type

```
class Duck {}

class Whale {}

function somefunc(Duck $duck) {}

$animal = unserialize($object);
somefunc($animal);
```

Type: **Duck**

- Reason: 1st argument to **somefunc**
- Node: **$animal**

Possible classes: **Duck**

Type Inference Rules: Known Operators

```
class Duck {}

class Whale {}

function is_it_duck() {
    $animal = unserialize($object);
    if ($animal instanceof Duck) {
        echo "It's a duck!\n"
    }
}
```

Type Inference Rules: Known Operators

```
class Duck {}

class Whale {}

function is_it_duck() {
    $animal = unserialize($object);
    if ($animal instanceof Duck) {
        echo "It's a duck!\n"
    }
}
```

Type: **Duck**

- Reason: **instanceof**
- Node: **$animal**

Type Inference Rules: Known Operators

```
class Duck {}

class Whale {}

function is_it_duck() {
    $animal = unserialize($object);
    if ($animal instanceof Duck) {
        echo "It's a duck!\n"
    }
}
```

Type: **Duck**
- Reason: **instanceof**
- Node: **$animal**

Possible classes: **Duck**

Nested Classes

```php
class Human {
    public $best_friend;
    public function sing() {
        echo "Human singing\n";
    }
}

class Cat {
    public function meow() {
        echo "Cat meowing\n";
    }
}

class Dog {
    public function bark() {
        echo "Dog barking\n";
    }
}

$hacker = unserialize($object);
$hacker->sing();
$hacker->best_friend->bark();
```

Nested Classes

```php
class Human {
    public $best_friend;
    public function sing() {
        echo "Human singing\n";
    }
}

class Cat {
    public function meow() {
        echo "Cat meowing\n";
    }
}

class Dog {
    public function bark() {
        echo "Dog barking\n";
    }
}

$hacker = unserialize($object);
$hacker->sing();
$hacker->best_friend->bark();
```

Type: **Human**

- Reason: **sing** method
- Node: **$hacker**

Nested Classes

```php
class Human {
    public $best_friend;
    public function sing() {
        echo "Human singing\n";
    }
}

class Cat {
    public function meow() {
        echo "Cat meowing\n";
    }
}

class Dog {
    public function bark() {
        echo "Dog barking\n";
    }
}

$hacker = unserialize($object);
$hacker->sing();
$hacker->best_friend->bark();
```

Type: **Human**
- Reason: **sing** method
- Node: **$hacker**

Type: **Human**
- Reason: **best_friend** property
- Node: **$hacker**

Nested Classes

```php
class Human {
    public $best_friend;
    public function sing() {
        echo "Human singing\n";
    }
}

class Cat {
    public function meow() {
        echo "Cat meowing\n";
    }
}

class Dog {
    public function bark() {
        echo "Dog barking\n";
    }
}

$hacker = unserialize($object);
$hacker->sing();
$hacker->best_friend->bark();
```

Type: **Human**
- Reason: **sing** method
- Node: **$hacker**

Type: **Human**
- Reason: **best_friend** property
- Node: **$hacker**

Type: **Dog**
- Reason: **bark** method
- Node: **$hacker->best_friend**

Nested Classes

```php
class Human {
    public $best_friend;
    public function sing() {
        echo "Human singing\n";
    }
}

class Cat {
    public function meow() {
        echo "Cat meowing\n";
    }
}

class Dog {
    public function bark() {
        echo "Dog barking\n";
    }
}

$hacker = unserialize($object);
$hacker->sing();
$hacker->best_friend->bark();
```

