

# QUACK: Hindering Deserialization Attacks via Static Duck Typing

Yaniv David<sup>1</sup>, Neophytos Christou<sup>2</sup>, Andreas D. Kellas<sup>1</sup>,  
Vasileios P. Kemerlis<sup>2</sup>, Junfeng Yang<sup>1</sup>

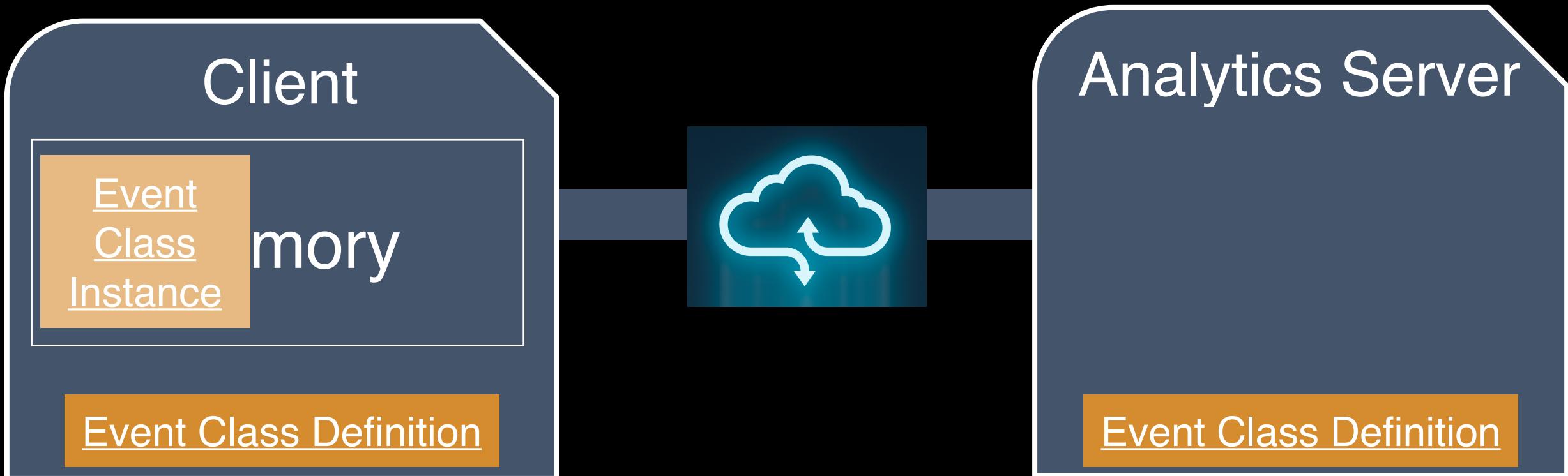
<sup>1</sup>Columbia University   <sup>2</sup>Brown University



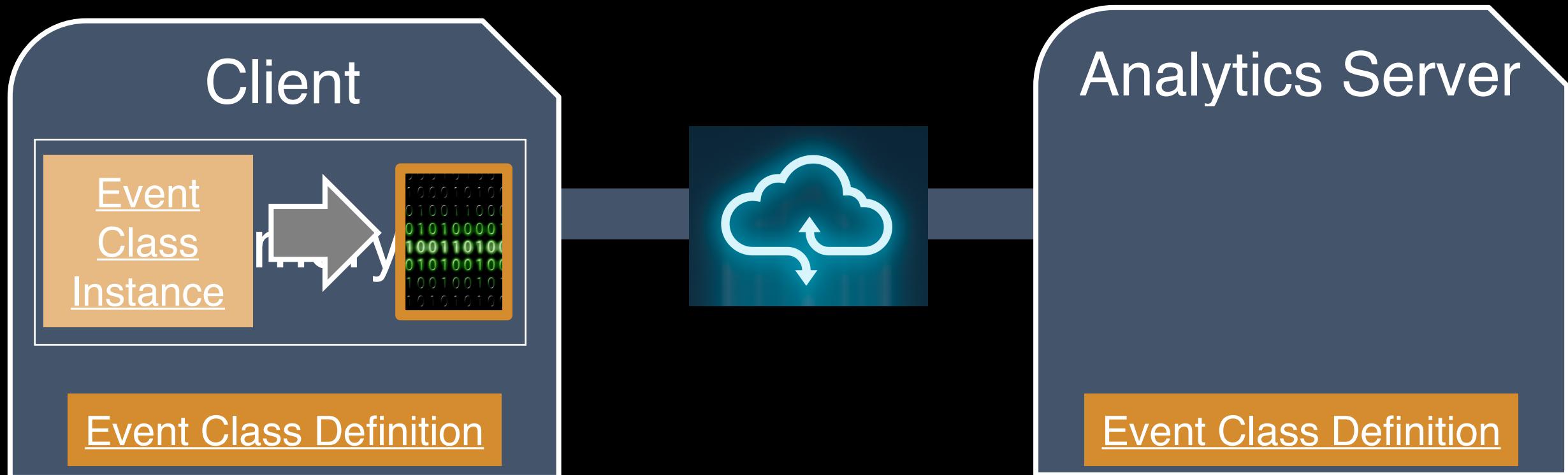
# Background: What's the Deal With Serialization?



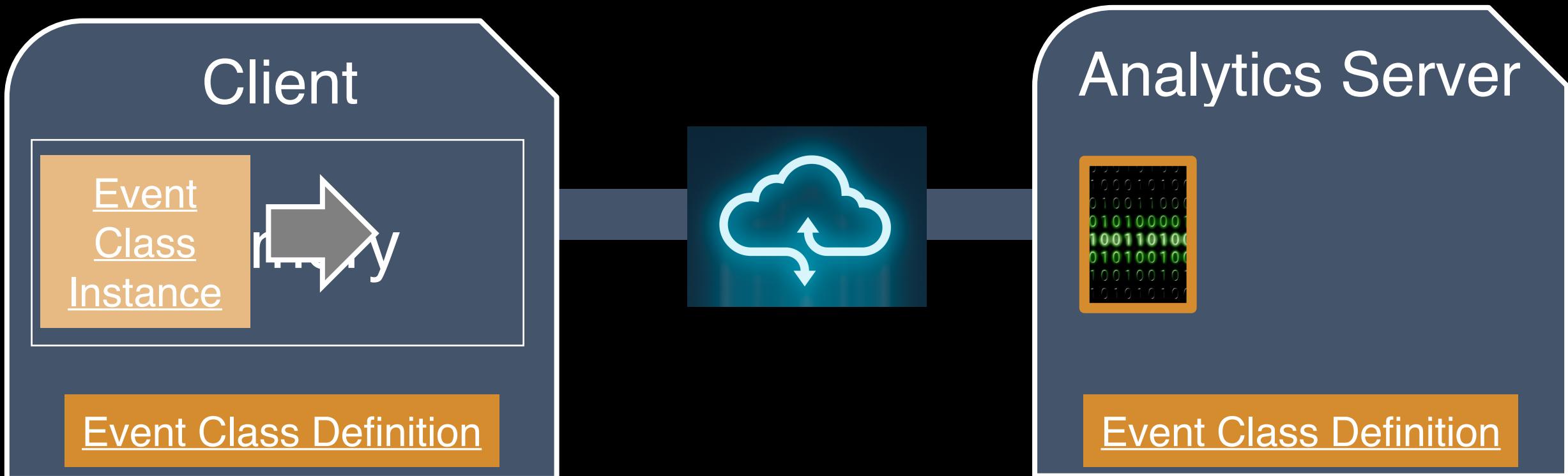
# Background: What's the Deal With Serialization?



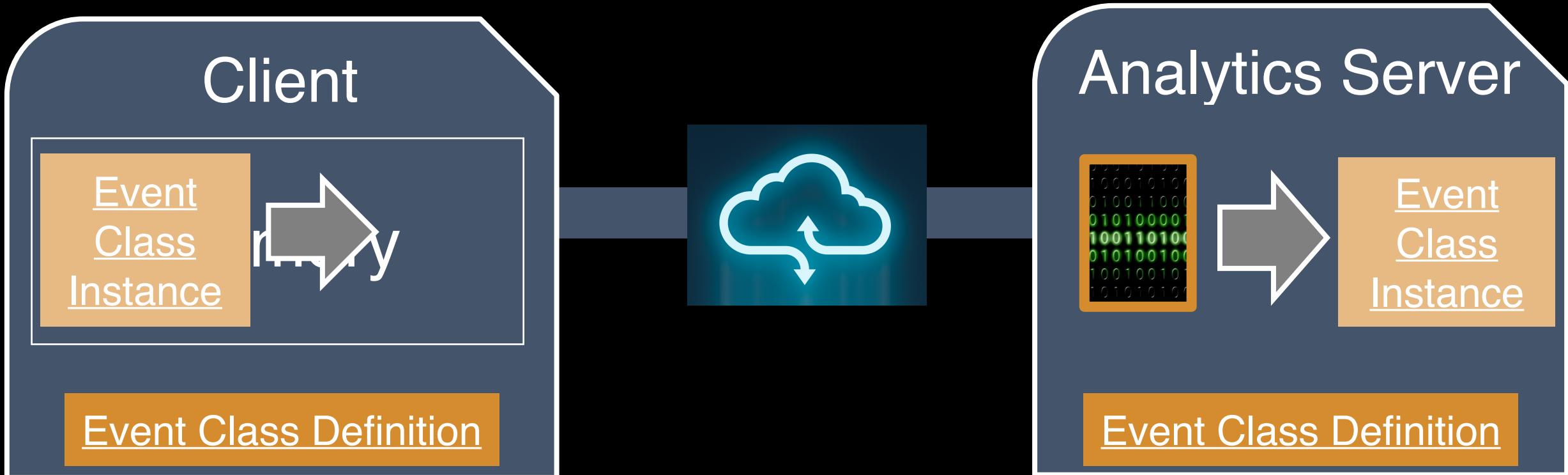
# Background: What's the Deal With Serialization?



# Background: What's the Deal With Serialization?



# Background: What's the Deal With Serialization?



# Ease-Of-Use Trumps Safety

Client

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send =  
    serialize(event);
```



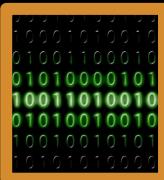
Analytics Server

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
recv_event =  
    deserialize(ser_event);
```

# Ease-Of-Use Trumps Safety

Client

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send =  
    serialize(event);
```



Analytics Server

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
recv_event =  
    deserialize(ser_event);
```

# Ease-Of-Use Trumps Safety

Client

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send =  
    serialize(event);
```

O:13:"MessageLogger":1:{  
2 s:22:"\x00MessageLogger\x00logFile";  
3 s:9:".htaccess";}

Analytics Server

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
recv_event =  
    deserialize(ser_event);
```



# Ease-Of-Use Trumps Safety

Client

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send =  
    serialize(event);
```

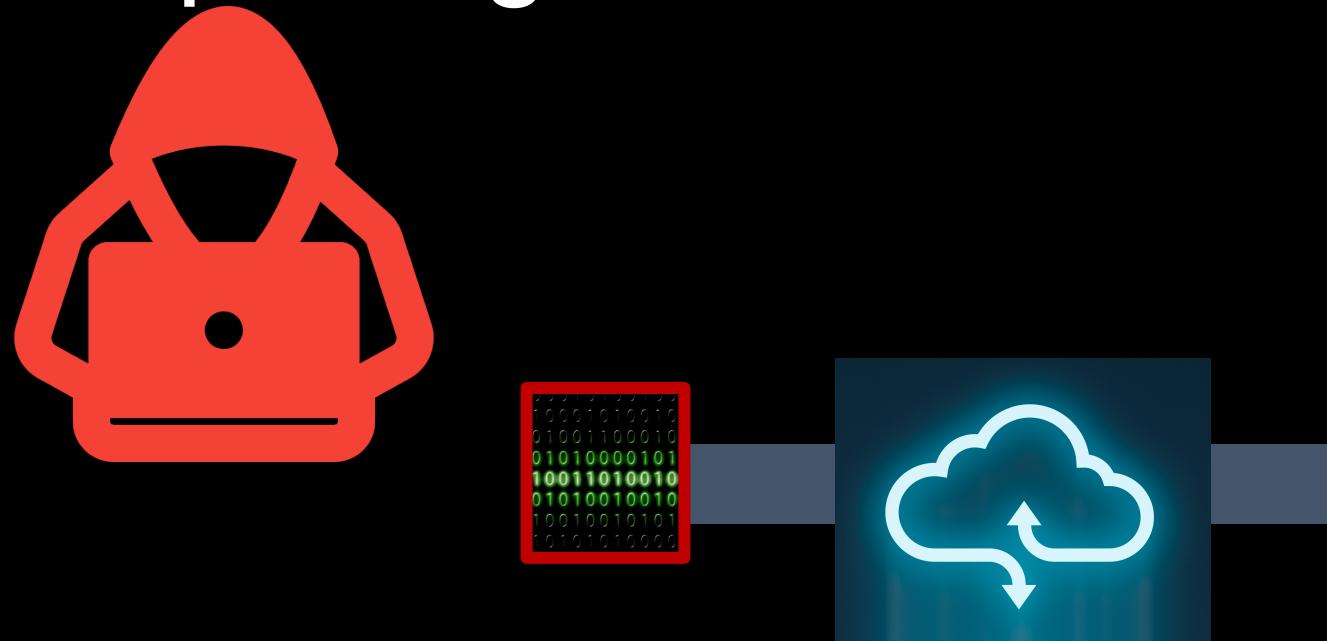
Event()  
->wrapped\_obj=...

Analytics Server

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
recv_event =  
    deserialize(ser_event);
```



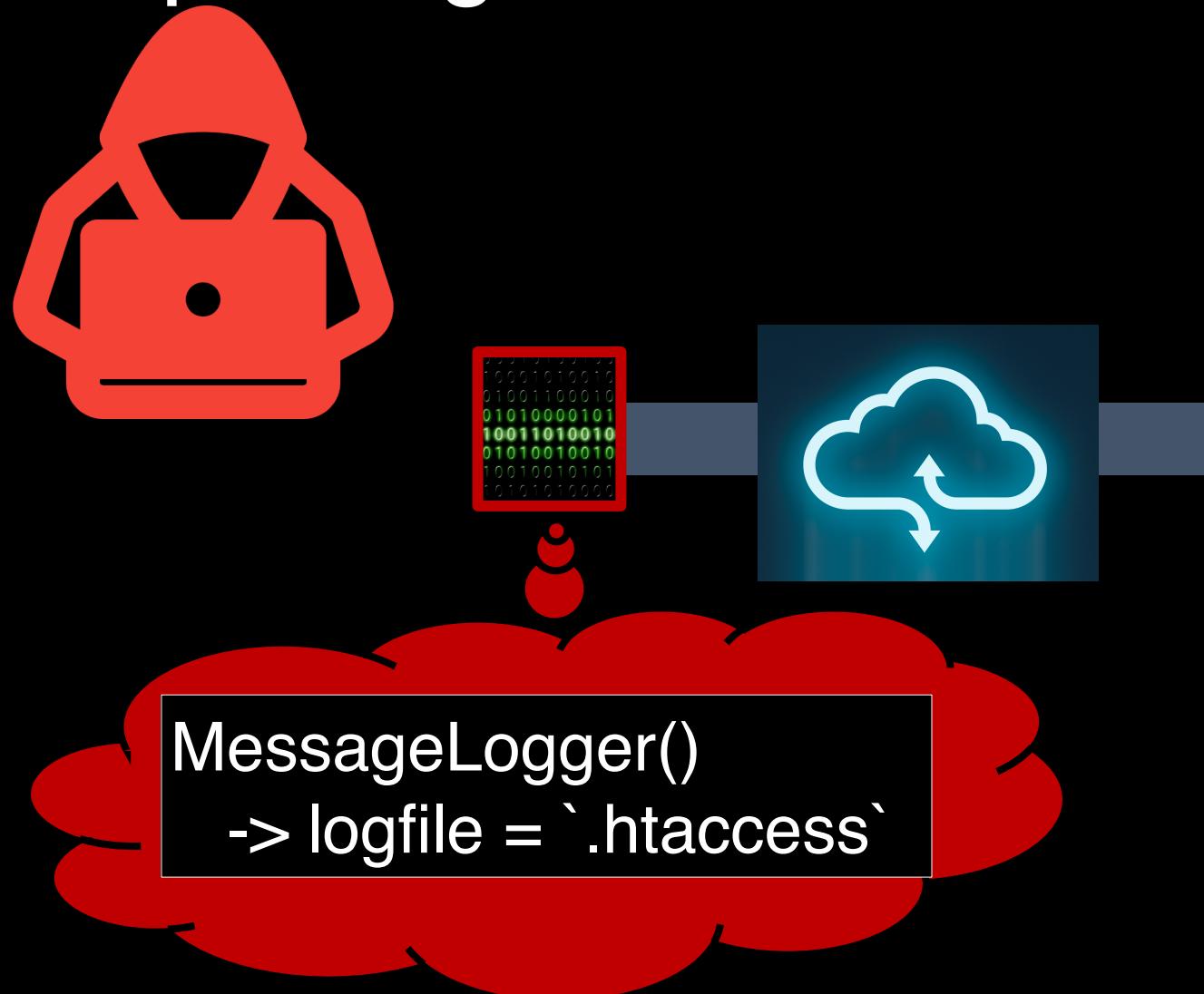
# Exploiting A Deserialization Vulnerability



Analytics Server

```
recv_event =  
    deserialize(ser_event);
```

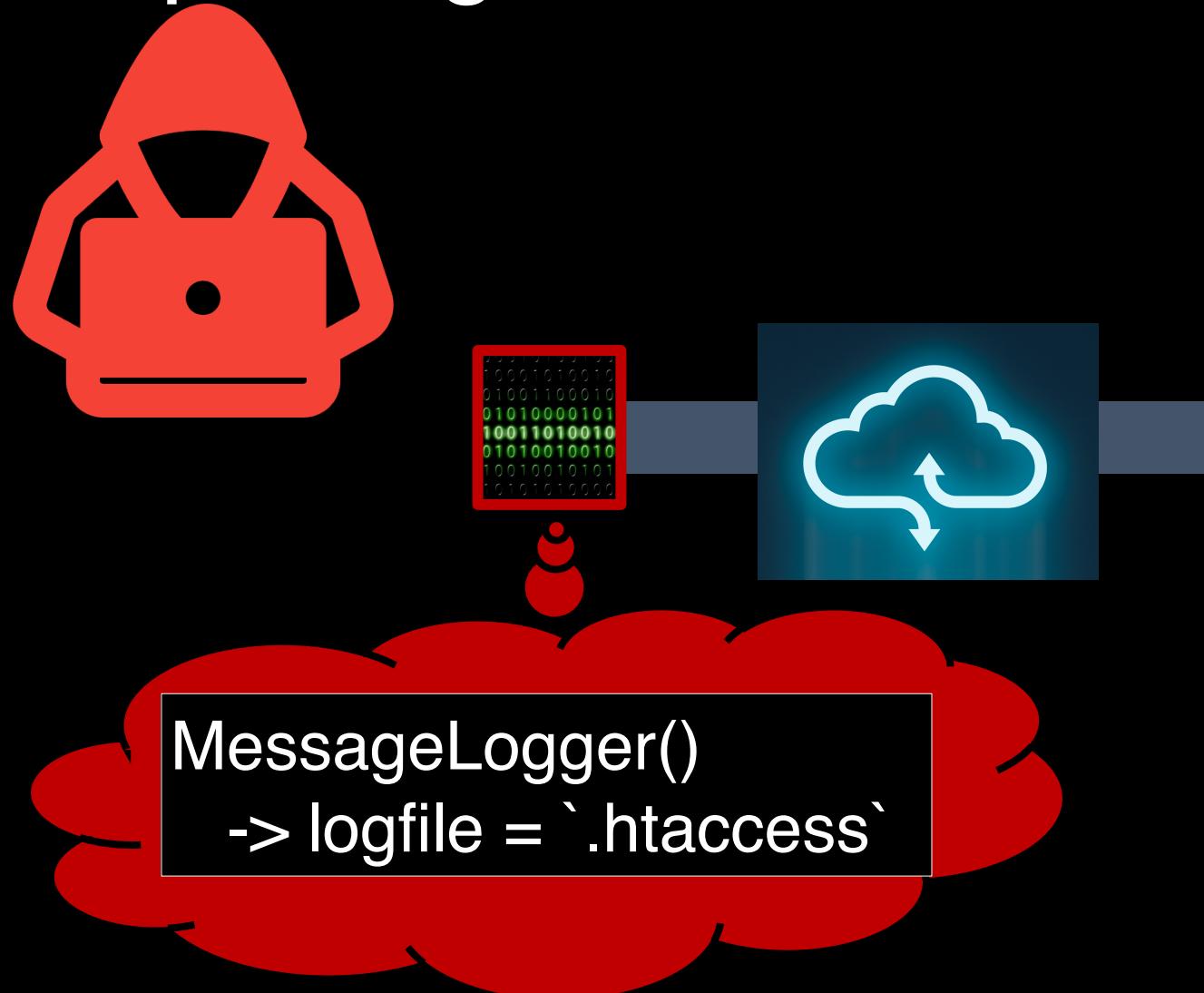
# Exploiting A Deserialization Vulnerability



Analytics Server

```
recv_event =  
deserialize(ser_event);
```

# Exploiting A Deserialization Vulnerability



Analytics Server

```
recv_event =  
    deserialize(ser_event);
```

LogginClass.MessageLogger

# Exploiting A Deserialization Vulnerability

A diagram illustrating the exploit flow. It starts with a red cloud containing the code for the `MessageLogger` class. An arrow points down to a red cloud containing the constructor for `MessageLogger` with the parameter `-> logfile = `/.htaccess``. Another arrow points right to a blue rounded rectangle labeled "Analytics Server". Inside the "Analytics Server" box is the code `recv_event = deserialize(ser_event);`. A blue arrow points from the `logfile` value in the first cloud to the `deserialize` call in the second cloud.

```
class MessageLogger {  
    public function __wakeup() {  
        unlink(this->logFile); }  
}
```

```
MessageLogger()  
-> logfile = `/.htaccess`
```

Analytics Server

```
recv_event =  
    deserialize(ser_event);
```

LogginClass.MessageLogger

# Exploiting A Deserialization Vulnerability

```
class MessageLogger {  
    public function __wakeup() {\n        unlink(this->logFile); }  
}
```

Invoked upon deserialization →  
Subverted to an Exploit-Building Class

```
recv_event =  
    deserialize(ser_event);
```

```
MessageLogger()  
-> logfile = `/.htaccess`
```

LogginClass.MessageLogger

# Deserialization Attacks Affect Real-World Applications

# Deserialization Attacks Affect Real-World Applications



## OWASP Top Ten

- A1 – Injections
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities (XXE)
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Components with Known Vulnerabilities
- A10 – Insufficient Logging & Monitoring

# Deserialization Attacks Affect Real-World Applications



## OWASP Top Ten

- A1 – Injections
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities (XXE)
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Components with Known Vulnerabilities
- A10 – Insufficient Logging & Monitoring

# Deserialization Attacks Affect Real-World Applications



OWASP®

## OWASP Top Ten

- A1 – Injections
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities (XXE)
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Components with Known Vulnerabilities
- A10 – Insufficient Logging & Monitoring

# GitHub Advisory Database

Security vulnerability database inclusive of CVEs and GitHub originated security advisories.

GitHub reviewed advisories

All reviewed 15,705

Composer 2,446

Q cwe:502

1,025 advisories

# Deserialization Attacks Affect Real-World Applications

hackerone

- Hacktivity
- Opportunities
- Directory
- Leaderboard

CWE-502

Deserialization of Untrusted Data

Reports Severity Remediation

Remediation Distribution (all time)

Submissions

Time Period	Submissions
< 1 day	24
1-2 days	18
2-3 days	13
3-7 days	60
7-30 days	148
30-90 days	132
90-365 days	119
365+ days	43
Pending	94

# Deserialization Attacks Affect Real-World Applications

The screenshot shows a blog post on the hackerone.com website. The header includes the hackerone logo, the text "NOW PART OF Google Cloud", and navigation links for Platform, Solutions, Intelligence, Services, Resources, and Company. A search bar and user profile icons are also present. The main content area features a large title: "Now You Serial, Now You Don't – Systematically Hunting for Deserialization Exploits". Below the title, the author is listed as "ALYSSA RAHMAN" and the date as "DEC 13, 2021 | 17 MIN READ". To the right of the article, there is a sidebar with a "Submissions" chart showing a list of numbers: 24, 18, 13, 60, 148, 132, 119, 43, and 94. The number 94 is highlighted with a red border. At the bottom of the page, there is a footer with a "Submissions" section and a page number "5".

CWE-502

MANDIANT NOW PART OF Google Cloud

Platform Solutions Intelligence Services Resources Company

Blog Support

BLOG

## Now You Serial, Now You Don't – Systematically Hunting for Deserialization Exploits

ALYSSA RAHMAN

DEC 13, 2021 | 17 MIN READ

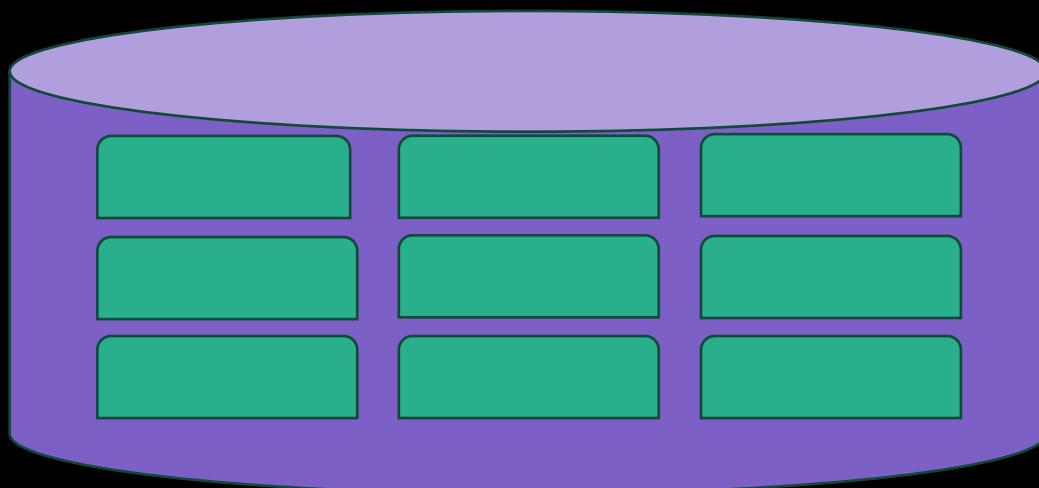
Submissions

24  
18  
13  
60  
148  
132  
119  
43  
94

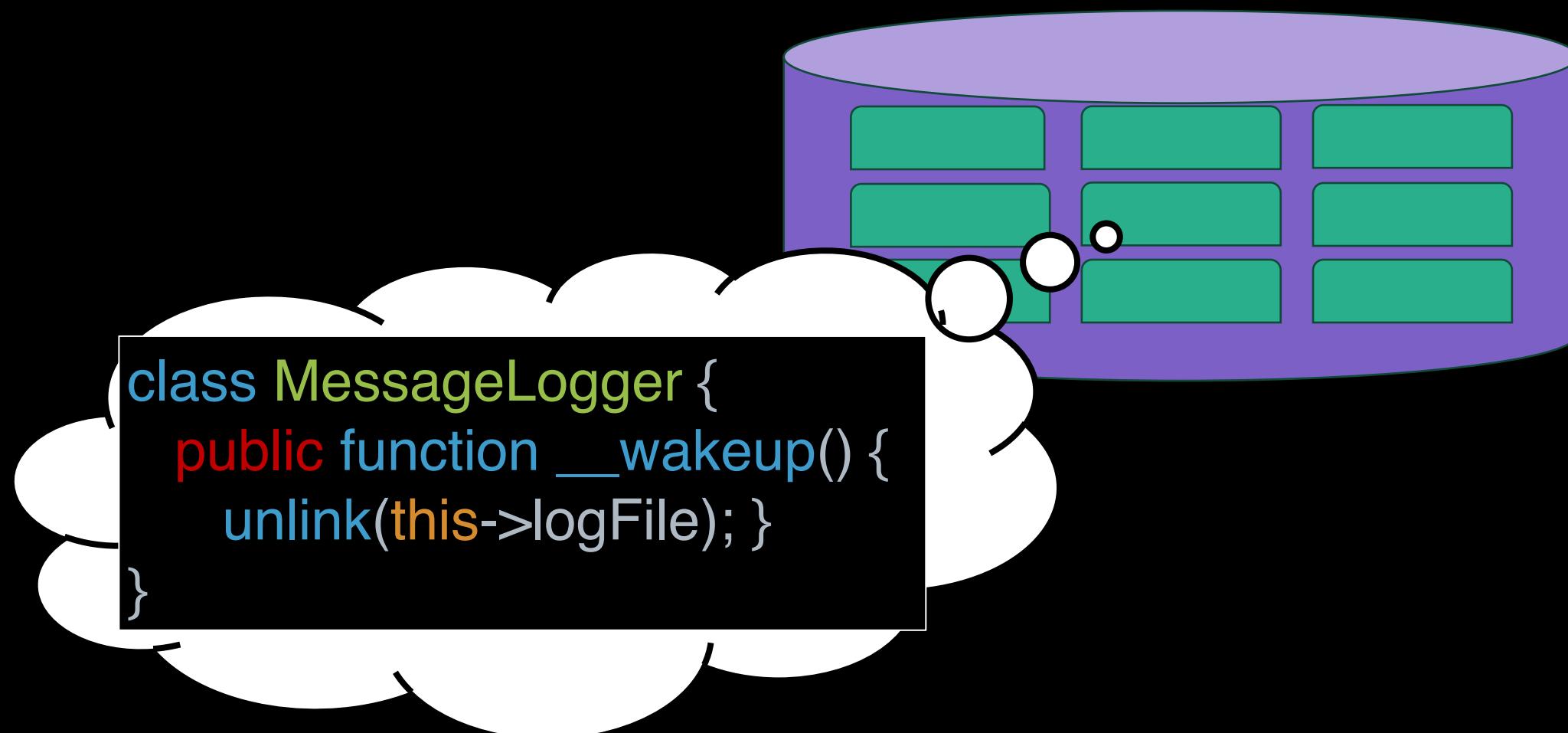
5

# Exploit Generation Techniques Are Improving

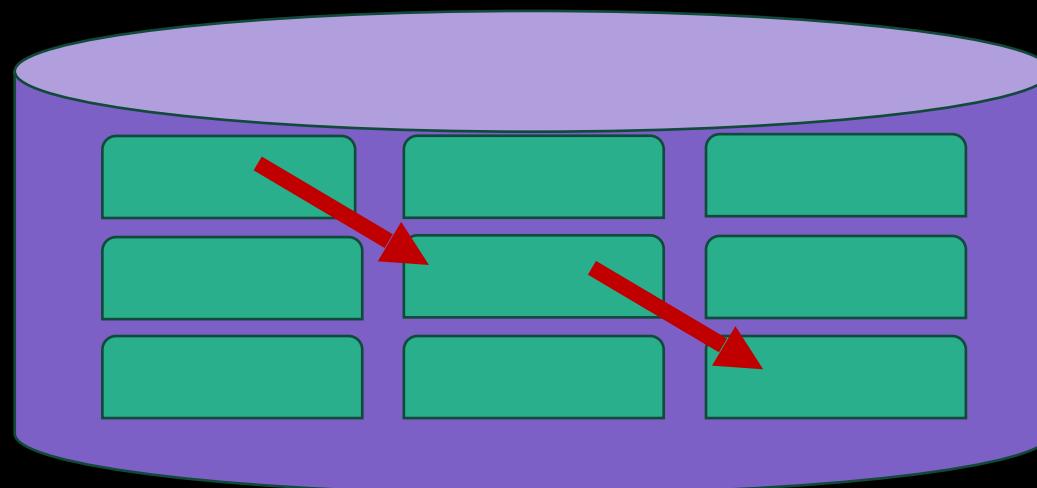
# Exploit Generation Techniques Are Improving



# Exploit Generation Techniques Are Improving



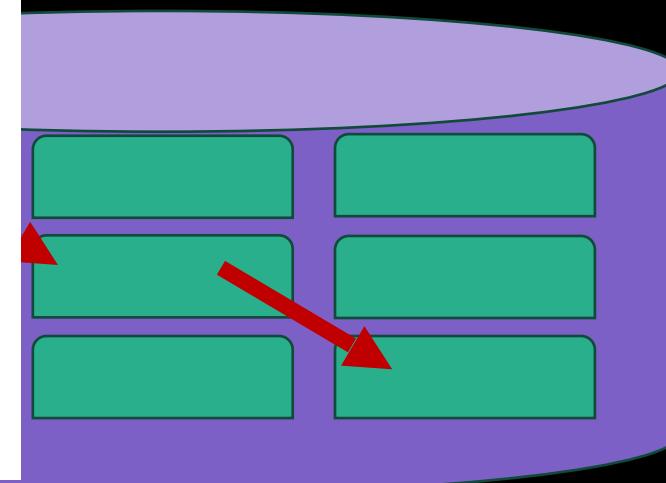
# Exploit Generation Techniques Are Improving



# Exploit Generation Techniques Are Improving

## Code Reuse Attacks in PHP: Automated POP Chain Generation

Johannes Dahse, Nikolai Krein, and Thorsten Holz  
Horst Görtz Institute for IT-Security (HGI)  
Ruhr-University Bochum, Germany  
[{firstname.lastname}@rub.de](mailto:{firstname.lastname}@rub.de)



# Exploit Generation Techniques Are Improving

## Code Reuse Attacks in PHP: Automated POP Chain Generation

Johanne

### FUGIO: Automatic Exploit Generation for PHP Object Injection Vulnerabilities

Sunyeo Park\*  
*KAIST*

Daejun Kim\*  
*KAIST*

Suman Jana  
*Columbia University*

Sooel Son  
*KAIST*

# Exploit Generation Techniques Are Improving

## Code Reuse Attacks in PHP: Automated POP Chain Generation

Johanne

FUGIO: Automatic Exploit Generation for  
PHP Object Injection Vulnerabilities

Sunny

KA  
Improving Java Deserialization Gadget Chain  
Mining via Overriding-Guided Object Generation

Sicong Cao<sup>†\*</sup>, Xiaobing Sun<sup>†✉</sup>, Xiaoxue Wu<sup>†✉</sup>, Lili Bo<sup>†✉</sup>, Bin Li<sup>†</sup>, Rongxin Wu<sup>‡</sup>,  
Wei Liu<sup>†</sup>, Biao He<sup>§</sup>, Yu Ouyang<sup>§</sup>, Jiajia Li<sup>§</sup>

# Exploit Generation Techniques Are Improving

## Code Reuse Attacks in PHP: Automated POP Chain Generation

Johanne

FUGIO: Automatic Exploit Generation for  
PHP Object Injection Vulnerabilities

Sunny

KA  
Improving Java Deserialization Gadget Chain  
Mining via Overriding-Guided Object Generation

ODDFUZZ: Discovering Java Deserialization Vulnerabilities  
via Structure-Aware Directed Greybox Fuzzing

Sicong Cao<sup>†\*</sup>, Biao He<sup>‡</sup>, Xiaobing Sun<sup>†✉</sup>, Yu Ouyang<sup>‡</sup>, Chao Zhang<sup>§</sup>, Xiaoxue Wu<sup>†</sup>, Ting Su<sup>¶</sup>,  
Lili Bo<sup>†</sup>, Bin Li<sup>†</sup>, Chuanlei Ma<sup>‡</sup>, Jiajia Li<sup>‡</sup>, Tao Wei<sup>‡</sup>

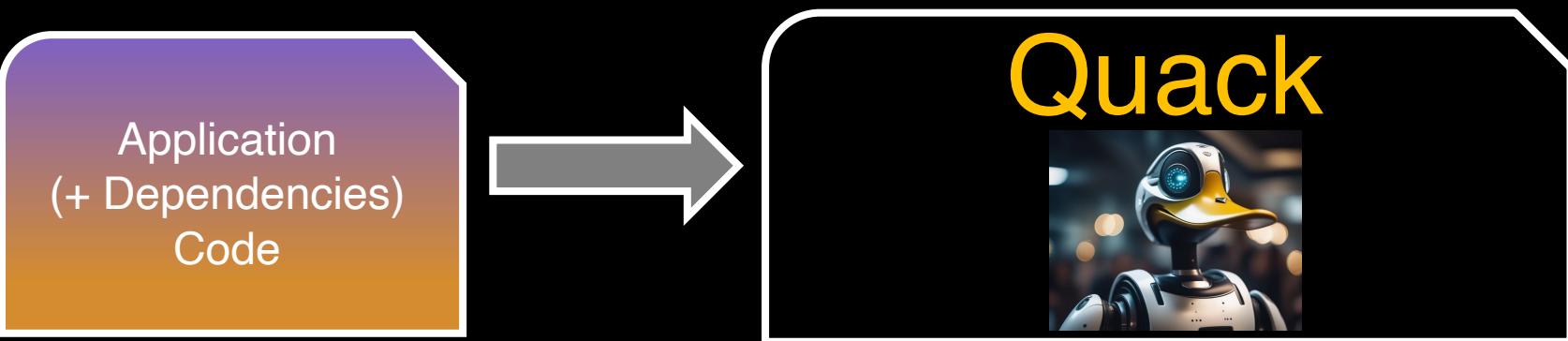
# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]

Application  
(+ Dependencies)  
Code

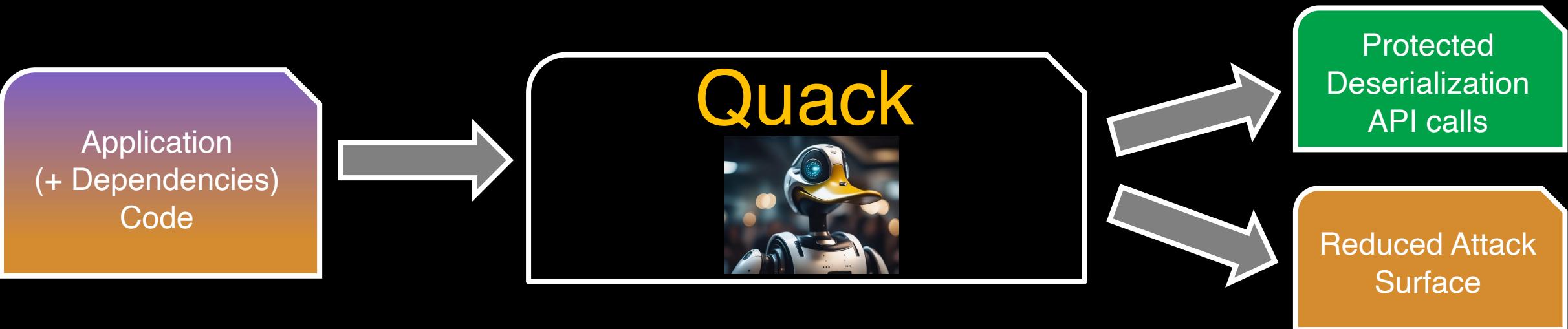
# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



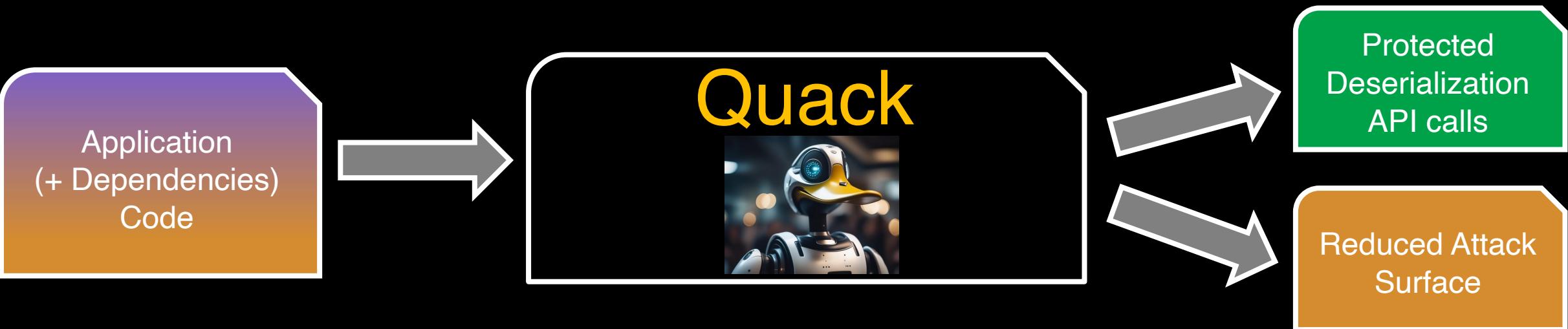
# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



# Quack: Hindering Deserialization Attacks

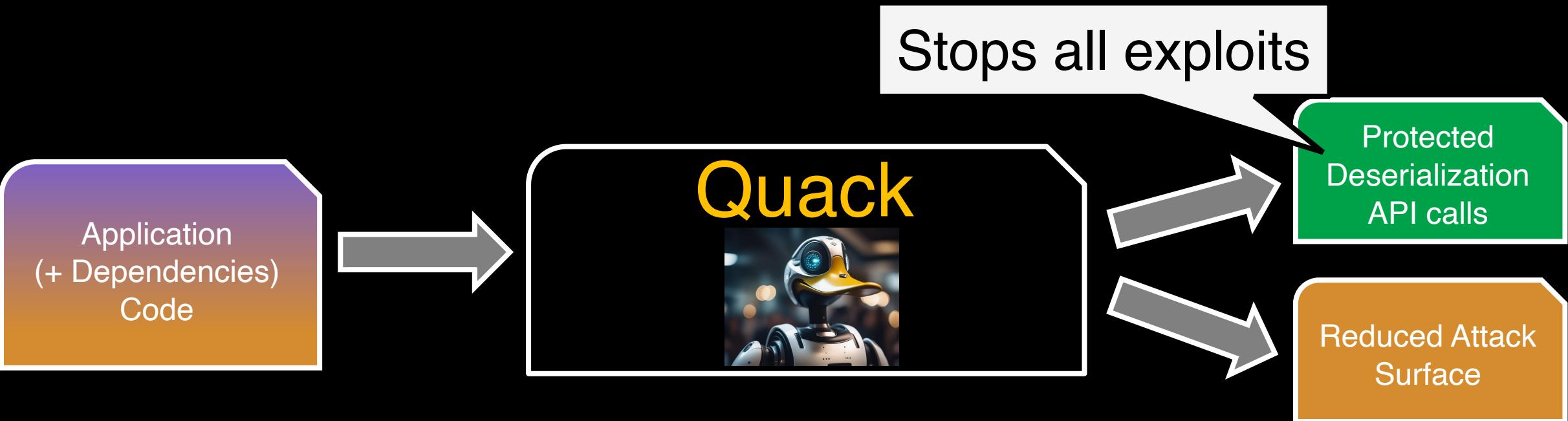
David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



Built Quack-php and evaluated on diverse set of real applications  
against SOTA exploit-generation-tool

# Quack: Hindering Deserialization Attacks

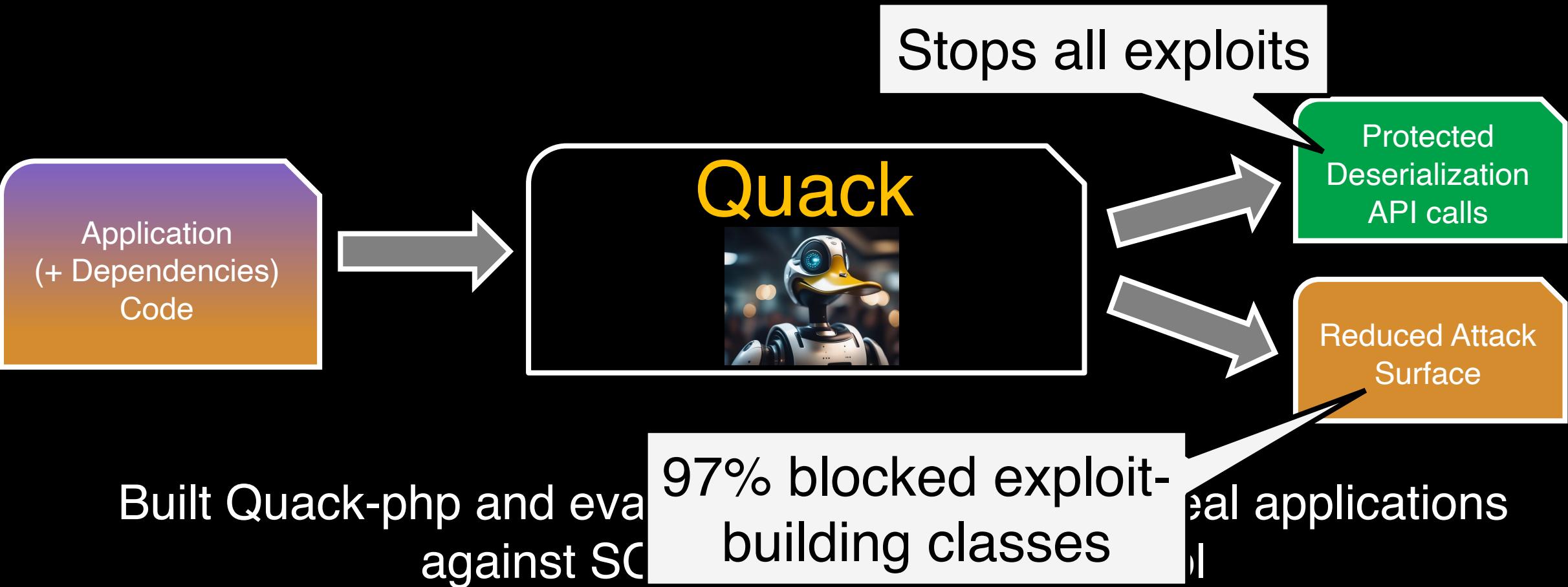
David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



Built Quack-php and evaluated on diverse set of real applications  
against SOTA exploit-generation-tool

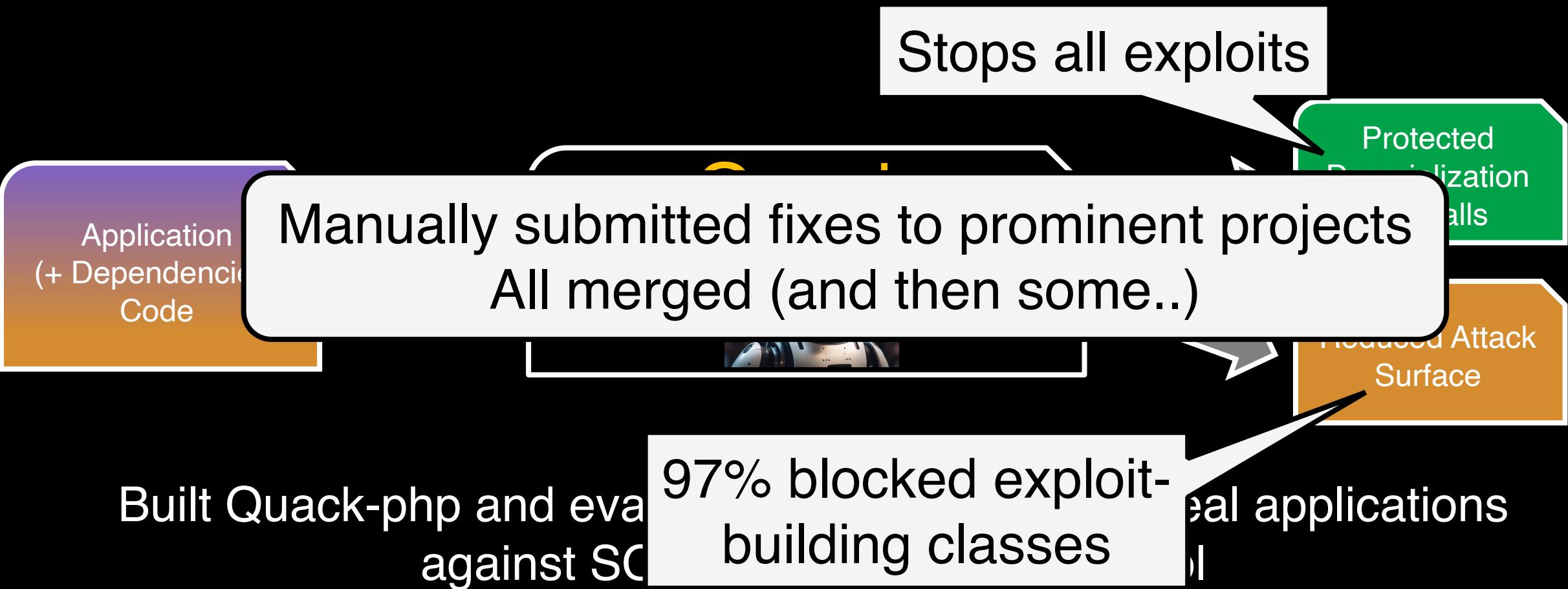
# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]



# Quack: David et. al



Application  
(+ Dependencies)  
Code

M

Built Qua



markstory commented on Jun 20, 2023

Thank you 🎉

exploits

Protected  
Initialization  
Calls

projects

Reduced Attack  
Surface

real applications

# Quack: Hindering Deserialization Attacks

David et. al., Hindering Deserialization Attacks via Static Duck Typing [NDSS '24]

Stops all exploits

The screenshot shows a GitHub commit page for a pull request. The commit message is "Set allowed\_classes to false in unserialize call #338". It includes a link to the commit author's profile, "slackero committed on May 30, 2023 Verified". Below the commit message, it says "Showing 88 changed files with 127 additions and 114 deletions." To the right of the commit message, there are three colored callout boxes: a green one for "Selected serialization calls", an orange one for "Attack surface", and a white one containing the text "Stops all exploits".

Set allowed\_classes to false in unserialize call #338

v1.10-dev

slackero committed on May 30, 2023 Verified

Showing 88 changed files with 127 additions and 114 deletions.

Selected serialization calls

Attack surface

Stops all exploits

Built Quack-php and eval against SC

97% blocked exploit-building classes

real applications

# Why Not Limit Deserialized Classes?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

# Why Not Limit Deserialized Classes?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

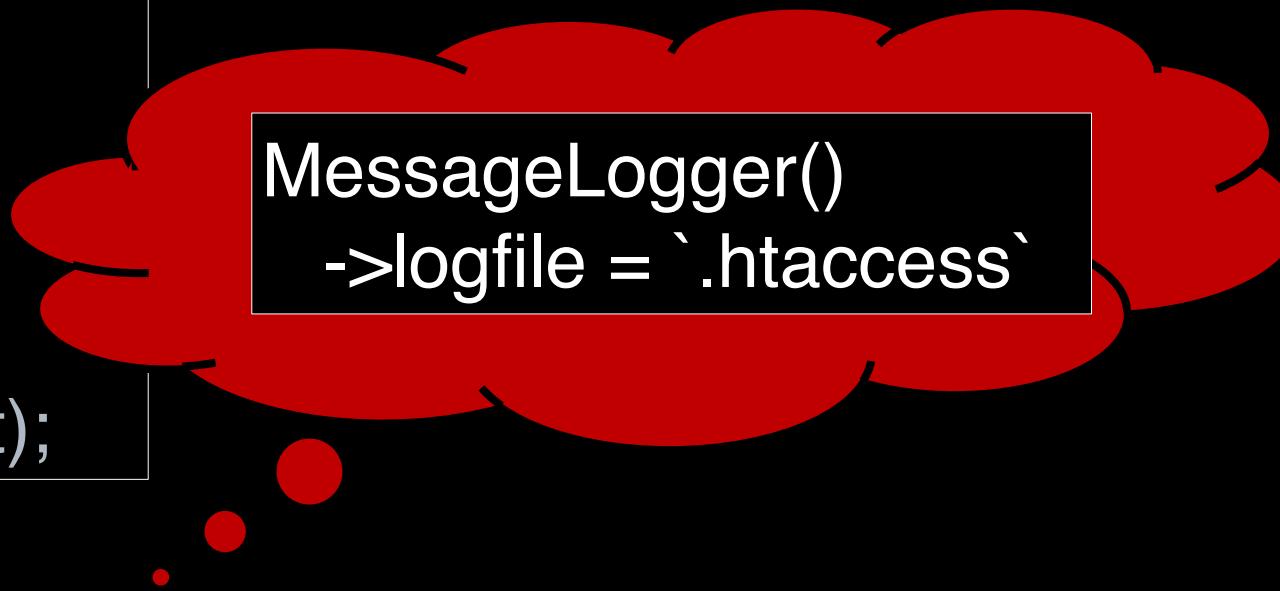
# Why Not Limit Deserialized Classes?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Classes?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

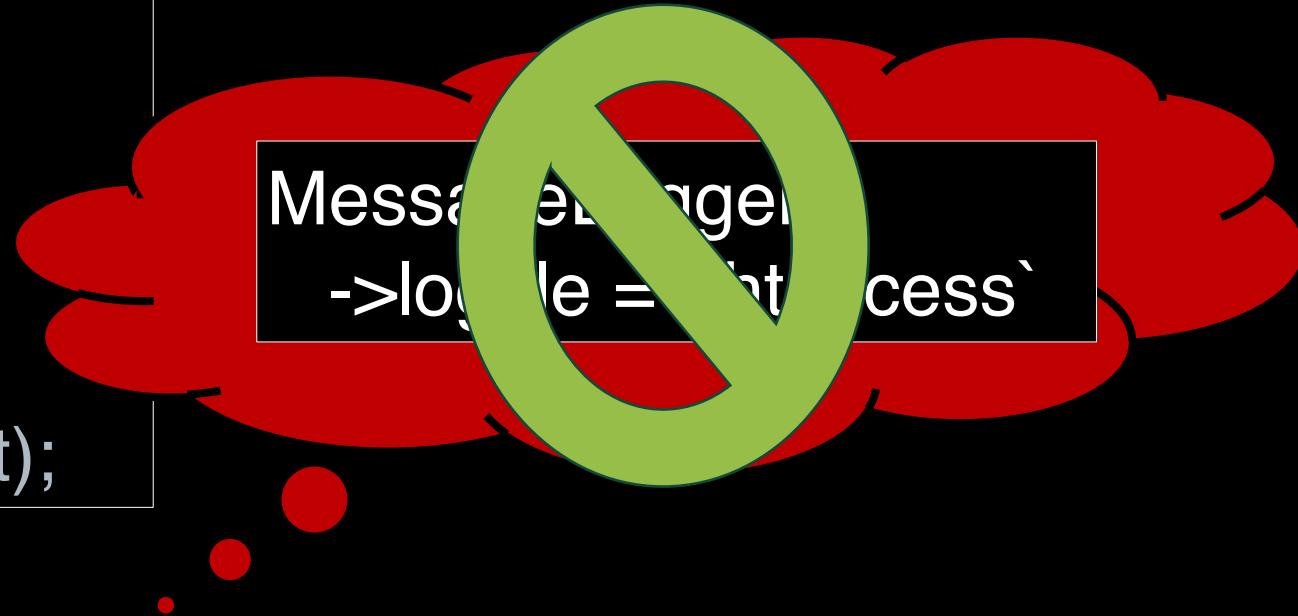


```
MessageLogger()  
->logfile = `htaccess`
```

```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Classes?

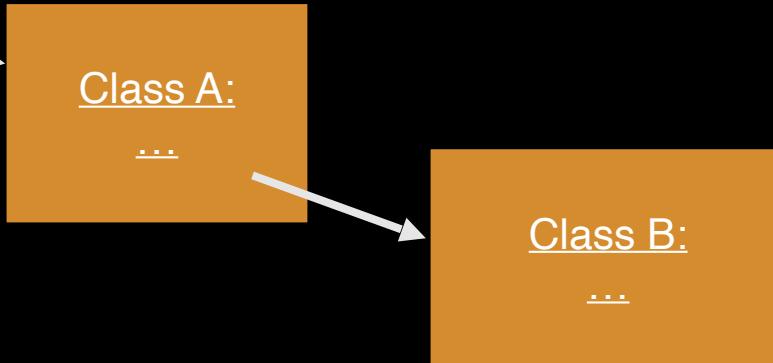
```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```



```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Classes?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

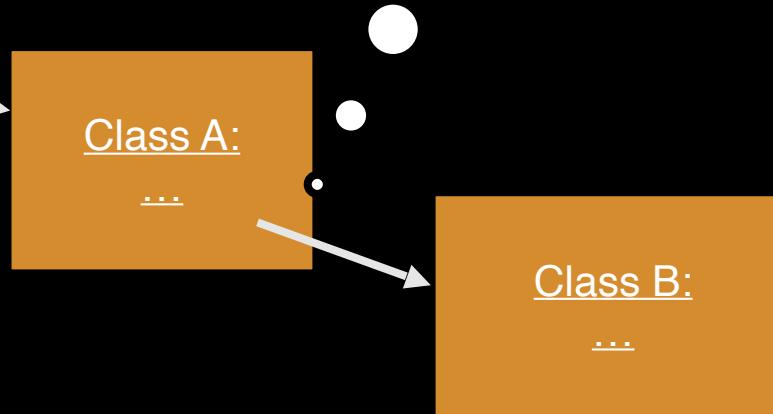


```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Classes?

What happens if  
these are updated?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

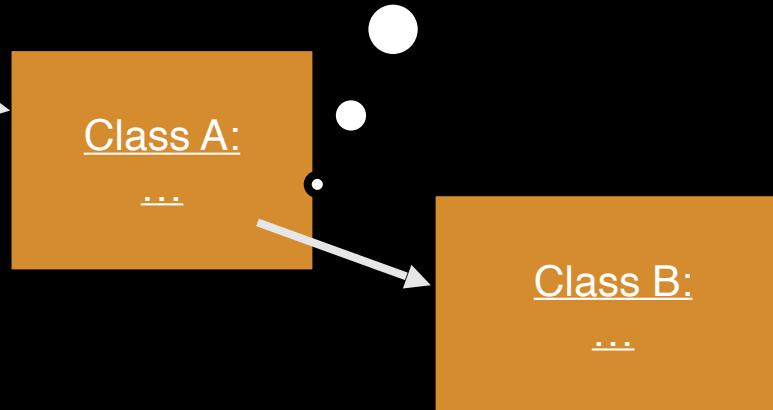


```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Objects?

What happens if  
these are updated?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```

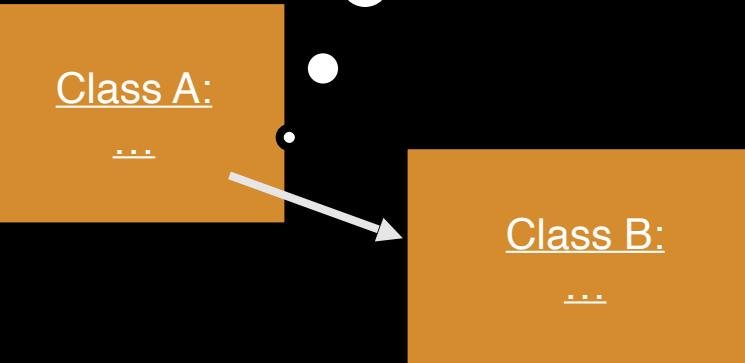


```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```

# Why Not Limit Deserialized Classes?

What happens if  
these are updated?

```
class Event {  
    private wrapped_obj;  
    /* snip */  
}  
/* snip */  
to_send = serialize(event);
```



```
recv_event = deserialize(ser_event, ['allowed_classes'=>['Event']]);
```



# Key Challenge in Protecting Deserialization API calls

# Key Challenge in Protecting Deserialization API calls

- Infer all types for root objects + fields and collections

# Key Challenge in Protecting Deserialization API calls

- Infer all types for root objects + fields and collections

```
recv_event = deserialize(ser_event, ['allowed_classes'=>[???]]);
```

# Key Challenge in Protecting Deserialization API calls

- Infer all types for root objects + fields and collections

```
recv_event = deserialize(ser_event, ['allowed_classes'=>[???]]);
```



Type checkers are  
not geared for this

# Key Challenge in Protecting Deserialization API calls

- Infer all types for root objects + fields and collections

Type inference is too  
conservative

```
recv_event = deserialize(ser_event, ['allowed_classes'=>[???]]);
```

Type checkers are  
not geared for this

# Key Challenge in Protecting Deserialization API calls

- Infer all types for root objects + fields and collections

Type inference is too  
conservative

```
recv_event = deserialize(ser_event, ['allowed_classes' -> ??])
```

Type checkers are  
not geared for this

$e \in [[\text{Class}, \text{int}, \text{str}], \dots]$   
 $| \perp$

# Insight: Infer Classes Using Object's Usage

```
recv_event = deserialize(ser_event, ['allowed_classes'=>??]);
```

# Insight: Infer Classes Using Object's Usage

```
recv_event = deserialize(ser_event, ['allowed_classes'=>??]);
```

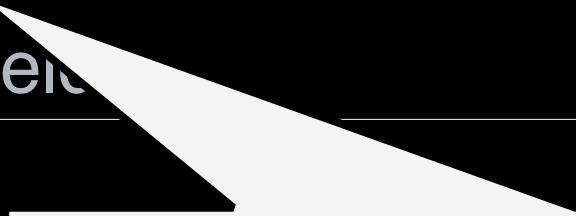
Type class components ≠ full type  
(for security, not optimization)

# Insight: Infer Classes Using Object's Usage

```
recv_event = deserialize(ser_event, ['allowed_classes'=>??]);  
recv_event->my_call();  
something = recv_event->my_field
```

# Insight: Infer Classes Using Object's Usage

```
recv_event = deserialize(ser_event, ['allowed_classes'=>??]);  
recv_event->my_call();  
something = recv_event->my_field();
```



Deserialized objects must contain  
“my\_call” and “my\_field”

# Insight: Infer Classes Using Object's Usage

Decision Point

```
recv_event = deserialize(ser_event, ['allowed_classes'=>??]);  
recv_event->my_call();  
something = recv_event->my_field();
```

Deserialized objects must contain  
“my\_call” and “my\_field”

# Key Idea: Targeted On-Demand Class Identity Inference

- Start with the deserializable classes set
  - Conservatively handle dynamic class-loading patterns
- Gather objects (and fields, recursively) usage evidence
  - Over-approximate or know when to stop
- Use collected evidence to filter class set

# Key Idea: Targeted On-Demand Class Identity Inference

Start with the deserializable classes set

- Conservatively handle dynamic class-loading patterns
- Gather objects (and fields, recursively) usage evidence
  - Over-approximate or know when to stop
- Use collected evidence to filter class set

# Key Idea: Targeted On-Demand Class Identity Inference

Start with the deserializable classes set

- Conservatively handle dynamic class-loading patterns
- Gather objects (and fields, recursively) usage evidence
  - Over-approximate or know when to stop
- Use collected evidence to filter class set

Inferring structure for input objects is useful for many tasks

# Analyzing Our Motivating Example

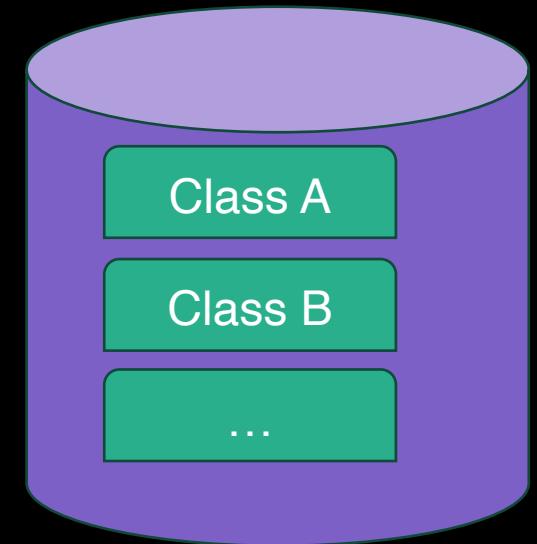
queue.php

```
/* snip */  
raw_event = owa->getLast('event');  
event = deserialize(raw_event);  
  
owa::getEventDispatcher()->notify(event);
```

Open Web Analytics v1.5.6 containing the deserialization  
vulnerability CVE-2014-2294

# Analyzing Our Motivating Example

```
queue.php  
/* snip */  
raw_event = owa->getLast('event');  
event = deserialize(r  
owa::getEventDispatcher()->notify(event);
```



Open Web Analytics v1.5.6 containing the deserialization vulnerability CVE-2014-2294

# Analyzing Our Motivating Example

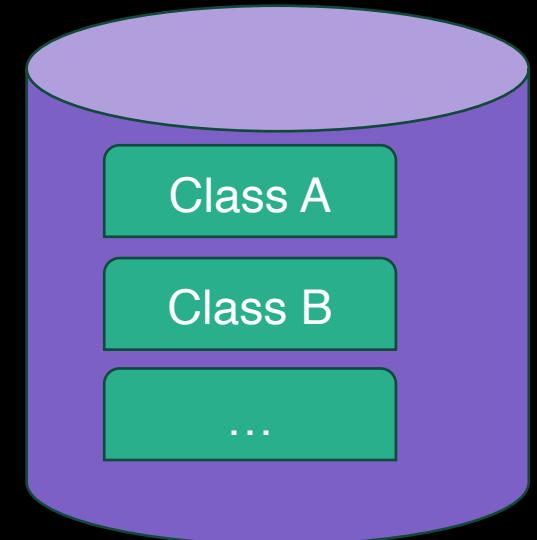
Root Object

/\* snip \*/

```
raw_event = owa->getLast('event');  
event = deserialize(r
```

```
owa::getEventDispatcher()->notify(event);
```

queue.php



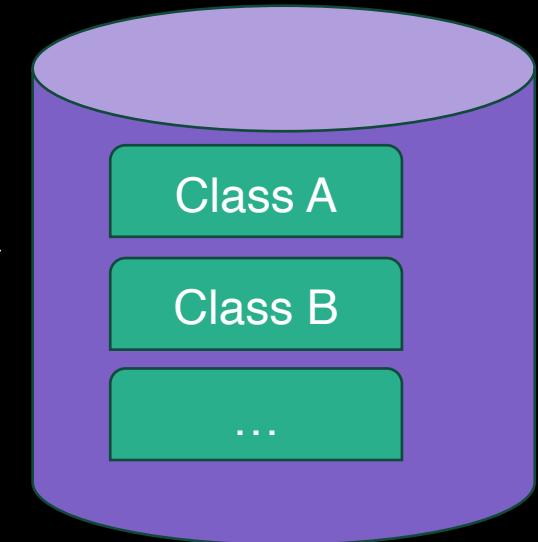
Open Web Analytics v1.5.6 containing the deserialization vulnerability CVE-2014-2294

# Analyzing Our Motivating Example

Root Object

queue.php

```
/* snippet */  
raw_event = owa->getLast('event');  
event = deserialize(  
owa::getEventDispatcher()->notify(event);
```

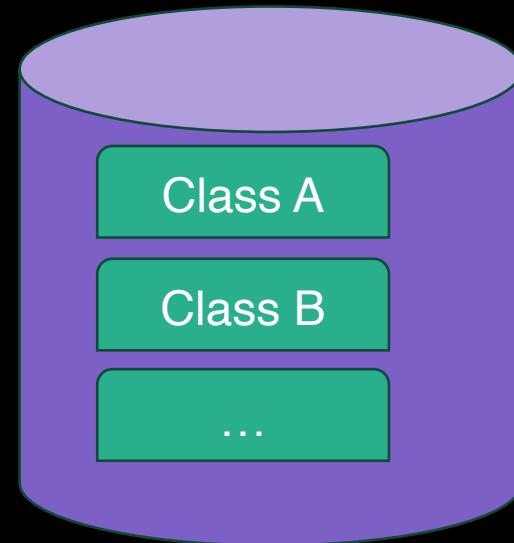


Open Web Analytics  
Follow the link containing the deserialization  
vulnerability CVE-2014-2294

# Analyzing Our Motivating Example

## eventDispatch.php

```
class owa_eventDispatcher {  
    function notify(event) {  
        owa_coreAPI::debug("Notifying listeners of"  
            + event->getEventType());  
    }  
}
```

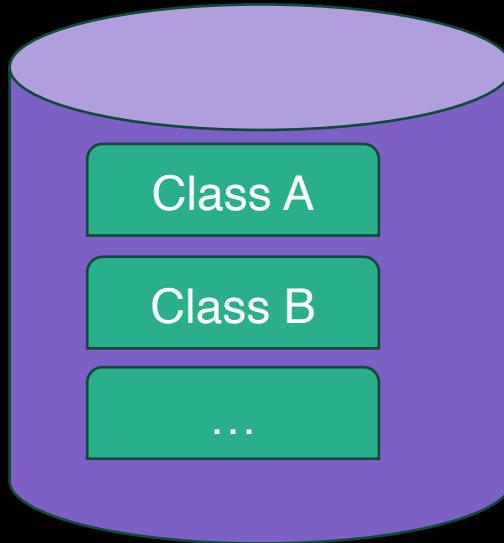


# Analyzing Our Motivating Example

## eventDispatch.php

```
class owa_eventDispatcher {  
    function notify(event) {  
        owa_coreKPI::debug("Notifying listeners of"  
            + event->getEventType());  
    }  
}
```

No Type Hint



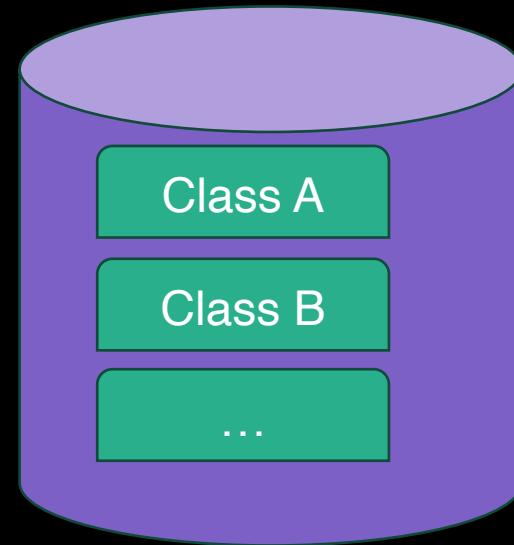
# Analyzing Our Motivating Example

## eventDispatch.php

```
class owa_eventDispatcher {  
    function notify(event) {  
        owa_coreKPI::debug("Notifying listeners of"  
            + event->getEventType());
```

No Type Hint

Has method with this name



# Analyzing Our Motivating Example

## eventDispatch.php

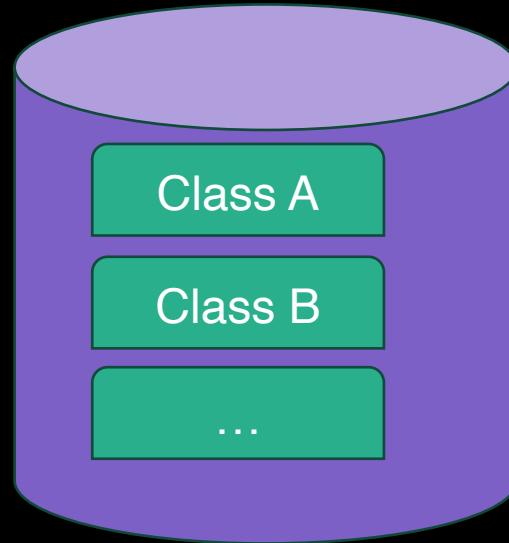
```
class owa_eventDispatcher {  
    function notify(event) {  
        owa_core::API::debug("Notifying listeners of"  
            + event->getEventType());  
    }  
}
```

No Type Hint

+ event->getEventType());

Has method with this name

Returns *string* or *no-return-type*



# Analyzing Our Motivating Example

Rule Type	Partial Statement Matching Rule	Possible Classes
Exact	FunctionX (arg1, argi-1, t, argi+1, ...)	TypeOf(FunctionX's arg;)
	ClassXInstance MethodX (arg1, argi-1, t, argi+1, ...)	TypeOf(ClassX → MethodX's argi)
	(TypeName) t	TypeName
	Expr ? t: a (or symmetric case)	TypeOf(a)
Duck Typing	t->MethodX (...)	Classes with a method named 'MethodX'
	t.Fieldx	Classes with a field named 'FieldX'
	t <BinaryOp>a	Types allowing < BinaryOp>(e.g., "+" or ">=") with TypeOf(a)
	tOp> (or symmetric case)	Types allowing Op (e.g., "++")
	t[offset or key]	Types compatible with slicing
	a <AssignOp> t	Types allowing <AssignOp>(e.g., +=) with TypeOf(a)
	switch (t): case (a)	Types allowing equality check against TypeOf(a)

# Analyzing Our Motivating Example

## eventDispatch.php

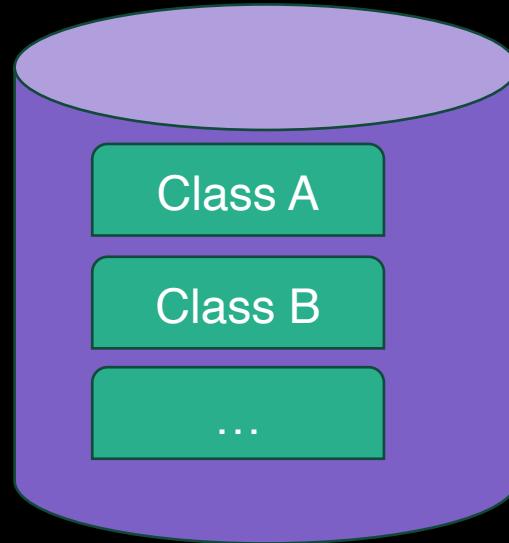
```
class owa_eventDispatcher {  
    function notify(event) {  
        owa_core::API::debug("Notifying listeners of"  
            + event->getEventType());  
    }  
}
```

No Type Hint

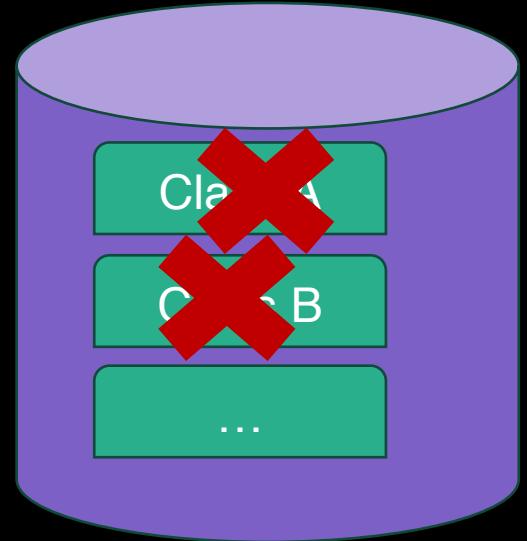
+ event->getEventType());

Has method with this name

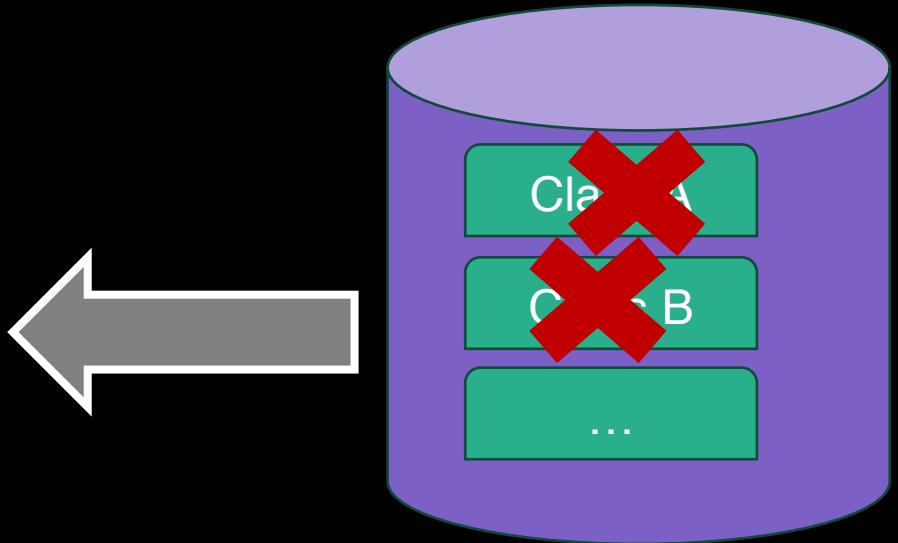
Returns *string* or *no-return-type*



# Analyzing Our Motivating Example



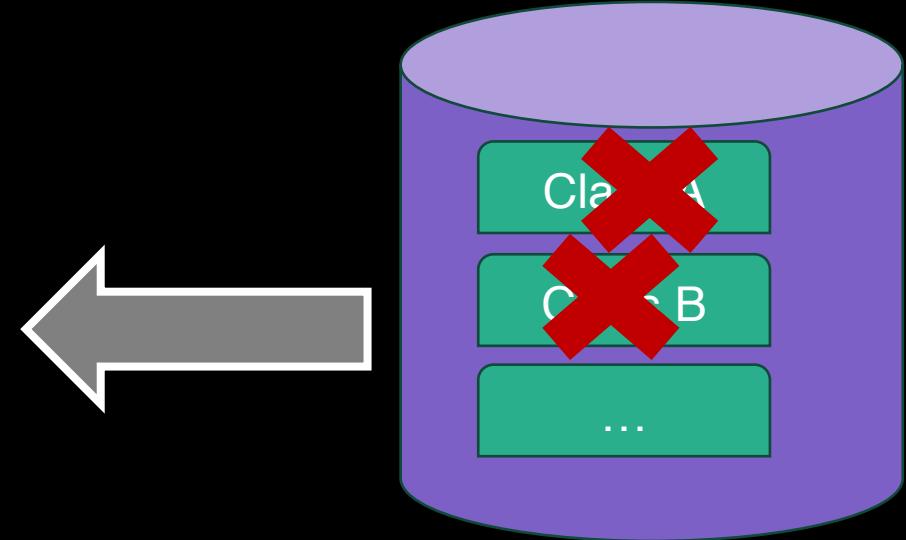
# Analyzing Our Motivating Example



# Analyzing Our Motivating Example

event.php

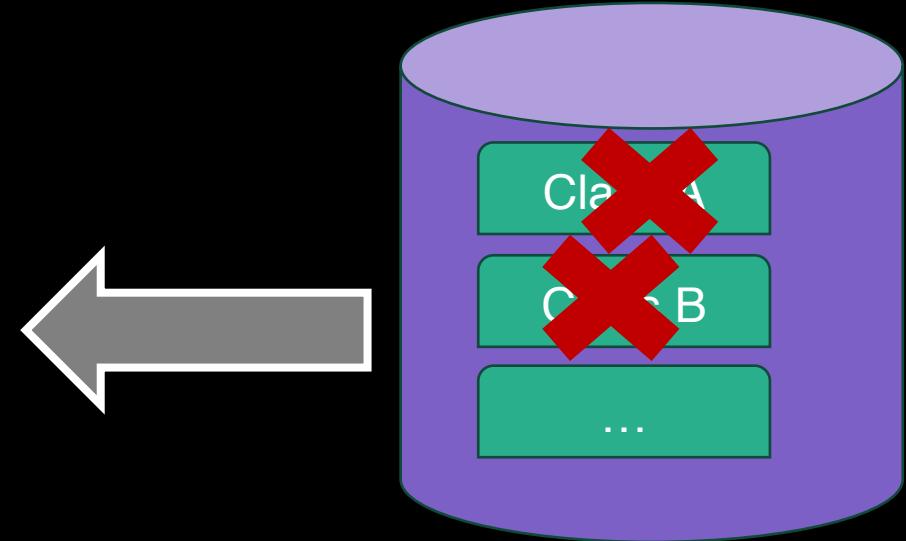
```
class owa_event {  
    function getEventType () { /* snip */}  
}
```



# Analyzing Our Motivating Example

event.php

```
class owa_event {  
    function getEventType () { /* snip */}  
}
```



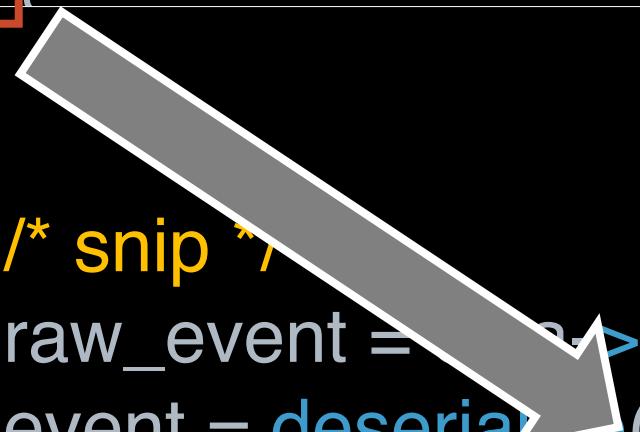
# Analyzing Our Motivating Example

```
event.php
```

```
class owa_event {
    function getEve
}
```

```
/* snip */
raw_event = a->getLast('event');
event = deserialize(raw_event,
++                                ['allowed_classes' => ['owa_event']]);
owa::getEventDispatcher()->notify(event);
```

```
queue.php
```



The diagram illustrates the flow of code between two files: `event.php` and `queue.php`. The `event.php` file contains a class definition and a function `getEve`. Below this, there is a block of code starting with `/* snip */`, followed by the assignment `raw_event = a->getLast('event');`, the deserialization step `event = deserialize(raw_event,`, and finally the dispatching call `owa::getEventDispatcher()->notify(event);`. A large grey arrow originates from the `queue.php` section of `event.php` and points down to the `queue.php` file, indicating that the code in `queue.php` is executed after the code in `event.php`.

# Analyzing Our Motivating Example

```
event.php
```

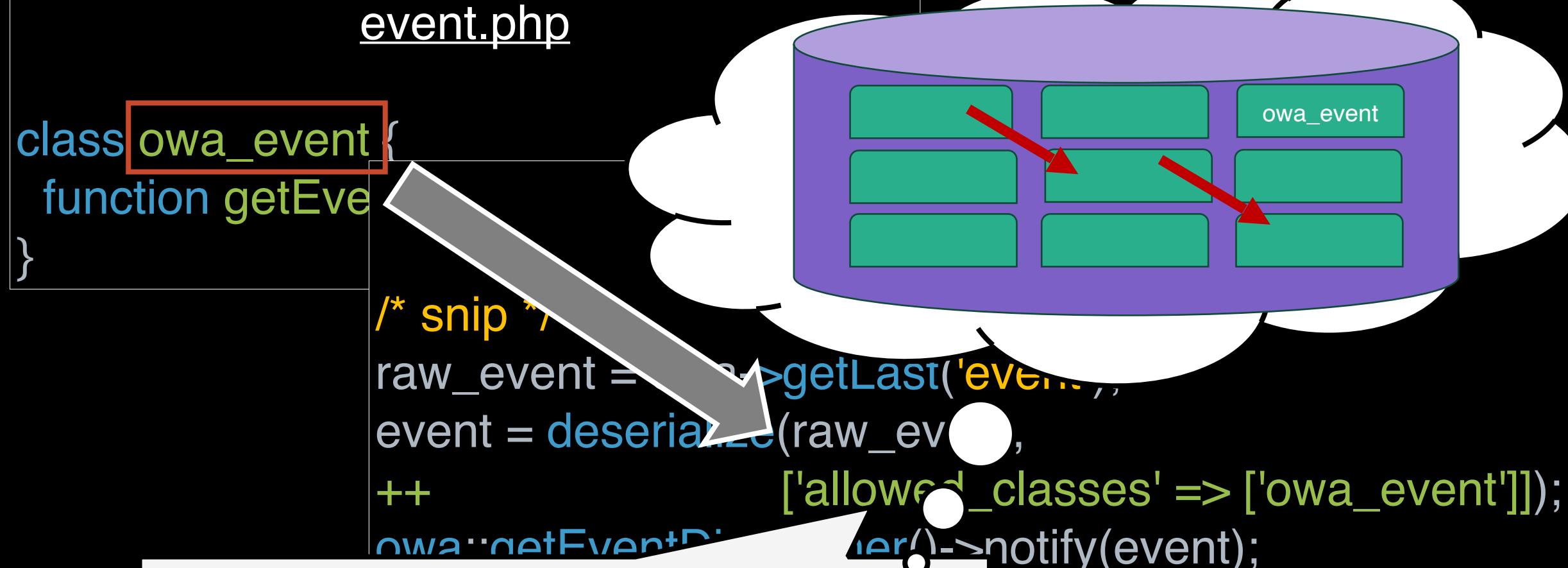
```
class owa_event {
    function getEvent()
}

/* snip */
raw_event = $a->getLast('event');
event = deserialize(raw_event,
++                                ['allowed_classes' => ['owa_event']]);
$owaa->getEventData();
$owaa->notify(event);
```

```
queue.php
```

