|  |  |
| --- | --- |
|  | BỘ GIÁO DỤC VÀ ĐÀO TẠO  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM** |

**ĐỒ ÁN CHUYÊN NGÀNH**

**CROSS-SITE SCRIPTING**

**(XSS)**

Ngành: **AN TOÀN THÔNG TIN**

Chuyên ngành: **AN TOÀN THÔNG TIN**

Giảng viên hướng dẫn : Ths. Dương Minh Chiến

Sinh viên thực hiện :

MSSV: 1811770177 Tên: Nguyễn Minh Triết

Lớp: 18DATA1

TP. Hồ Chí Minh, 2021

# LỜI MỞ ĐẦU

Ngày nay, khi Internet được phổ biến rộng rãi, các tổ chức, cá nhân đều có nhu cầu giới thiệu thông tin của mình trên xa lộ thông tin cũng như thực hiện các phiên giao dịch trực tuyến. Vấn đề nảy sinh là khi phạm vi ứng dụng của các ứng dụng Web ngày càng mở rộng thì khả năng xuất hiện lỗi (lỗ hổng) và bị tấn công càng cao, trở thành đối tượng cho nhiều người tấn công với các mục đích khác nhau.

Bên cạnh đó, các hình thức phá hoại ứng dụng Web cũng trở nên tinh vi và phức tạp hơn. Trong đó không thể không kể tới loại tấn công XSS ( Cross-Site Scripting ), đây là một kỹ thuật tân công website rất phổ biến, đem tới mối nguy hiểm rất lớn cho người dùng, các tin tặc có thể sử dụng kỹ thuật tấn công này để ăn căp cookíe, mật khẩu, tiêm các loại trojan … và nhiều hành động nguy hiểm khác, gây thiệt hại về cả mặt kinh tế lẫn tinh thần cho người bị tấn công.

Xuất phát từ những thực tế đó, đồ án đi sâu vào việc tìm hiểu cách thức tấn công và phòng chống các cuộc tấn công XSS (Cross-Site Scripting ), nhằm cho mọi người thấy được sự nguy hiểm và cách để có thể phòng chống một cuộc tấn công XSS.

# LỜI CAM ĐOAN

Chúng tôi xin cam đoan đồ án trên là công trình nghiên cứu của chúng tôi dưới sự hướng dẫn của Ths. Dương Minh Chiến. Những nhận định được nêu ra trong đồ án cũng là kết quả từ sự nghiên cứu trực tiếp, nghiêm túc, độc lập của chúng tôi dựa vào các cơ sở tìm kiếm, hiểu biết và nghiên cứu tài liệu khoa học hay bản dịch khác đã được công bố. Đồ án vẫn sẽ giúp đảm bảo được tính khách quan, trung thực và khoa học.

# LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn Ths. Dương Minh Chiến, là đã tận tâm hướng dẫn chúng em qua từng buổi học trên lớp cũng như những buổi nói chuyện, thảo luận về lĩnh vực sáng tạo trong nghiên cứu khoa học.

Em cũng xin bày tỏ lòng biết ơn đến ban lãnh đạo của Trường Đại học Công Nghệ TPHCM và các Khoa Phòng ban chức năng đã trực tiếp và gián tiếp giúp đỡ em trong suốt quá trình học tập và nghiên cứu đề tài này.

Với điều kiện thời gian cũng như kinh nghiệm còn hạn chế của một học viên, bài báo cáo này không thể tránh được những thiếu sót. Chúng em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các quý thầy cô để tôi có điều kiện bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

Xin chân thành cảm ơn.

# **MỤC LỤC**

[LỜI MỞ ĐẦU 1](#_Toc88989429)

[LỜI CAM ĐOAN 2](#_Toc88989430)

[LỜI CẢM ƠN 3](#_Toc88989431)

[MỤC LỤC 4](#_Toc88989432)

[Chương 1. TỔNG QUAN 6](#_Toc88989433)

[1.1 Tổng quan đề tài 6](#_Toc88989434)

[1.1.1 Lịch sử 7](#_Toc88989435)

[1.1.2 Sơ lược về XSS 7](#_Toc88989436)

[1.2 Mục tiêu đề tài 6](#_Toc88989437)

[1.3 Cấu trúc đề tài 6](#_Toc88989438)

[Chương 2. CƠ SỞ LÝ THUYẾT 6](#_Toc88989439)

[2.1 Định nghĩa 6](#_Toc88989440)

[2.2 Các loại tấn công XSS 7](#_Toc88989441)

[2.2.1 Reflected XSS 8](#_Toc88989442)

[2.2.2 Stored XSS 8](#_Toc88989443)

[2.2.3 DOM Based XSS 8](#_Toc88989444)

[2.3 Mức độ nguy hiểm của XSS 7](#_Toc88989445)

[2.4 XSS có thể làm những gì 7](#_Toc88989446)

[2.4.1 Đánh cắp phiên người dùng trên trang web 8](#_Toc88989447)

[2.4.2 Lấy thông tin các phím người dùng nhập 8](#_Toc88989448)

[2.4.3 Làm giả giao diện trang web 8](#_Toc88989449)

[2.4.4 Thu thập thông tin người dùng bằng trang HTML giả 8](#_Toc88989450)

[2.4.5 Gây sự cố trình duyệt (từ chối dịch vụ cục bộ) 8](#_Toc88989451)

[2.4.6 Trực tiếp tải tệp 8](#_Toc88989452)

[2.4.7 Chuyển hướng trình duyệt 8](#_Toc88989453)

[2.5 Tìm kiếm và kiểm tra lỗ hổng 7](#_Toc88989454)

[2.5.1 Sử dụng BrupSuite 7](#_Toc88989455)

[2.5.2 XSS giữa các thẻ HTML 9](#_Toc88989456)

[2.5.3 XSS trong thuộc tính của thẻ HTML 10](#_Toc88989457)

[2.5.4 XSS trong javascript 11](#_Toc88989458)

[2.5.5 Vậy XSS thuộc giai đoạn nào trong quá trình pentestweb 11](#_Toc88989459)

[2.6 Quy trình tấn cuộc tấn công XSS 11](#_Toc88989460)

[2.7 Biện pháp phòng chống tấn công XSS 11](#_Toc88989461)

[2.7.1 Reflected và Stored XSS 12](#_Toc88989462)

[2.7.2 DOM XSS 12](#_Toc88989463)

[Chương 3. KẾT QUẢ THỰC NGHIỆM 12](#_Toc88989464)

[3.1 Thực hiện tấn công XSS 12](#_Toc88989465)

[3.2 Thực hiện ngăn chặn XSS 12](#_Toc88989466)

[Chương 4. KẾT LUẬN 12](#_Toc88989467)

[TÀI LIỆU THAM KHẢO 13](#_Toc88989468)

# TỔNG QUAN

## Tổng quan đề tài

### Lịch sử

Câu chuyện về XSS bắt đầu vào cuối năm 1999 với một nhóm nhỏ các kỹ sư bảo mật của Microsoft. Trung tâm phản hồi bảo mật của Microsoft và nhóm bảo mật của Microsoft Internet Explorer đã được nghe về các cuộc tấn công mà một số trang web đang gặp phải trong đó các thẻ tập lệnh và thẻ hình ảnh độc hại đã được đưa vào các trang html. Một số cuộc tấn công bắt nguồn từ các liên kết được gửi đến những nạn nhân không cảnh giác, lấy cookie của người dùng và đưa chúng cho các bên thứ ba. Các cuộc tấn công rất dai dẳng về bản chất, lưu lượng được lưu trữ và đưa vào lặp đi lặp lại, ảnh hưởng đến bất kỳ ai truy cập vào các trang . Michael Barrett, PayPal CISO từ năm 2006 đến năm 2013, mô tả việc phát hiện ra một khai thác XSS được phản ánh trong phần mềm American Express vào khoảng thời gian đó. Việc khai thác này đã được chứng minh trong chuỗi quản lý American Express và thậm chí đã được mô tả cho nhóm Trung tâm Bảo mật của Microsoft trong một hội nghị vào mùa thu năm 1999. Đến tháng 12, nhóm kỹ sư Bảo mật của Microsoft đã chính thức bắt đầu nghiên cứu về lỗ hổng bảo mật và đến tháng 2 năm thiên niên kỷ mới, đã xuất bản một báo cáo chi tiết về lỗ hổng bảo mật kết hợp với CERT. Trong bài viết này, cuộc tấn công đã được đặt tên cho nó, Cross Site Scripting (XSS).

Tập lệnh trang web chéo được đặt tên như vậy bởi vì các trường hợp đầu tiên là các tập lệnh được gửi từ một trang web và sau đó được đưa vào trang web đích. Ví dụ đầu tiên được sử dụng trong bài viết là một phân loại XSS được gọi là Reflected XSS. Reflected XSS liên quan đến việc gửi đầu vào do người dùng kiểm soát thông qua một yêu cầu, cho phép máy chủ tạo html với đầu vào được bao gồm, sau đó trả lại html được yêu cầu có kèm theo tập lệnh độc hại.