Type: **Human**
- Reason: **sing** method
- Node: **$hacker**

Type: **Human**
- Reason: **best_friend** property
- Node: **$hacker**

Type: **Dog**
- Reason: **bark** method
- Node: **$hacker->best_friend**

Possible classes: **Human, Dog**

Roadmap

QUACK

Mitigating deserialization exploits with *static duck typing*

- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Roadmap

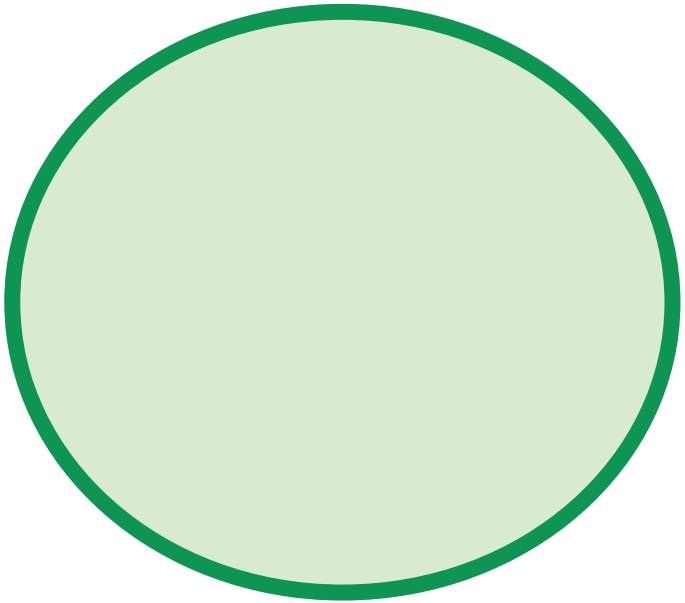
QUACK

Mitigating deserialization exploits with *static duck typing*

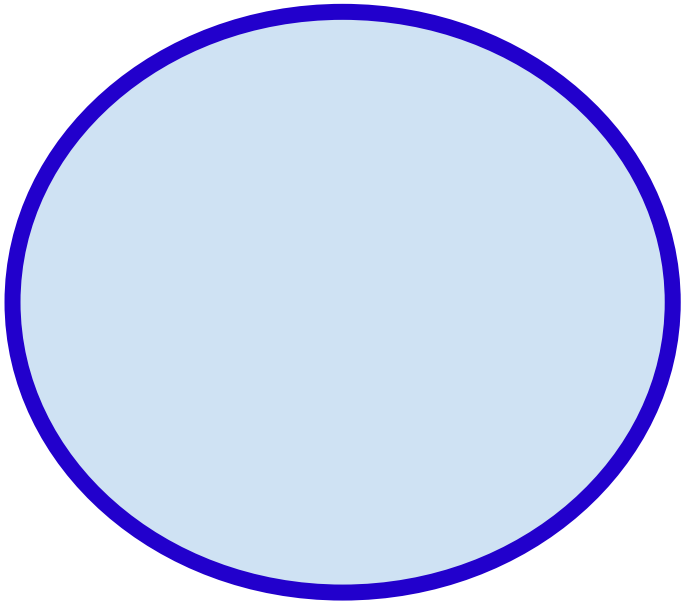
- QUACK design goals
- Identifying available classes
- Restricting classes with static duck typing
- Putting it all together

Setting allowed_classes

Available classes

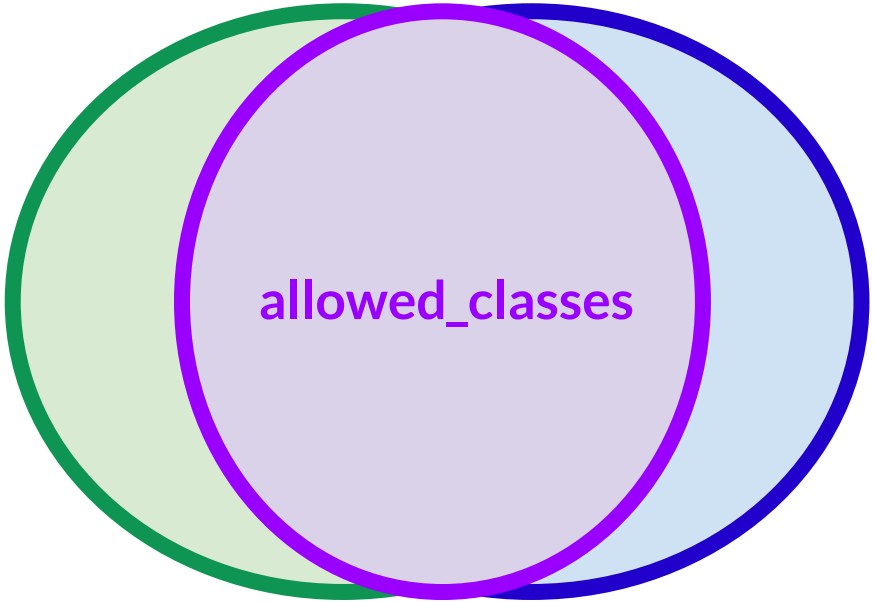


Possible classes



Setting allowed_classes

Available classes



Possible classes

Setting allowed_classes

```
foreach ($data['#']['answer'] as $answerxml) {
    $ans = $format->import_answer($answerxml);
    $options = unserialize($ans->feedback['text']);
    $question->choices[] = array(
        'answer' => $ans->answer,
        'choicegroup' => $options->draggroup,
        'infinite' => $options->infinite,
    );
}
```

