**LAB 4: Arbitrary object injection in PHP**

1. **Đặc điểm**

Bài lab này là một ví dụ điển hình về lỗ hổng deserialization - một lỗ hổng nguy hiểm mà không phải ai cũng nhận ra mức độ nghiêm trọng. Cơ bản, lỗ hổng này xuất hiện khi một ứng dụng chuyển đổi dữ liệu từ định dạng nhị phân (serialize) trở lại thành đối tượng trong hệ thống (deserialize) mà không có kiểm tra kỹ lưỡng. Nếu kẻ tấn công có thể đưa vào dữ liệu tuỳ ý, họ có khả năng lợi dụng quá trình này để thực thi mã độc ngay trên server.

Bài lab sử dụng session để lưu trữ thông tin về người dùng sau khi họ đăng nhập. Điểm đặc biệt là session này dựa trên một cơ chế serialize/deserialization – nghĩa là khi server tạo session cho người dùng, nó chuyển đổi dữ liệu thành định dạng nhị phân. Sau đó, mỗi khi người dùng gửi request, dữ liệu này sẽ được chuyển đổi ngược lại thành các đối tượng.

Do không có cơ chế kiểm tra chặt chẽ, server dễ dàng nhận và thực thi các đối tượng mà kẻ tấn công gửi lên, kể cả các đối tượng có mã độc. Điều này có nghĩa là kẻ tấn công có thể tiêm một đối tượng với các lệnh nguy hiểm vào session, và khi server giải mã đối tượng đó, mã độc sẽ được thực thi.

Mục tiêu cụ thể trong bài lab này là xóa file morale.txt trong thư mục của người dùng Carlos. Bằng cách lợi dụng cơ chế này, có thể tiêm một đối tượng có chứa lệnh xóa file vào hệ thống.

1. **Nguyên tắc hoạt động của dạng tấn công**

**Serialization và Deserialization:**

Serialization là quá trình chuyển một đối tượng thành chuỗi ký tự hoặc nhị phân để lưu trữ hoặc truyền tải qua mạng. Ngược lại, deserialization là quá trình phục hồi lại đối tượng từ chuỗi đó.

Trong PHP, hai hàm phổ biến dùng cho việc này là serialize() và unserialize(). Khi ứng dụng sử dụng unserialize() trên dữ liệu mà người dùng có thể can thiệp, đó là lúc lỗ hổng xuất hiện.

**Khai thác lỗ hổng bằng cách tiêm đối tượng tùy ý:**

Khi unserialize() xử lý dữ liệu mà không có kiểm tra bảo mật, PHP sẽ tự động phục hồi lại mọi đối tượng được lưu trong chuỗi serialized. Nếu kẻ tấn công biết được các class và phương thức trong mã nguồn, họ có thể tạo ra một chuỗi serialized chứa mã độc, gửi lên server, và tận dụng ứng dụng để thực thi lệnh.

Đặc biệt, nếu trong mã nguồn có các class với phương thức \_\_destruct() (tự động chạy khi đối tượng bị hủy), kẻ tấn công có thể tận dụng phương thức này để thực thi mã mà họ muốn.

**Tạo payload chứa đối tượng độc hại:**

Để khai thác, kẻ tấn công cần tạo một payload chứa đối tượng đã được serialize có thể gây hại. Ví dụ, nếu ứng dụng có một class CustomTemplate với phương thức \_\_destruct() để xóa file, kẻ tấn công có thể dùng class này để tạo một đối tượng có tên file độc hại và xóa file đó.

**Gửi payload lên server:**

Sau khi tạo payload, kẻ tấn công tiêm nó vào ứng dụng qua các điểm nhập liệu như cookie hoặc session. Khi server unserialize payload này, đối tượng CustomTemplate được phục hồi và gọi phương thức \_\_destruct(), xóa file mà kẻ tấn công đã chỉ định.

**Thực thi mã độc:**

Nếu lỗ hổng được khai thác thành công, lệnh mã độc trong payload sẽ được thực thi. Trong bài lab này, nhiệm vụ là xóa file morale.txt của Carlos. Điều này đạt được khi phương thức \_\_destruct() trong đối tượng được phục hồi và thực hiện lệnh unlink() để xóa file.