Stops 14 different exploit chains

# Analyzing Our Motivating Example



Stops 14 different exploit chains

# Analyzing Our Motivating Example

event.php

```
class owa_event {
    function getEve
}

/* snip */
raw_event = >>>getLast('eve...,
event = deserialize(raw_ev
++ 
owa->getEventD...
    [allowed_classes' => ['owa_event']]);
    &event)->notify(event);
```

Stops 14 different exploit chains

# Evaluation

- Compare against **FUGIO**, SOTA automatic exploit generation tool
  - Dynamically collect available classes and composes them into a gadget-chain
- Vulnerability Datasets
  - **FUGIO** – all vulnerable php applications used by previous papers
  - **VULN202X** – A sample of PHP deserialization vulnerabilities published at/after 2020
- Measure:
  - Exploit-building classes blocked (“Positive”)
  - Classes wrongfully excluded (“Negative”)

# Quack Automatically Stops Or Hinders Attacks

Dataset	#CVEs	Exploit-Building Classes [AVG(STD)]	
		Initial Count	% Blocked
FUGIO	7	79 (55)	100%
VULN202x	10	194(114)	97(7)%

# Quack Automatically Stops Or Hinders Attacks

Dataset	#CVEs	Exploit-Building Classes [AVG(STD)]	
		Initial Count	% Blocked
FUGIO	7	79 (55)	100%
VULN202x	10	194(114)	97(7)%

# Protecting Applications Against FUGIO

Application	CVE	# FUGIO-Generated Exploits
Piwik 0.4.5	CVE-2009-4137	
Joomla-3.0.2	CVE-2013-1453	
CubeCart 5.2.0	CVE-2013-1465	
Contao CMS 3.2.4	CVE-2014-1860	
Open Web Analytics	CVE-2014-2294	

# Protecting Applications Against FUGIO

Application	CVE	# FUGIO-Generated Exploits
		Original Version
Piwik 0.4.5	CVE-2009-4137	1
Joomla-3.0.2	CVE-2013-1453	2
CubeCart 5.2.0	CVE-2013-1465	1
Contao CMS 3.2.4	CVE-2014-1860	5
Open Web Analytics	CVE-2014-2294	14

# Protecting Applications Against FUGIO

Application	CVE	# FUGIO-Generated Exploits	
		Original Version	Quack-Protected Version
Piwik 0.4.5	CVE-2009-4137	1	0
Joomla-3.0.2	CVE-2013-1453	2	0
CubeCart 5.2.0	CVE-2013-1465	1	0
Contao CMS 3.2.4	CVE-2014-1860	5	0
Open Web Analytics	CVE-2014-2294	14	0

# Quack Automatically Stops Or Hinders Attacks

Dataset	#CVEs	Exploit-Building Classes [AVG(STD)]	
		Initial Count	% Blocked
FUGIO	7	79 (55)	100%
VULN202x	10	194(114)	97(7)%

# Quack Automatically Stops Or Hinders Attacks

Dataset	#CVEs	Exploit-Building Classes [AVG(STD)]	
		Initial Count	% Blocked
FUGIO	7	79 (55)	100%
VULN202x	10	194(114)	97(7)%

# Protecting Against Recent CVEs

Application	CVE	Exploit-Building Classes		
		#Blocked	#Remaining	%Blocked
ForkCMS 5.8.3	2020-24036	221	23	91%
WP-hotel-booking 10.2.1	2020-29047	103	0	100%
OpenCATS-0.9.5 (1)		288	0	100%
OpenCATS-0.9.5 (2)		232	56	81%
OpenCATS-0.9.5 (3)	2021-25294	288	0	100%
OpenCATS-0.9.5 (4)		288	0	100%
OpenCATS-0.9.5 (5)		232	56	81%
WP-AIOSEO 4.1.0.1	2021-24307	23	0	100%
WP-booking-calendar 9.1.1	2022-1463	96	0	100%
WP-lead-generated 1.23	2023-28667	40	0	100%

# Protecting Against Recent CVEs

Application	CVE	Exploit-Building Classes		
		#Blocked	#Remaining	%Blocked
ForkCMS 5.8.3	2020-24036	221	23	91%
WP-hotel-booking 10.2.1	2020-29047	103	0	100%
OpenCATS-0.9.5 (1)		288	0	100%
OpenCATS-0.9.5 (2)		232	56	81%
OpenCATS-0.9.5 (3)	2021-25294	288	0	100%
OpenCATS-0.9.5 (4)		288	0	100%
OpenCATS-0.9.5 (5)		232	56	81%
WP-AIOSEO 4.1.0.1	2021-24307	23	0	100%
WP-booking-calendar 9.1.1	2022-1463	96	0	100%
WP-lead-generated 1.23	2023-28667	40	0	100%

# Protecting Against Recent CVEs

Application	CVE	Exploit-Building Classes		
		#Blocked	#Remaining	%Blocked
ForkCMS 5.8.3	2020-24036	221	23	91%
WP-hotel-booking 10.2.1	2020-29047	10	0	100%
OpenCATS-0.9.5 (1)		28	0	100%
OpenCATS-0.9.5 (2)				81%
OpenCATS-0.9.5 (3)				
OpenCATS-0.9.5 (4)				
OpenCATS-0.9.5 (5)				
WP-AIOSEO 4.1.0.1				100%
WP-booking-calendar 9.1.1		25		100%
WP-lead-generated 1.23	2023-28667	40	0	100%

call\_user\_func\_array([\$class, \$method], ...)

new \$class(...)

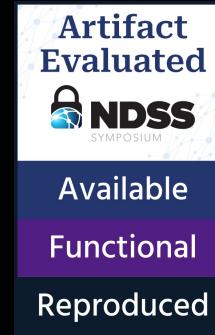
# Performance and Safety

Dataset	#CVEs	Exploit-Building Classes [AVG(STD)]	
		Initial Count	% Blocked
FUGIO	7	79 (55)	100%
VULN202x	10	212(106)	97(7)%

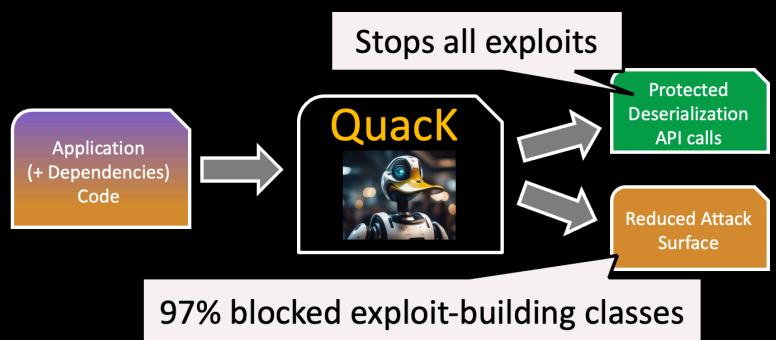
- Favor soundness → **no Negatives**
- Offline project scan: < 7 minutes
- Enforcement incurs **negligible overheads**



<https://github.com/columbia/quack>



### Quack: Securing Applications Against Deserialization Attacks



## Thank You! Questions?

### Stopping Deserialization Exploits

A code snippet from "event.php" shows a class definition for "owa\_event" with a "getEvent" function. The code then includes a comment block, followed by code that decodes a raw event, deserializes it, and checks its allowed classes. A grey arrow points from the "queue.php" file to this code, indicating its context.

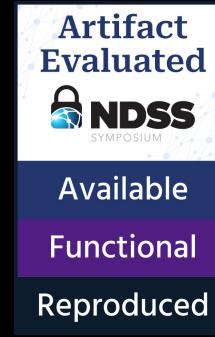
```
event.php
class owa_event {
    function getEvent() {
        /* snip */
        raw_event = owa->getLast('event');
        event = deserialize(base64_decode(raw_event),
        ++
        ['allowed_classes' => ['owa_event']]);
        owa->getEvent();
        owa->notify(event);
    }
}
```

queue.php

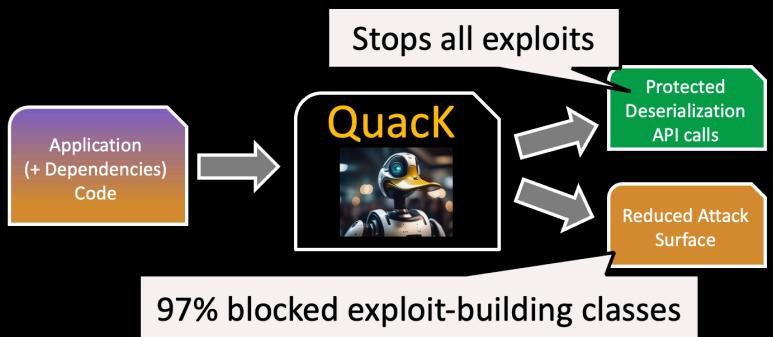
Stops 14 different exploit chains



<https://github.com/columbia/quack>



### Quack: Securing Applications Against Deserialization Attacks



# Thank You! Questions?

### Stopping Deserialization Exploits

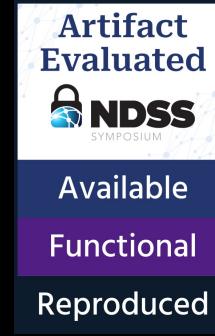
```
event.php
class owa_event {
    function getEvent() {
        /* snip */
        raw_event = owa->getLast('event');
        event = deserialize(base64_decode(raw_event),
        ++
        ['allowed_classes' => ['owa_event']]);
        owa->getEvent();
        owa->notify(event);
    }
}
```

The code snippet shows a class named `owa_event` with a method `getEvent()`. Inside the method, there is a block of code that handles the deserialization of a raw event from the database. The code uses the `base64_decode` function to decode the raw event and then deserializes it into an object. A check is performed to ensure that the allowed classes for deserialization are limited to `'owa_event'`. This prevents exploit chains that rely on deserializing objects from untrusted sources.

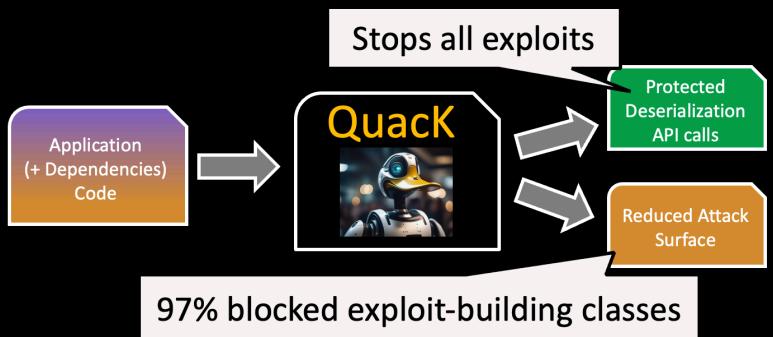
Stops 14 different exploit chains



<https://github.com/columbia/quack>



### Quack: Securing Applications Against Deserialization Attacks



# Thank You! Questions?

### Stopping Deserialization Exploits

```
event.php
class owa_event {
    function getEvent() {
        /* snip */
        raw_event = owa->getLast('event');
        event = deserialize(base64_decode(raw_event),
        ++
        ['allowed_classes' => ['owa_event']]);
        owa->getEvent();
        owa->notify(event);
    }
}
```

The code snippet shows a class named `owa_event` with a method `getEvent()`. Inside the method, it retrieves the last event from `owa`, decodes it using `base64_decode`, and then deserializes it. A check is performed to ensure the deserialized object belongs to the allowed classes, specifically `'owa_event'`. This logic is part of the `queue.php` file.

Stops 14 different exploit chains

54