Tuy nhiên, Stored XSS cũng thịnh hành vào thời điểm đó. Vào cuối những năm 1990, người ta thường tìm thấy chức năng của trang web cho phép các dữ liệu độc hại như những thứ trong cuộc tấn công dựa trên Reflected XSS vẫn tồn tại trong các phiên duyệt web. Các dữ liệu độc hại này sẽ được lưu trữ trong kho dữ liệu thông qua html. Thường thì các dữ liệu độc hại sẽ được lưu trữ trong cơ sở dữ liệu, nhưng email, tệp từ xa và nhật ký lỗi cũng rất phổ biến để tìm dữ liệu độc hại được lưu trữ. Vì các dữ liệu này đều có thể được lưu trữ và phản ánh, các cuộc tấn công này được nhóm lại với nhau dưới một tiêu đề: XSS.

Reflected XSS và Stored XSS có 2 sự khác biệt lớn trong quá trình tấn công.

Thứ nhất, để khai thác Reflected XSS, hacker phải lừa được nạn nhân truy cập vào URL của mình. Còn Stored XSS không cần phải thực hiện việc này, sau khi chèn được mã nguy hiểm vào CSDL của ứng dụng, hacker chỉ việc ngồi chờ nạn nhân tự động truy cập vào. Với nạn nhân, việc này là hoàn toàn bình thường vì họ không hề hay biết dữ liệu mình truy cập đã bị nhiễm độc.

Thứ 2, mục tiêu của hacker sẽ dễ dàng đạt được hơn nếu tại thời điểm tấn công nạn nhân vẫn trong phiên làm việc(session) của ứng dụng web. Với Reflected XSS, hacker có thể thuyết phục hay lừa nạn nhân đăng nhập rồi truy cập đến URL mà hắn ta cung cấp để thực thi mã độc. Nhưng Stored XSS thì khác, vì mã độc đã được lưu trong CSDL Web nên bất cứ khi nào người dùng truy cập các chức năng liên quan thì mã độc sẽ được thực thi, và nhiều khả năng là những chức năng này yêu cầu phải xác thực(đăng nhập) trước nên hiển nhiên trong thời gian này người dùng vẫn đang trong phiên làm việc.

Từ những điều này có thể thấy Stored XSS nguy hiểm hơn Reflected XSS rất nhiều, đối tượng bị ảnh hưởng có thế là tất cả nhưng người sử dụng ứng dụng web đó. Và nếu nạn nhân có vai trò quản trị thì còn có nguy cơ bị chiếm quyền điều khiển web.

Những khác biệt rõ rệt về tính bền bỉ và phạm vi này là động lực trong việc xác định các phân loại XSS cho đến năm 2005. Reflected XSS và Stored XSS là hai cách tấn công chính và bất kỳ biến thể nào khác đều dựa trên Reflected XSS. Tuy nhiên, đến năm 2005, JavaScript đã chạy tràn lan. JQuery đã được một năm kể từ khi được tạo ra và các nhà phát triển triển khai chức năng trong Document Object Model mà Brendan Eich không bao giờ có ý định làm nó. Nhận thấy phạm vi ngày càng gia tăng của JavaScript dựa trên ứng dụng khách như một cơn bão sắp tới, Amit Klein đề xuất rằng thế giới bảo mật nên công nhận cách thức tấn công thứ ba của XSS mà ông gọi là XSS dựa trên DOM.

DOM-Based XSS đề cập đến một lỗ hổng tập hợp dữ liệu từ một phần của Document Object Model và ghi nó vào một phần khác của Document Object Model trong trình duyệt. Các khái niệm đầu vào và đầu ra này có thể được mô tả chính thức như là nguồn dữ liệu và dữ liệu chìm. Nếu dữ liệu nhập qua nguồn DOM nằm trong tầm kiểm soát của người dùng và bồn rửa DOM ghi dữ liệu đó mà không cần làm sạch, chức năng đó có thể dẫn đến lỗ hổng XSS dựa trên DOM.

XSS đã gây ra những thiệt hiệt rất lớn cho rất nhiều trang web trong đó có cả Google,

Paypal và Uber

Google (2017): Một chuyên gia bảo mật đã kiếm được 10000$ với khi tiềm ra được một lỗi XSS trên google map, với lỗi này google map ước tính đã thiệt hại gần 5000$, Zohar Shachar, người đứng đầu bộ phận bảo mật ứng dụng tại Wix.com, đã báo cáo lỗi này cho Google vào ngày 23 tháng 4 và được thưởng 5.000 USD ngay sau đó. Google đã tiết lộ công khai vấn đề, tuyên bố nó "đã được khắc phục" vào ngày 7 tháng 6. Vài phút sau khi Shachar được thông báo về bản vá và phần thưởng thanh toán tiền thưởng, anh ấy nói rằng anh ấy đã tìm thấy một cách bỏ qua cho bản sửa lỗi của Google Maps. Điều đó cuối cùng đã kiếm được cho anh ta thêm 5.000 đô la nữa.

Uber (21/11/2018): Do hai lỗ hổng IDOR trong Tealium, có thể xâm nhập tài khoản của quản trị viên và đưa Javascript tùy ý vào https://tags.tiqcdn.com/utag/uber/\*, kẻ tấn công có thể tận dụng một cuộc tấn công XSS được lưu trữ trên một số Uber các miền. Ngoài ra, bất kỳ người dùng nào cũng có thể đặt lại mật khẩu và MFA của người dùng Tealium, cho phép kẻ tấn công chiếm đoạt tài khoản và thay mặt họ sửa đổi mã. Mặc dù Uber không sở hữu Tealium, chúng tôi trả tiền thưởng dựa trên tác động. Vì chúng tôi đã triển khai rộng rãi điều này trên nhiều miền uber.com khác nhau, nên chúng tôi đã trao một khoản tiền thưởng tương đương với tác động của lỗi này. Lỗi đã được @mdv tiết lộ cho nhà cung cấp và đã được vá cho tất cả người dùng Tealium, với lỗi này uber ước tính có thể thiệt hại rất lớn những vì phát hiện ra kịp thời nên đã chi một khoản 6000$ tiền thưởng dành cho người phát hiện ra nó.

Paypal (080/8/2019): Do cấu hình trong giao diện người dùng, máy chủ bộ nhớ đệm, nhà nghiên cứu có thể sử dụng tính năng nhập lậu yêu cầu để chuyển yêu cầu trang thành chuyển hướng được lưu trong bộ nhớ cache. Nếu chuyển hướng được lưu trong bộ nhớ cache được truy cập bởi một người dùng hợp pháp, nội dung của kẻ tấn công sẽ được hiển thị thay vì trang được yêu cầu. Mặc dù điều này sẽ không ảnh hưởng đến bất kỳ dữ liệu back-end nào, nhưng điều này có thể ảnh hưởng đến tính toàn vẹn của một số trang nhất định, bao gồm cả khả năng can thiệp vào trang đăng nhập. PayPal đã làm việc với nhà nghiên cứu và các nhóm kỹ thuật của chúng tôi để khắc phục sự cố và xác nhận rằng không có bằng chứng về các cuộc tấn công trong thế giới thực, ước tính rằng sự cố này có thể gây thiệt hại lên đến hàng trăm ngàn đô, paypal đã trả thưởng cho nhà nghiên cứu 18900$ khi phát hiện ra lỗi này.

### Sơ lược về XSS

Cross-Site Scripting hay còn được gọi tắt là XSS (thay vì gọi tắt là CSS để tránh nhầm lẫn với CSS-Cascading Style Sheet của HTML) là một kĩ thuật tấn công bằng cách chèn những đoạn mã độc vào một ứng dụng web, và trình duyệt của người dùng thực thi đoạn mã đó như thể nó là một phần của trang web đến từ một nguồn đáng tin cậy. Những mã độc này thường được viết với ngôn ngữ lập trình phía client như Javascript, HTML, VBScript, Flash… Tuy nhiên, cách tấn công này thông thường sử dụng Javascript và HTML.

Đây là phương pháp tấn công nhắm chủ yếu về phía người dùng (client site). Nguyên nhân chính của loại tấn công này là xác thực đầu vào dữ liệu người dùng không phù hợp, dữ liệu độc hại từ đầu vào có thể xâm nhập vào dữ liệu đầu ra.

Cách thức hoạt động của XSS Cross-site scripting hoạt động bằng cách điều khiển một trang web dễ bị tấn công để nó trả về JavaScript độc hại cho người dùng. Khi mã độc thực thi trên trình duyệt của nạn nhân, kẻ tấn công hoàn toàn có thể thực hiện các tương tác theo ý mình trên máy người dùng.

## Mục tiêu đề tài

Với những mối nguy hại mà việc tấn công XSS đem lại, chúng tôi muốn tìm hiểu cách thức mà các tin tặc sử dụng để tấn công những người dùng, tư đó đưa ra những kết luận về mức độ nguy hiểm và các biện pháp phòng tránh một cách hiệu quả nhất, đem lại cho người dùng những trải nghiệm tốt hơn, an toàn hơn trên môi trường mạng internet.

## Cấu trúc đề tài

Chương 1. TỔNG QUAN

1. Tổn quan đề tài
2. Mục tiêu đề tài
3. Cấu trúc đề tài

Chương 2. CƠ SỞ LÝ THUYẾT

1. Định nghĩa
2. Các loại tấn công XSS
3. Mức độ nguy hiểm của XSS
4. XSS có thể làm những gì
5. Tìm kiếm và kiểm tra lỗ hổng
6. Quy trình tấn công XSS
7. Biện pháp phòng chống tấn công XSS

Chương 3. KẾT QUẢ THỰC NGHIỆM

1. Thực hiện tấn công XSS
2. Thực hiện ngăn chặn XSS

Chương 4. KẾT LUẬN

# CƠ SỞ LÝ THUYẾT

## Định nghĩa

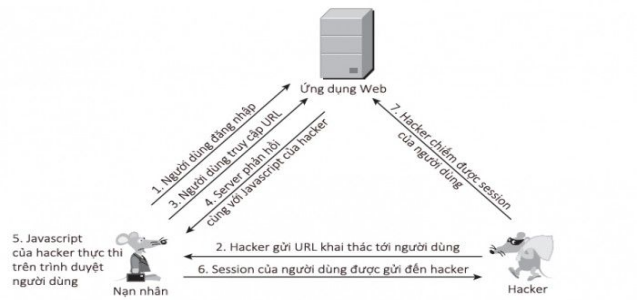
XSS (là viết tắt của cụm từ Cross-Site Scripting) hiểu đơn giản là một hình thức tấn công bằng mã độc phổ biến, trong đó, hacker sẽ lợi dụng các lỗ hổng để chèn các mã script vào, sau đó gửi cho người dùng (hoặc người dùng vô tình truy cập vào trang bị nhiễm mã độc). Qua đó đánh cắp thông tin Cookie của người dùng và dùng nó để đăng nhập các tài khoản trên website đã bị nhiễm mã độc.

Đây là hình thức tấn công đơn giản nhưng gây nhiều thiệt hại lớn đối với các website như: ngân hàng, thương mại điện tử. Đồng thời, đây cũng là một vấn đề bảo mật cần chú tâm khi lập trình một ứng dụng web.

## Các loại tấn công XSS

### Reflected XSS

Trong hình thức này, kẻ tấn công thường gắn thêm đoạn mã độc vào URL của website chúng ta và gởi đến nạn nhân, nếu nạn nhân truy cập URL đó thì sẽ bị dính mã độc. Điều này xảy ra do ta không chú ý filter input từ URL của website mình. Hình thức này khác với Stored là khi nào nạn nhân click vào đường dẫn có chứa mã độc thì mới bị tấn công.



Hình 2.1 Sơ đồ cách Reflected XSS hoạt động

Phương thức tấn công:

Giả sử một trang web có chức năng tìm kiếm môt cụm từ mà người dùng cung cấp trong thanh URL: <https://examp.com/search?id=123>. Web sẽ phản hồi với URL này và lặp lại nội dụng được tìm kiếm:

<p>You searched for: 123</p>

Trong trường hợp web không thực hiện quá trình xử lý nội dung tìm kiếm một cách an toàn, tại đây kẻ tấn công có thể lợi dụng để thực hiện một cuộc tấn công như sau: [https://examp.com/search?id=<script>a](https://examp.com/search?id=%3cscript%3ea)lert(‘XSS’)</script>.

URL này sẽ phản hồi lại và đoạn mã độc sẽ được thực thì, trong trường hợp trên thì một thông báo sẽ được hiện lên với nội dung XSS.

<p>You searched for: [<script>a](https://examp.com/search?id=%3cscript%3e/*)lert(‘XSS’)</script></p>

Cuối cùng, kẻ tấn công sẽ phát tán URL này cho nạn nhân thông qua: email, nhắn tin, bình luận,… .Khi nạn nhân truy cập vào URL độc hại này ngay lập tức thông báo XSS được hiện lên.

### Stored XSS

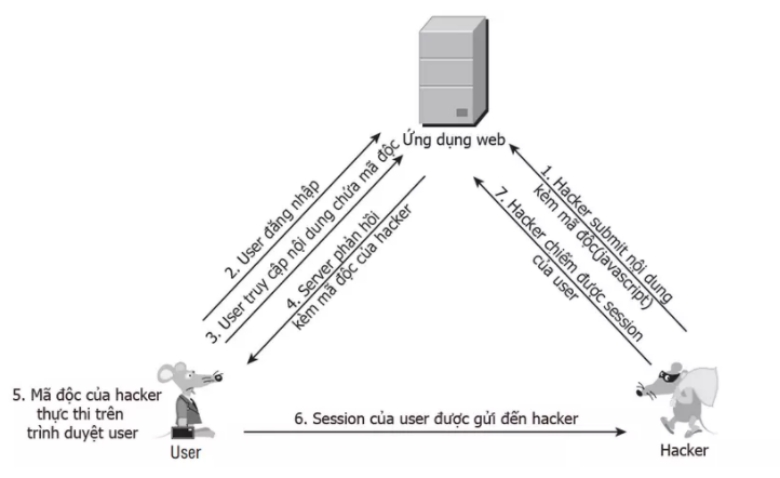
Stored XSS là dạng tấn công mà tin tặc chèn trực tiếp các mã độc vào cơ sở dữ liệu của trang web. Dạng tấn công này xảy ra khi các dữ liệu được gửi lên máy chủ không được kiểm tra kỹ lưỡng mà được lưu trực tiếp vào cơ sở dữ liệu. Khi người dùng truy cập vào trang web này thì những đoạn script độc hại sẽ được thực thi chung với quá trình tải trang web.

Khác với kiểu tấn công Reflect, Stored XSS hướng đến nhiều đối tượng tận công hơn, bằng cách tiêm vào cơ sở dữ liệu của trang web những đoạn mã độc hại thông qua các điểm như: các ô góp ý, các ô bình luận …không được lọc dữ liệu đầu vào.

Để thực hiện kỹ thuật Stored XSS, tin tặc cần thực hiện tối thiểu qua hai bước:

* Đầu tiên tin tặc sẽ thông qua các ô như : ghi chú, bình luận, góp ý… để chèn một đoạn mã độc vào trong cơ sở dữ liệu của trang web
* Tiếp theo khi người dùng truy cập vào trang web và thực hiện thao tác có liên quan đến dữ liệu có chứa mã độc này, đoạn mã sẽ được kích hoạt trên chính trình duyệt của người dùng.

Ngoài ra các tin tắc còn có thể tấn công theo một kịch bản khác:



Hình 2.2 Sơ đồ cách Stored XSS hoạt động

Như ở trên chúng ta có thể thấy Stored XSS khá giống với Reflect XSS, tuy nhiên vẫn có 2 sự khác biệt khá lớn trong quá trình tấn công:

Thứ nhất, để khai thác Reflect XSS, tin tặc phải lừa được nạn nhân truy cập vào URL của mình. Còn đối với Stored XSS, vì đoạn mã đã được lưu trên cơ sở dữ liệu nên tin tặc chỉ cần ngồi chờ nạn nhân thao tác đến phần dữ liệu có chữa mã độc.

Thứ hai, việc chiếm phiên làm việc sẽ dễ dàng hơn, đối với Reflect XSS, phiên làm việc có thể sẽ bị hết hạn sau một khoảng thời gian, khi đó cho dù nạn nhân có truy cập vào URL của tin tặc thì việc chiếm được phiên làm việc vẫn là không thể, tuy nhiên đối với Stored XSS thì không, vì đoạn mã độc đã được lưu trên cơ sở dữ liệu của chính trang web nên việc chiếm phiên làm việc sẽ luôn sảy ra khi người dùng truy cập vào dữ liệu đó.

Từ những điều trên, ta có thể thấy Stored XSS nguy hiểm hơn Reflect XSS rất nhiều, số lượng nạn nhân có thể là tất cả những người truy cập vào trang web đó, nếu là quản trị viên thì tin tắc còn có thể chiếm quyền điều khiển trang web.

### DOM Based XSS

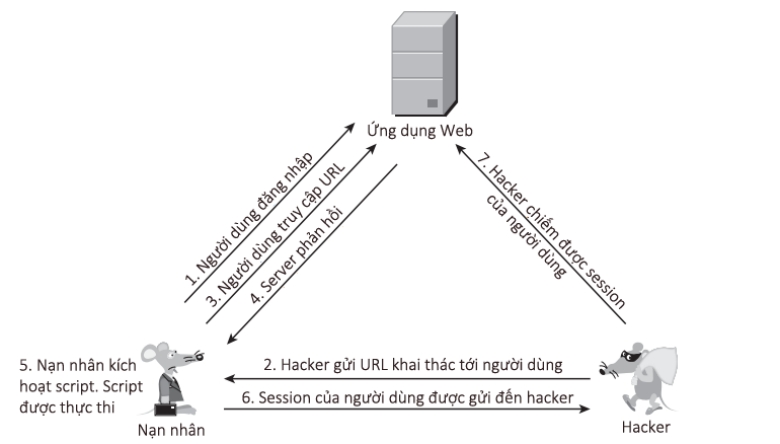
DOM Based XSS là một kỹ thuật khai thác XSS dựa trên việc thay đổi cấu trúc DOM (Document Object Model) của trang web, cụ thể là HTML.Trong DOM-based XSS, chuỗi mã độc sẽ không được xử lý bởi trình duyệt nạn nhân cho đến khi JavaScript hợp pháp của website được thực thi.

Trong DOM-based XSS, không có tập lệnh độc hại nào được chèn vào trang, tập lệnh duy nhất được tự động thực thi trong khi tải trang là một phần hợp pháp của trang. Vấn đề là tập lệnh hợp pháp này sử dụng dữ liệu của người dùng để thêm HTML vào trang. Vì chuỗi độc hại được chèn vào trang bằng cách sử dụng innerHTML, nên nó được phân tích cú pháp dưới dạng HTML, khiến tập lệnh độc hại được thực thi.

Để thực hiện tấn công, tin tặc sẽ cần thực hiện các bước sau:

* Gửi cho nạn nhân một URL có chứa một đoạn HTML và JavaScript.
* Khi nạn nhân truy cập vào URL đó và thực hiện thao tác khiến JavaScript kích hoạt đoạn mã độc sẽ được thực thi, khi đó tin tặc sẽ có thể chiếm phiên làm việc của nạn nhân.

Dưới đây là mô hình thực hiện cuộc tấn công:



Hình 2.3 Sơ đồ cách DOM XSS hoạt động

Có thể thấy DOM base XSS là một kỹ thuật tấn công nguy hiểm dó chúng không thể bị phát hiện từ phía máy chủ, các tin tặc có thể dựa vào kiểu tấn công này để thực thi các loại tấn công khác, đem lại mối nguy hại rất lớn đối với những người dùng mạng hiện nay.

## Mức độ nguy hiểm của XSS

XSS được biết đến là lỗi phổ biến nhất và luôn nằm trong top 10 những lỗi bảo mật nguy hiểm được xác nhận bởi OSWAP.

Hầu hết mọi người thường nghĩ XSS không hề nguy hiểm so với các lỗi như: SQL injection, CSRF,… , có thể vì những suy nghĩ về XSS đơn giản chỉ là làm hiện lên một thông báo, chuyển hướng, ghi vài dòng vào trang web. Do đó nhiều người lập trình web thường chủ quan khi không lọc các dữ liệu đầu vào, đầu ra, từ đó đã tạo nền móng cho những kẻ tấn công khai thác lỗ hổng này.

Về mức độ nguy hiểm kẻ tấn công có thể tận dụng để:

* Phát tán worms trên các trang mạng xã hội. Facebook, Twitter và YouTube đều đã bị tấn công thành công theo cách này.
* Session hijacking: JavaScript độc hại có thể gửi session ID đến một trang web từ xa dưới sự kiểm soát của tin tặc, cho phép tin tặc mạo danh người dùng đó bằng cách chiếm quyền điều khiển một session đang diễn ra.
* Đánh cắp danh tính. Nếu người dùng nhập thông tin bí mật như số thẻ tín dụng vào một trang web bị xâm nhập, những chi tiết này có thể bị đánh cắp bằng JavaScript độc hại.
* Tấn công từ chối dịch vụ và phá hoại trang web
* Đánh cắp dữ liệu nhạy cảm, chẳng hạn như mật khẩu.
* Gian lận tài chính trên các website ngân hàng.

Các cuộc tấn công Reflected XSS ít nguy hiểm hơn các cuộc tấn công Stored XSS, gây ra sự cố dai dẳng khi người dùng truy cập một trang cụ thể, nhưng phổ biến hơn nhiều. Bất kỳ trang nào lấy tham số từ yêu cầu GET hoặc POST và hiển thị lại thông số đó cho người dùng theo một cách nào đó đều có khả năng gặp rủi ro. Một trang không xử lý các tham số chuỗi truy vấn là nội dung không đáng tin cậy có thể cho phép tạo các URL độc hại. Kẻ tấn công sẽ phát tán các URL độc hại này trong email, trong phần bình luận hoặc trên diễn đàn. Vì liên kết trỏ đến một trang web mà người dùng tin cậy, nên họ có nhiều khả năng nhấp vào nó hơn mà không biết tác hại của nó.

Trong các hình thức tấn công XSS phân chia cấp bậc nguy hiểm cao nhất là DOM, sau đó là Stored và Reflected. Vì sao DOM lại là nguy hiểm nhất? Ta được biệt khi kẻ tấn công chèn lệnh độc hại vào và nó được xem là một phần chính của trang web. Về Stored mã độc được lưu và cơ sở dữ liệu của máy chủ, lệnh được khởi động bất cứ khi nào nạn nhân truy cập vào web, cuối cùng là Reflect tấn công khi người dùng nhấp vào đường dẫn có gẵn mã độc.

## XSS có thể làm những gì

### Đánh cắp phiên người dùng trên trang web

Trình duyệt sử dụng một tệp văn bản nhỏ để lưu trữ dữ liệu cục bộ quan trọng về một trang web nhất định. Tệp này chứa cái mà chúng tôi gọi là cookie, các cặp biến và giá trị có một số ý nghĩa đối với ứng dụng đã gửi chúng đến trình duyệt. Cookie được sử dụng để xác định một người sau khi nạn nhân đăng nhập vào một ứng dụng, vì vậy máy chủ không cần phải hỏi lại thông tin đăng nhập mỗi khi người dùng yêu cầu tài nguyên. Mặc dù cookie hợp lệ (chúng hết hạn sau một thời gian nhất định), một phiên người dùng đang hoạt động trong ứng dụng. Nếu những cookie hợp lệ này bị đánh cắp, kẻ trộm có thể mạo danh người dùng đó và tương tác với ứng dụng giống như cách người dùng thực làm mà không cần biết mật khẩu của anh ta.

Điều này cho phép truy cập vào tất cả dữ liệu cá nhân được lưu trữ về người dùng, chẳng hạn như số điện thoại, địa chỉ nhà riêng và thậm chí cả chi tiết thẻ tín dụng của họ trong một trang web thương mại điện tử chẳng hạn. Đối với quản trị viên trang web (quản trị viên), một cuộc tấn công XSS có thể dẫn đến việc tiếp quản trang web của họ và thậm chí cả máy tính nơi nó được lưu trữ.

### Lấy thông tin các phím người dùng nhập

Bằng cách có thể nắm bắt những gì người dùng nhập vào các trường biểu mẫu, như những gì để đăng nhập (tên người dùng và mật khẩu), kẻ tấn công cũng có thể xâm phạm tài khoản người dùng trong một trang web nhất định.

### Làm giả giao diện trang web

Người dùng có thể bị lừa khi nghĩ rằng trang web đã truy cập đã bị tấn công hoặc nó không hoạt động, điều này có thể dẫn đến hoảng sợ hoặc không thể thực hiện các hành động trong ứng dụng như mua một mặt hàng.

### Thu thập thông tin người dùng bằng trang HTML giả

Với khả năng phục vụ bất kỳ nội dung nào, kẻ tấn công có thể thuyết phục người dùng nhập hoặc nhập lại thông tin đăng nhập của mình vào ứng dụng, nhưng lần này là gửi chúng cho kẻ tấn công.

### Gây sự cố trình duyệt (từ chối dịch vụ cục bộ)

Rất hiếm khi sử dụng, nhưng có thể. Chẳng hạn, kẻ tấn công có thể muốn tránh xa một đối thủ nhất định trong một cuộc đấu giá bằng cách làm cho trình duyệt của họ không phản hồi.

### Trực tiếp tải tệp

Thật dễ dàng để làm cho trình duyệt của người dùng tải xuống bất kỳ tệp nào bằng XSS, nhưng không nhất thiết phải thực thi nó, điều này sẽ cấp quyền truy cập vào máy người dùng. Thật không may, do thực tế là kẻ tấn công có quyền kiểm soát một số khía cạnh khác của trang web đáng tin cậy, có vẻ như không quá khó để lừa người dùng mở nó.

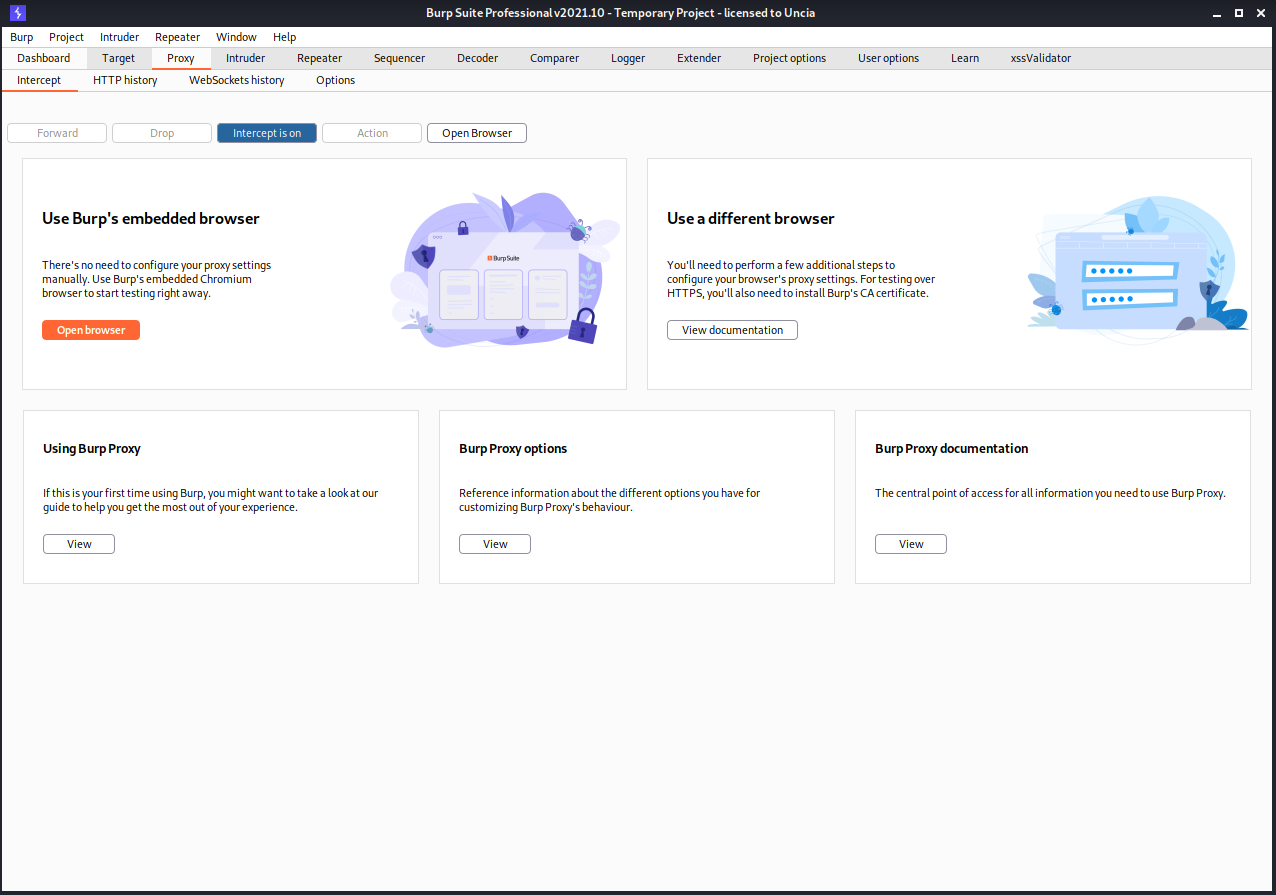
### Chuyển hướng trình duyệt

Một lần nữa, với mục đích chiếm đoạt máy người dùng, kẻ tấn công có thể chuyển hướng trình duyệt một cách vô hình đến một địa chỉ web khác, nơi một ứng dụng được chuẩn bị khác sẽ cố gắng phá vỡ rào cản của trình duyệt để truy cập vào hệ điều hành của người dùng (điều này sẽ dẫn đến thỏa hiệp). Nếu người dùng có trình duyệt lỗi thời hoặc dễ bị tấn công, kẻ tấn công có cơ hội thành công lớn.

## Tìm kiếm và kiểm tra lỗ hổng

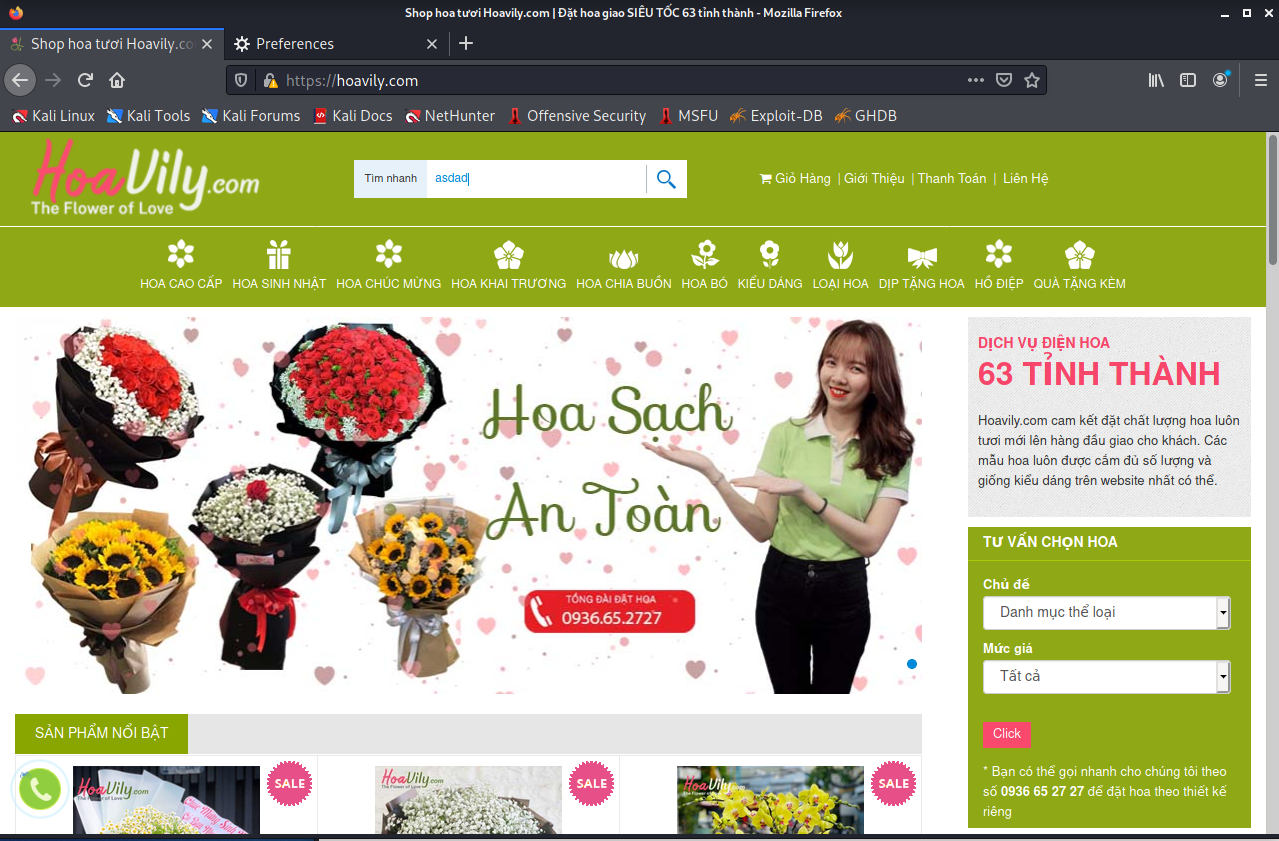
### Sử dụng BrupSuite

Bước 1: Mở Brup Suite Pro 🡪 Proxy 🡪 chuyển Intercept is on qua off



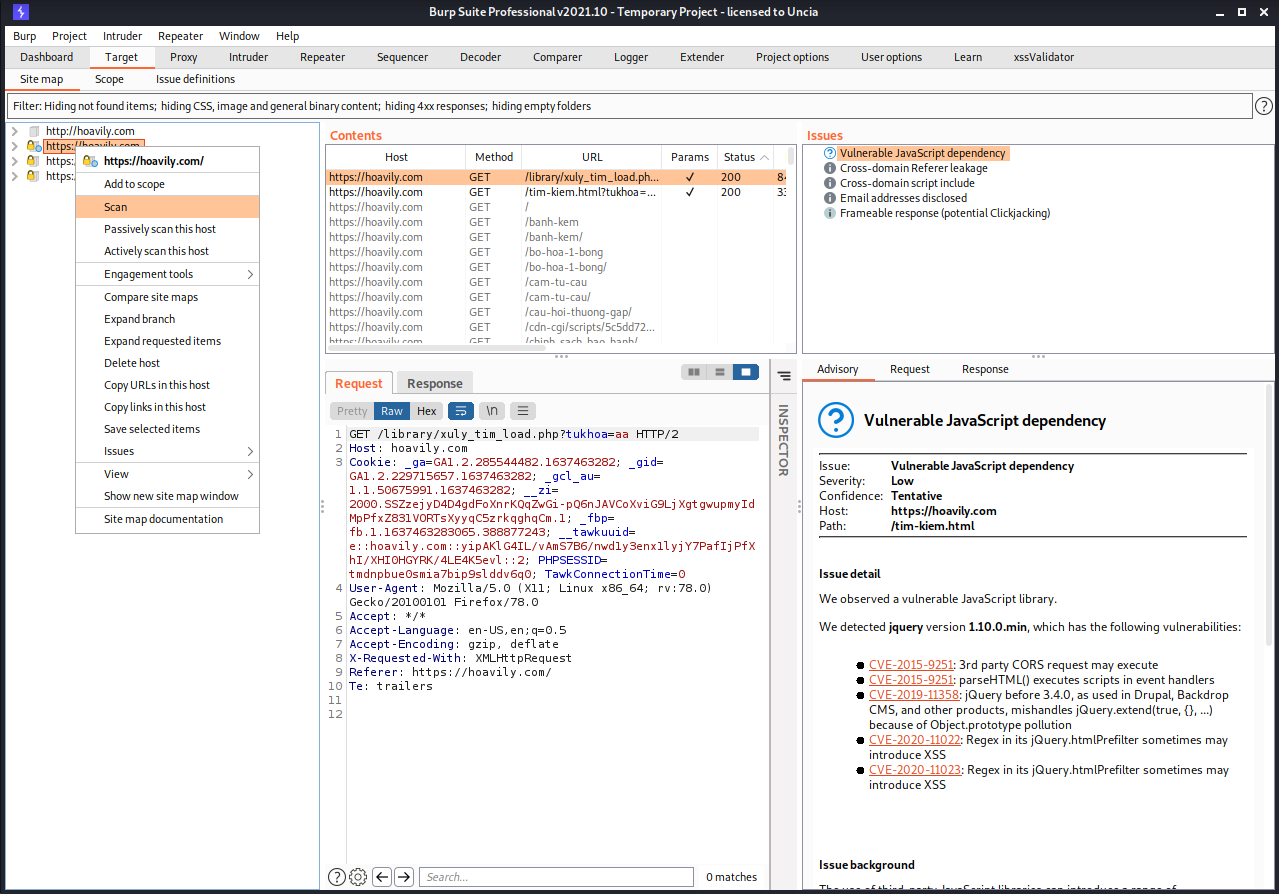
Hình 2.4 Mở Brup Suite Pro

Bước 2: Mở trình duyệt đến trang web cần kiểm tra XSS 🡪 thêm chuỗi bất kì vào các ô dữ liệu đầu vào.



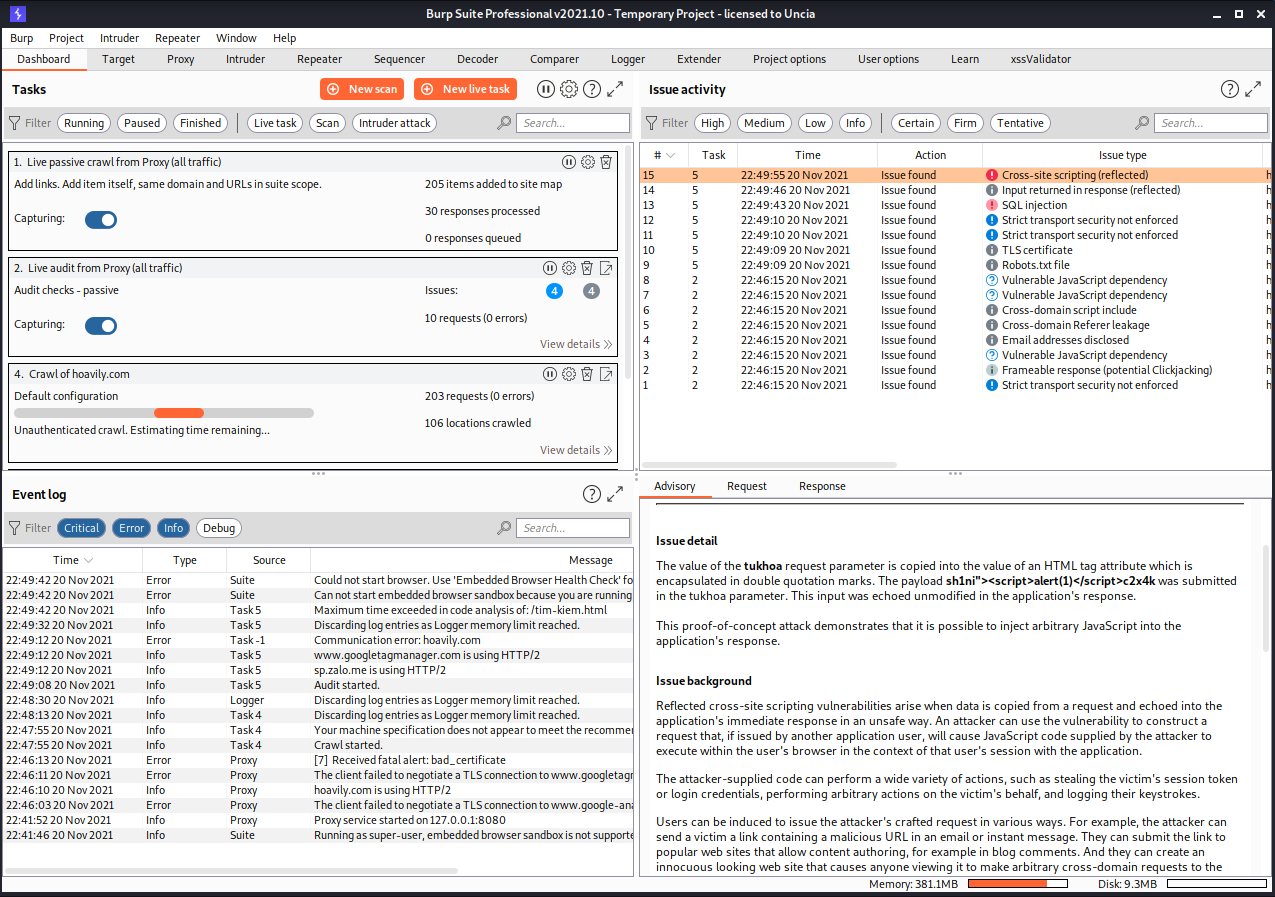
Hình 2.5 Tìm kiếm trên trang web cần kiểm tra XSS

Bước 3: Chọn Target 🡪 Chuột phải vào host cần kiểm tra 🡪 chọn Scan 🡪 OK.



Hình 2.6 Scan host kiểm tra XSS

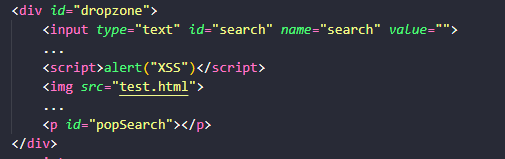
Bước 4: Chú ý bên Issue activity, nếu có lỗi thông tin sẽ hiên lên tại đây.



Hình 2.7 Thông tin lỗi được hiện lên trong Brup Suite

### XSS giữa các thẻ HTML

Trong trường hợp XSS giữa các thẻ HTML, chúng ta đã biết phải đưa ra một thẻ HTML để đoạn javascrip có thể được thực thi.



Hình 2.8 Chèn script giữa các thẻ HTML

Như trường hợp trên, thẻ script: <script>alert("XSS")</script> được chèn vào, và trong quá trình hiển thị trang web trên trình duyệt thì thẻ script này cũng được thực thi.

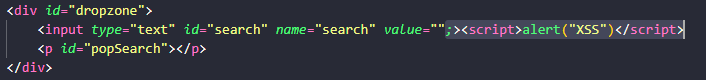
Chúng ta có thể chèn vào đây bất kì thẻ nào, để quá trình trong quá hiển thị trang web thì thẻ HTML đó cũng được thực thi.

Ví dụ một số thẻ phổ biến:

* <a>
* <p>
* <abbr>
* <input>
* <div>
* <b>
* <img>

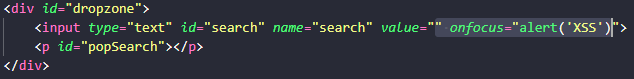
### XSS trong thuộc tính của thẻ HTML

Trong quá trình kiểm tra một điểm đầu vào, chúng ta thường xuyên dùng cách đóng thuộc tính của thẻ hiện tại và chèn một thẻ khác vào, như hình minh họa dưới đây:



Hình 2.9 Chèn XSS bằng cách đóng thẻ HTML

Vậy giả sử trong trường hợp này, các dấu ngoặc nhọn được mã hóa hoặc được lọc, thì đoạn scrip sẽ không được thoát ra khỏi thẻ hiện tại, nên nó sẽ không được thực thi. Với điều kiện có thể đóng thuộc tính hiện tại và thêm một thuộc tính mới ngay trong thẻ HTML đầu vào, ta có thể sử dụng cách tương tự như hình minh hạo dưới đây:



Hình 2.10 Chèn XSS bằng thuộc tính trong thẻ HTML

Với tải trọng là: “ onfocus=”alert(‘XSS’) thì sự kiện onfocus sẽ được thực thi mà không cần đến sự tác động nào từ nạn nhân.

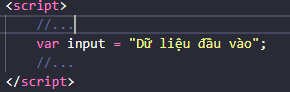
* “ dùng để đóng lại thuộc tính value
* onfocus=”alert(‘XSS’) thêm một thuộc tính độc hại vào cùng với dấu “ từ thuộc tính value thì thuộc tính onfocus sẽ trở nên hoàn thiện onfocus=”alert(‘XSS’)”.

Một số thuộc tính cơ bản có trong tất cả các thẻ:

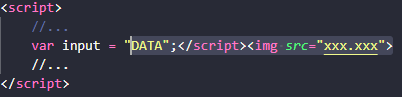
* onfocus
* onclick
* onchange
* oninput
* onkeydown…

### XSS trong javascript

Trong trường hợp này dữ liệu đầu vào được xử lý trong thẻ script, việc đơn giản chúng ta nghĩ đến là thoát ra khỏi thẻ script đó và chèn vào thẻ khác mà chúng muốn với tải trọng như: DATA“; </script><img src=xxx.xxx>.



Hình 2.11 Đoạn javascript dữ liệu đầu vào



Hình 2.12 Chèn XSS trong javascript

Ở đây, thẻ script được đóng lại bằng tải trọng: “</scrip> và thẻ hình đã được thêm vào ngay sau đó. Lý do điêu này hoạt động vì trình duyệt thực hiện phân tích cú pháp HTML để các phần tử trang web và sau đó các đoạn mã script sẽ được thực thi. Tải trọng được đưa lên sẽ phá hỏng lệnh gốc với một </script> bị dư thừa, nhưng nó sẽ không ảnh hưởng đến các câu lệnh tiếp theo được thực thi.

### Vậy XSS thuộc giai đoạn nào trong quá trình pentestweb

Quy trình pentestweb:

1. Information Gathering
2. Configuration and Deployment Mangement Testing
3. Identify Management Testing
4. Authentication Testing
5. Authorization Testing
6. Sesstion Management Testing
7. Input Validation Testing
8. Testing for Error Handing
9. Testing for weak Cryptography
10. Bussiness Logic Testing
11. Client Side Testing

Theo như một quy trình pentest chuẩn thì XSS sẽ nằm ở bước bảy của quy trình, đây là bước kiểm tra những dữ liệu sẽ nhập vào một trang web

## Quy trình tấn cuộc tấn công XSS

**Bước 1:** Mở website cần kiểm tra

**Bước 2:** Xác định các chỗ (phần) cần kiểm tra XSS. 1 Site bất kỳ bao giờ cũng có các phần: Search, error message, web form. Chủ yếu lỗi XSS nằm ở phần này, nói chung XSS có thể xảy ra ở chỗ nào mà người dùng có thể nhập dữ liệu vào và sau đó nhận được một cái gì đó. Ví dụ chúng ta nhập vào chuỗi „XSS‟

**Bước 3:** Xác minh khả năng site có bị lỗi XSS hay không bằng cách xem các thông tin trả về. Ví dụ chúng ta thấy thế này: „Không tìm thấy XSS…‟ , hay là „Tài khoản XSS không chính xác‟, „Đăng nhập với XSS không thành công‟… thì khi đó khả năng chỗ đó bị dính XSS là rất cao.

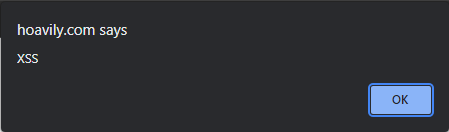
**Bước 4:** Khi đã xác định chỗ có khả năng bị dính lỗi XSS thì chúng ta sẽ chèn những đoạn code của chúng ta vào để thử tiếp, ví dụ như sau:

Chèn đoạn code này: < script>alert('XSS')< /script> vào ô bị lỗi và nhấn nút Submit, nếu chúng ta nhận được một popup có chữ „XSS‟ thì 100% bị dính XSS.

Ta có thể nhập vào form lỗi các thẻ sau:

* <script>alert('CSS Vulnerable')</script>
* <i\*g csstest=javascript:alert('CSS Vulnerable')>
* &{alert('CSS Vulnerable') };
* <script>window.open( “https://www.google.com/” )</script>
* <META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">

Ngoài những câu lệnh trên, <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> là trang web chữa những tải trọng có thể tham khảo để thực hiện một màn khởi đầu cho cuộc tấn công XSS.



Hình 2.13 Thông báo XSS hiện lên

Hiện nay hầu như các web đã sử dụng các bộ lọc chặn các ký tự như: > < “ ‘; nếu muốn bypass qua thì phải view source và dùng các phương pháp để tránh các bộ lọc của webserver.

**Bước 5:** Lên kế hoạch kịch bản tấn công.

Từ những lỗ hổng XSS sẽ là tiền để để dẫn tới 1 cuộc tấn công CSRF.

## Biện pháp phòng chống tấn công XSS

### Reflected và Stored XSS

* + - Không bao giờ chèn dữ liệu không đáng tin cậy

Quy tắc đầu tiên là từ chối tất cả - không đưa dữ liệu không đáng tin cậy vào tài liệu HTML của bạn trừ khi nó nằm trong một trong các vị trí được xác định trong Quy tắc số 1 đến Quy tắc số 5. Lý do cho Quy tắc đầu là có rất nhiều ngữ cảnh lạ trong HTML khiến danh sách các quy tắc mã hóa trở nên rất phức tạp. Chúng tôi không thể nghĩ ra bất kỳ lý do chính đáng nào để đưa dữ liệu không đáng tin cậy vào những bối cảnh này. Điều này bao gồm "ngữ cảnh lồng nhau" như URL bên trong JavaScript - quy tắc mã hóa cho những vị trí đó rất phức tạp và nguy hiểm.

Trong script:

<script>...NEVER PUT UNTRUSTED DATA HERE...</script>

Trong HTML comment:

<!--...NEVER PUT UNTRUSTED DATA HERE...-->

Trong attribute name:

<div ...NEVER PUT UNTRUSTED DATA HERE...=test />

Trong tag name:

<NEVER PUT UNTRUSTED DATA HERE... href="/test" />

Trong CSS:

<style>

...NEVER PUT UNTRUSTED DATA HERE...

</style>

* + - Mã hóa HTML trước khi chèn dữ liệu không đáng tin cậy vào nội dung phần tử HTML

Quy tắc số 2 dành cho khi bạn muốn đưa dữ liệu không đáng tin cậy trực tiếp vào phần thân HTML ở đâu đó. Điều này bao gồm bên trong các thẻ thông thường như div, p, b, td, v.v. Hầu hết các khuôn khổ web đều có phương pháp mã hóa / thoát HTML cho các ký tự được nêu chi tiết bên dưới. Tuy nhiên, điều này hoàn toàn không đủ đối với các ngữ cảnh HTML khác. Bạn cũng cần phải thực hiện các quy tắc khác được nêu chi tiết tại đây.

<body>

...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...

</body>

<div>

...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...

</div>

Mã hóa các ký tự sau bằng mã hóa thực thể HTML để ngăn chuyển sang bất kỳ ngữ cảnh thực thi nào, chẳng hạn như trình xử lý tập lệnh, kiểu hoặc sự kiện. Sử dụng các thực thể hex được khuyến khích trong thông số kỹ thuật. 5 ký tự quan trọng trong XML (&, <,>, ", '):

& --> &amp;

< --> &lt;

> --> &gt;

" --> &quot;

' --> &#x27;

* + - Mã hóa thuộc tính trước khi chèn dữ liệu không đáng tin cậy vào các thuộc tính chung HTML

Quy tắc số 3 là để đưa dữ liệu không đáng tin cậy vào các giá trị thuộc tính điển hình như width, name, value, vv… Điều này không được sử dụng cho các thuộc tính phức tạp như href, src, style hoặc bất kỳ trình xử lý sự kiện nào như onmouseover. Điều cực kỳ quan trọng là các thuộc tính của trình xử lý sự kiện phải tuân theo Quy tắc số 3 cho Giá trị dữ liệu JavaScript HTML.

Trong thuộc tính UNquoted:

<div attr=...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...>content

Trong đơn thuộc tính quoted:

<div attr='...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...'>content

Inside double quoted attribute :

<div attr="...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...">content

* + - Mã hóa javascript trước khi thêm vào không đáng tin cậy vào database

Quy tắc số 4 liên quan đến mã JavaScript được tạo động - cả khối tập lệnh và thuộc tính xử lý sự kiện. Nơi an toàn duy nhất để đưa dữ liệu không đáng tin cậy vào mã này là bên trong "giá trị dữ liệu" được trích dẫn. Việc bao gồm dữ liệu không đáng tin cậy bên trong bất kỳ ngữ cảnh JavaScript nào khác là khá nguy hiểm, vì rất dễ chuyển sang ngữ cảnh thực thi với các ký tự bao gồm (nhưng không giới hạn ở) dấu chấm phẩy, dấu bằng, dấu cách, dấu cộng và nhiều hơn nữa, vì vậy hãy thận trọng khi sử dụng .