1. **Mô tả**
   1. **Tóm tắt**

* Đăng nhập vào tài khoản của bạn và chú ý rằng cookie phiên có chứa một đối tượng PHP đã được serialize.
* Từ sơ đồ trang web, nhận thấy rằng trang web tham chiếu đến file /libs/CustomTemplate.php. Nhấp chuột phải vào file và chọn "Send to Repeater".
* Trong Burp Repeater, nhận thấy rằng bạn có thể đọc mã nguồn bằng cách thêm dấu ngã (~) vào tên file trong dòng yêu cầu.
* Trong mã nguồn, nhận thấy rằng class CustomTemplate có chứa phương thức ma thuật \_\_destruct(). Phương thức này sẽ gọi phương thức unlink() trên thuộc tính lock\_file\_path, từ đó xóa file ở đường dẫn này.
* Trong Burp Decoder, sử dụng cú pháp chính xác cho dữ liệu PHP đã được serialize để tạo một đối tượng CustomTemplate với thuộc tính lock\_file\_path được đặt thành /home/carlos/morale.txt. Đảm bảo sử dụng đúng nhãn kiểu dữ liệu và chỉ báo độ dài. Đối tượng cuối cùng nên trông như sau:

"O:14:"CustomTemplate":1:{s:14:"lock\_file\_path";s:23:"/home/carlos/morale.txt";}"

* Mã hóa đối tượng này thành Base64 và URL, sau đó lưu vào clipboard của bạn.
* Gửi một yêu cầu chứa cookie phiên đến Burp Repeater.
* Trong Burp Repeater, thay thế cookie phiên bằng cookie đã sửa đổi trong clipboard của bạn.
* Gửi yêu cầu. Phương thức ma thuật \_\_destruct() sẽ tự động được gọi và sẽ xóa file của Carlos.
  1. **Chi tiết**

|  |
| --- |
| Dùng tài khoản thử nghiệm đăng nhập (wiener:peter) |
| Send to Repeater academyLabHeader |
| Send to Repeater CustomTemplate.php |
| Trong request từ CustomTemplate thêm ~ vào sau php và Send để nhận Response |
| Trong request từ acadamyLabHeader chỉnh Decoded-base64 như trên |
| Sao chép cookie |
| Thay đổi giá trị cookie trong trình duyệt |
| Reset trang và nhận được thông báo như trên |

1. **Phương pháp ngăn chặn**

* Tránh sử dụng unserialize() trên dữ liệu không đáng tin cậy
* Sử dụng json\_decode() thay cho unserialize()
* Triển khai lớp \_\_wakeup() và \_\_destruct() an toàn
* Sử dụng thư viện bảo mật
* Kiểm tra và lọc dữ liệu đầu vào chặt chẽ
* Cập nhật phiên bản PHP và các thư viện liên quan

**LAB 5: Exploiting Java deserialization with Apache Commons**

1. **Đặc điểm**

Lab này nhằm khai thác một lỗ hổng bảo mật trong cơ chế quản lý phiên dựa trên serialization của một ứng dụng Java. Kẻ tấn công có thể thực hiện mã độc thông qua việc gửi các đối tượng serialized độc hại.

Ứng dụng sử dụng một session cookie để lưu trữ một đối tượng Java đã được serialized. Điều này tạo ra một lỗ hổng bảo mật lớn nếu dữ liệu này không được kiểm tra và xác thực đúng cách.

Lab khai thác các gadget chains có sẵn trong thư viện Apache Commons Collections để tạo ra các đối tượng độc hại, cho phép thực thi mã khi các đối tượng này được deserialized.

1. **Nguyên tắc hoạt động của dạng tấn công**

**Serialization và Deserialization:**

Serialization là quá trình chuyển đổi một đối tượng Java thành một chuỗi byte để lưu trữ hoặc truyền tải. Ngược lại, deserialization là quá trình khôi phục đối tượng từ chuỗi byte đó. Nếu không có biện pháp bảo vệ, quá trình này có thể bị lợi dụng.

**Cơ chế phiên trong ứng dụng:**

Ứng dụng sử dụng một session cookie để lưu trữ trạng thái phiên của người dùng. Cookie này chứa một đối tượng Java đã được serialized, cho phép ứng dụng xác định người dùng giữa các yêu cầu HTTP. Tuy nhiên, nếu không được kiểm tra đúng cách, đây có thể trở thành điểm yếu bảo mật.