Setting allowed_classes

```
foreach ($data['#']['answer'] as $answerxml) {
    $ans = $format->import_answer($answerxml);
    $options = unserialize($ans->feedback['text']);
    $question->choices[] = array(
        'answer' => $ans->answer,
        'choicegroup' => $options->draggroup,
        'infinite' => $options->infinite,
    );
}
```

Setting allowed_classes

```
"conditions": [
    {
        "condType": "Duck",
        "field": "infinite",
        "reason": "HasField",
        "type": "Google_Service_Dataflow_SourceMetadata|
qtype_ddimageortext_drag_item|qtype_ddmarker_drag_item|
qtype_ddwtos_choice",
        "nodeId": "94489654568"
    },
    {
        "condType": "Duck",
        "field": "draggroup",
        "reason": "HasField",
        "type": "qtype_ddwtos_choice",
        "nodeId": "94489654568"
    }
]
```

```
"avail_classes": [
    ...
    "Google_Service_SQLAdmin_User",
    "HTMLPurifier_AttrDef_CSS_ListStyle",
    "Google_Service_AdExchangeSeller_Report",
    "qtype_ddwtos_choice",
    "ADODB2_mssqlnative",
    "Google_Service_YouTube_Watermarks_Resource",
    "moodle1_mod_data_handler",
    ...
]
```

Setting allowed_classes

```
foreach ($data['#']['answer'] as $answerxml) {
    $ans = $format->import_answer($answerxml);
    $options = unserialize($ans->feedback['text'],
        ['allowed_classes' => [qtype_ddwtos_choice::class]]);
    $question->choices[] = array(
        'answer' => $ans->answer,
        'choicegroup' => $options->draggroup,
        'infinite' => $options->infinite,
    );
}
```

Demo

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

Background

PHP deserialization exploits

QUACK

Mitigating deserialization exploits with *static duck typing*

Takeaways

The future of QUACK and deserialization exploits

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

QUACK's Effectiveness

Test setup: Identified 11 applications with CVEs

- 15 vulnerable unserialize calls
- Automated exploit generation tools automatically generate exploits for 5 CVEs

Results:

- QUACK blocked *all methods* for 12/15 vulnerable calls
  - Blocked 97% of methods overall
- No exploits can be generated

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

QUACK's Future

Battle testing: we want to know where QUACK can help!

- We welcome new users – please raise GitHub issues
- Contributing to Joern's PHP support helps QUACK, too

Improved usability

- Imagine: IDE integration with immediate suggestions for `allowed_classes`

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

Is the Deserialization Problem Solved?

QUACK *cannot* prevent “data-only” attacks

Always ask: do I *need* this deserialization call?
What other mitigations apply to my use case?

Other languages?
- Java and C# have similar considerations
- Python’s pickle is more challenging

Is the Deserialization Problem Solved?

QUACK *cannot* prevent "data-only" attacks

Always ask: do I *need* this deserialization call?
What otl

Zhang et al. "Automatic Policy Synthesis and Enforcement for Protecting Untrusted Deserialization" NDSS '24

Other la guages?
- Java and C# have similar considerations
- Python's pickle is more challenging

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

Roadmap

Takeaways

The future of QUACK and deserialization exploits

How well does QUACK work?

How can QUACK be improved?

Is the deserialization security problem solved?

Parting thoughts

BlackHat Sound Bites

Attackers exploit deserialization vulns by chaining together object types that the programmer never intended to instantiate

QUACK prevents exploits by observing how objects are used to infer and restrict the intended types

Developers: only use `unserialize` calls when needed, and use `allowed_classes` when you do. QUACK can help :)

github.com/columbia/quack

Paper: bit.ly/4eInLri