Inside a quoted string:

<script>alert('...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...')</script>

One side of a quoted expression:

<script>x='...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...'</script>

Inside quoted event handler:

<div onmouseover="x='...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...'"</div>

* + - Mã hóa CSS và xác thực nghiêm ngặt trước khi chèn dữ liệu không đáng tin cậy vào các giá trị thuộc tính kiểu HTML

Quy tắc số 5 dành cho khi bạn muốn đặt dữ liệu không đáng tin cậy vào biểu định kiểu hoặc thẻ kiểu. CSS mạnh mẽ một cách đáng ngạc nhiên và có thể được sử dụng cho nhiều cuộc tấn công. Do đó, điều quan trọng là bạn chỉ sử dụng dữ liệu không đáng tin cậy trong một giá trị thuộc tính và không sử dụng những nơi khác trong dữ liệu kiểu. Bạn nên tránh xa việc đưa dữ liệu không đáng tin cậy vào các thuộc tính phức tạp như url, hành vi và tùy chỉnh (-moz-binding). Bạn cũng không nên đặt dữ liệu không đáng tin cậy vào giá trị thuộc tính biểu thức của IE cho phép JavaScript.

* + - Mã hóa URL trước khi chèn dữ liệu không đáng tin cậy vào giá trị tham số URL HTML Quy tắc số 6 dành cho khi bạn muốn đưa dữ liệu không đáng tin cậy vào giá trị tham số HTTP GET.

<a href=”[http://www.somesite.com?test=...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...](http://www.somesite.com?test=...ENCODE%20UNTRUSTED%20DATA%20BEFORE%20PUTTING%20HERE...)”>link</a>

Ngoại trừ các ký tự chữ và số, hãy mã hóa tất cả các ký tự có giá trị ASCII nhỏ hơn 256 bằng định dạng mã hóa% HH. Bao gồm dữ liệu không đáng tin cậy trong dữ liệu: Không được phép sử dụng URL vì không có cách nào tốt để vô hiệu hóa các cuộc tấn công bằng mã hóa / thoát để ngăn chặn việc chuyển ra khỏi URL. Tất cả các thuộc tính phải được trích dẫn. Các thuộc tính chưa được trích dẫn có thể được chia nhỏ với nhiều ký tự bao gồm [dấu cách]% \* +, - /; <=> ^ |. Lưu ý rằng mã hóa thực thể là vô dụng trong ngữ cảnh này. CẢNH BÁO: Không mã hóa các URL hoàn chỉnh hoặc tương đối bằng mã hóa URL! Nếu đầu vào không đáng tin cậy được đặt vào href, src hoặc các thuộc tính dựa trên URL khác, thì nó nên được xác thực để đảm bảo rằng nó không trỏ đến một giao thức không mong muốn, đặc biệt là các liên kết javascript. Sau đó, URL phải được mã hóa dựa trên ngữ cảnh hiển thị giống như bất kỳ phần dữ liệu nào khác. Ví dụ: URL do người dùng điều khiển trong liên kết HREF phải được mã hóa thuộc tính.

* + - Vệ sinh đánh dấu HTML bằng Thư viện được thiết kế cho công việc

Nếu ứng dụng của bạn xử lý đánh dấu - đầu vào không đáng tin cậy được cho là chứa HTML - thì có thể rất khó xác thực. Việc mã hóa cũng khó khăn, vì nó sẽ phá vỡ tất cả các thẻ được cho là có trong đầu vào. Do đó, bạn cần một thư viện có thể phân tích cú pháp và làm sạch văn bản có định dạng HTML. Ví dụ [HtmlSanitizer](https://github.com/mganss/HtmlSanitizer)**,** [OWASP Java HTML Sanitizer](https://owasp.org/www-project-java-html-sanitizer/) **hay** [Ruby on Rails SanitizeHelper](http://api.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html)**.**

* + - Tránh các URL javascript

Các URL không đáng tin cậy bao gồm giao thức javascript: sẽ thực thi mã JavaScript khi được sử dụng trong các vị trí DOM URL, chẳng hạn như các thuộc tính HREF của thẻ neo hoặc các vị trí iFrame src. Đảm bảo xác thực tất cả các URL không đáng tin cậy để đảm bảo chúng chỉ chứa các lược đồ an toàn như HTTPS.

* + - Sử dụng HTTPOnly cookie flag

Như bạn có thể thấy, rất khó để ngăn chặn tất cả các lỗi XSS trong một ứng dụng. Để giúp giảm thiểu tác động của lỗ hổng XSS trên trang web của bạn, OWASP cũng khuyên bạn nên đặt cờ HTTPOnly trên cookie phiên của bạn và bất kỳ cookie tùy chỉnh nào bạn có mà không được truy cập bởi bất kỳ JavaScript nào bạn đã viết. Cờ cookie này thường được bật theo mặc định trong ứng dụng .NET, nhưng ở các ngôn ngữ khác, bạn phải đặt nó theo cách thủ công.

* + - Triển khai chính sách bảo mật nội dung(Content Security Policy)

Có một giải pháp phức tạp tốt khác để giảm thiểu tác động của lỗ hổng XSS được gọi là Chính sách bảo mật nội dung. Đó là cơ chế phía trình duyệt cho phép bạn tạo danh sách trắng nguồn cho các tài nguyên phía máy khách của ứng dụng web của bạn, ví dụ: JavaScript, CSS, hình ảnh, v.v. CSP thông qua tiêu đề HTTP đặc biệt hướng dẫn trình duyệt chỉ thực thi hoặc hiển thị tài nguyên từ các nguồn đó.

* + - Sử dụng JS Frameworks hiện đại

Các frameworks JavaScript hiện đại được tích hợp sẵn tính năng bảo vệ XSS khá tốt. Thông thường, API frameworks cho phép bỏ qua bảo vệ đó để hiển thị HTML không thoát hoặc bao gồm mã thực thi. Các phương pháp và đạo cụ API trong bảng dưới đây được coi là nguy hiểm và bằng cách sử dụng chúng, bạn có thể khiến người dùng của mình gặp phải lỗ hổng XSS. Nếu bạn thực sự phải sử dụng chúng, hãy nhớ rằng bây giờ tất cả dữ liệu phải được làm sạch bởi chính bạn.

### DOM XSS

* HTML Escape rồi đến JavaScript Escape trước khi chèn dữ liệu không đáng tin cậy vào văn bản phụ HTML.
* JavaScript Escape trước khi chèn dữ liệu không đáng tin cậy vào văn bản phụ thuộc tính HTML.
* JavaScript Escape trước khi chèn dữ liệu không đáng tin cậy vào văn bản phụ thuộc tính CSS.
* URL Escape rồi đến JavaScript Escape trước khi chèn dữ liệu không đáng tin cậy vào văn bản phụ thuộc tính URL.

# KẾT QUẢ THỰC NGHIỆM

## Thực hiện tấn công XSS

Chiến lược:

* Tìm web dính XSS
* Tạo đường dẫn khai thác đến web tấn công XSS
* Gửi đường dẫn khai thác
* Nhận thông tin từ nạn nhân truy cập (cookie, tài khoản mật khẩu,..).
* Dùng thông tin nạn nhân truy cập trang web dính XSS

## Thực hiện ngăn chặn XSS

Dùng các hàm chuyển đổi trong javascrip, php, c#

Dùng các famwork có sẵn trong HTML

# KẾT LUẬN

Không giống như các vấn đề liên quan đến bảo mật khác, XSS không thể loại bỏ trong một thời gian ngắn. Tuy nhiên, một tấn công XSS chỉ thực hiện được khi gửi một trang web cho trình duyệt web của nạn nhân có kèm theo mã script độc của kẻ tấn công. Thông qua đề tài này, chúng ta biết được cách hacker dùng một số công cụ và kĩ thuật để có thể tấn công XSS và việc tấn công diễn ra thế nào. Đồng thời, chúng ta có thể biết được những hậu quả khi bị tấn công XSS.

Vì thế, ngoài việc ứng dụng kiểm tra tính đúng đắn của dữ liệu trước khi sử dụng thì việc cần nhất là người dùng nên cảnh giác trước khi bước vào một trang Web mới hay khi nhận được một email rất thu hút nào đó. Có thể nói, nhờ vào sự cảnh giác của người dùng thì 90% đã đạt được sự bảo mật trong kĩ thuật này.

# TÀI LIỆU THAM KHẢO

[1]. [https://brutelogic.com.br/blog/](%20https:/brutelogic.com.br/blog/)

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

[3]. [https://cunghocweb123.blogspot.com/2013/07/tan-cong-XSS-va-cach-phong-chong-bao-mat.html](https://cunghocweb123.blogspot.com/2013/07/tan-cong-xss-va-cach-phong-chong-bao-mat.html)

[4]. [https://viblo.asia/p/tan-cong-XSS-va-cach-phong-chong-L4x5x09O5BM](https://viblo.asia/p/tan-cong-xss-va-cach-phong-chong-L4x5x09O5BM)

[5]. [XSS game (XSS-game.appspot.com)](http://xss-game.appspot.com/)

[6]. [https://securitydaily.net/cac-kieu-khai-thac-XSS-phan-3-DOM-based-XSS/](https://securitydaily.net/cac-kieu-khai-thac-xss-phan-3-dom-based-xss/)