**Lỗ hổng trong Deserialization:**

Nếu ứng dụng không thực hiện kiểm tra tính hợp lệ trước khi deserialization, kẻ tấn công có thể gửi một đối tượng serialized độc hại, chứa mã hoặc lệnh mà họ muốn thực thi. Lỗ hổng này có thể dẫn đến việc thực hiện mã từ xa, gây ra hậu quả nghiêm trọng.

**Sử dụng Apache Commons Collections:**

Lab này sử dụng các gadget chains có trong thư viện Apache Commons Collections. Những gadget này cho phép tạo ra các đối tượng mà khi được deserialized sẽ thực thi mã độc. Kẻ tấn công tận dụng các đối tượng này để thực hiện hành vi không mong muốn.

**Quy trình khai thác:**

Tạo payload: Người tham gia sử dụng công cụ ysoserial để tạo ra một đối tượng serialized độc hại, chứa payload cho lệnh mà họ muốn thực thi (ví dụ: xóa tệp).

Chỉnh sửa session cookie: Sau khi tạo thành công đối tượng độc hại, người tham gia sẽ thay thế giá trị của session cookie hiện tại bằng đối tượng serialized độc hại.

Gửi yêu cầu: Khi gửi yêu cầu với cookie đã chỉnh sửa, ứng dụng sẽ thực hiện deserialization đối tượng đó. Nếu không có kiểm tra, nó sẽ thực thi mã chứa trong payload.

**Kết quả:**

Khi đối tượng độc hại được deserialized, payload sẽ được thực thi, dẫn đến việc thực hiện các lệnh không mong muốn như xóa tệp morale.txt. Điều này chứng minh rằng kẻ tấn công có khả năng khai thác lỗ hổng trong cơ chế deserialization của ứng dụng, cho thấy tầm quan trọng của việc bảo vệ dữ liệu đầu vào trong các ứng dụng Java.

1. **Mô tả**
   1. **Tóm tắt**

Đăng nhập vào tài khoản thử nghiệm và quan sát rằng cookie phiên chứa một đối tượng Java đã được serialized. Gửi một yêu cầu chứa cookie phiên đến Burp Repeater.

Tải công cụ "ysoserial" về và thực hiện lệnh sau. Điều này sẽ tạo ra một đối tượng serialized mã hóa Base64 chứa payload:

Từ phiên bản Java 16 trở lên:

*java -jar ysoserial-all.jar \*

*--add-opens=java.xml/com.sun.org.apache.xalan.internal.xsltc.trax=ALL-UNNAMED \*

*--add-opens=java.xml/com.sun.org.apache.xalan.internal.xsltc.runtime=ALL-UNNAMED \*

*--add-opens=java.base/java.net=ALL-UNNAMED \*

*--add-opens=java.base/java.util=ALL-UNNAMED \*

*CommonsCollections4 'rm /home/carlos/morale.txt' | base64*

Từ phiên bản Java 15 trở về trước:

*java -jar ysoserial-all.jar CommonsCollections4 'rm /home/carlos/morale.txt' | base64*

Trong Burp Repeater, thay thế cookie phiên bằng cookie độc hại mà vừa tạo. Chọn toàn bộ cookie và sau đó mã hóa URL.

Gửi yêu cầu để hoàn thành lab.

* 1. **Chi tiết**

|  |
| --- |
| Vào link git <https://github.com/frohoff/ysoserial>, kéo xuống dưới và download file về |
| Dùng tài khoản wiener:peter để đăng nhập |
| Send to Repeater trong request login |
| Trong terminal, cd đến thư mục chứa file ysoserial-all.jar |
| Chọn phiên bản CommonsCollections mới nhất, copy đoạn mã trên |
| Paste đoạn mã trên vào Cookie và Send |
| URL-encode all characters |
| Sau khi Send thì hoàn thành bài lab |

1. **Phương pháp ngăn chặn**

* Kiểm tra và xác thực dữ liệu đầu vào:
* Sử dụng định dạng dữ liệu an toàn:
* Hạn chế quyền truy cập:
* Sử dụng các biện pháp mã hóa:
* Giám sát và ghi lại hoạt động:
* Cập nhật và vá lỗi:
* Sử dụng các thư viện an toàn: