Tìm hiểu về lỗ hổng CVE-2021-26084: Nguyên nhân, cách khai thác, cách thức phát hiện.

# Phạm vi ảnh hưởng

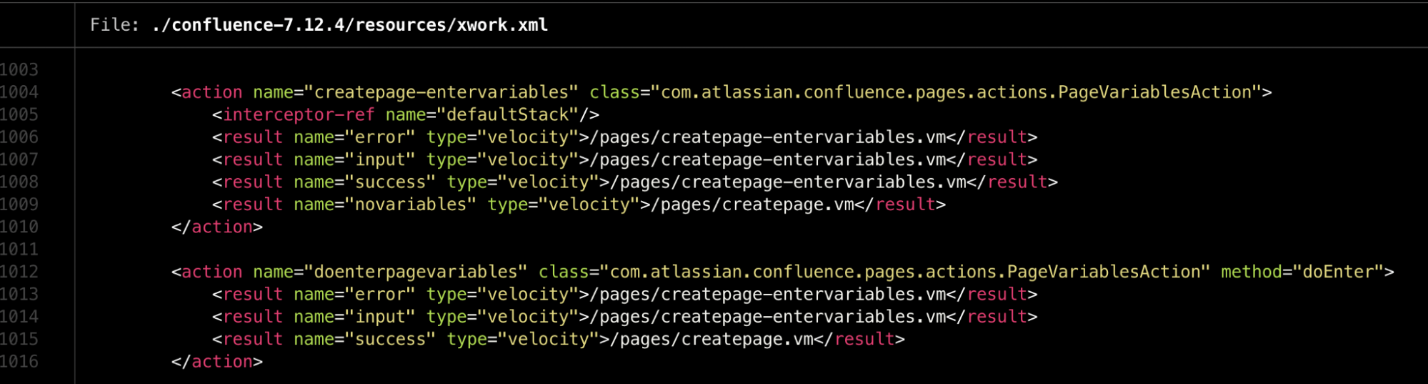
Phiên bản < 6.13.23

6.14.0 ≤ Phiên bản < 7.4.11

7.5.0 ≤ Phiên bản < 7.11.6

7.12.0 ≤ Phiên bản < 7.12.5

# Nguyên nhân, Cách khai thác

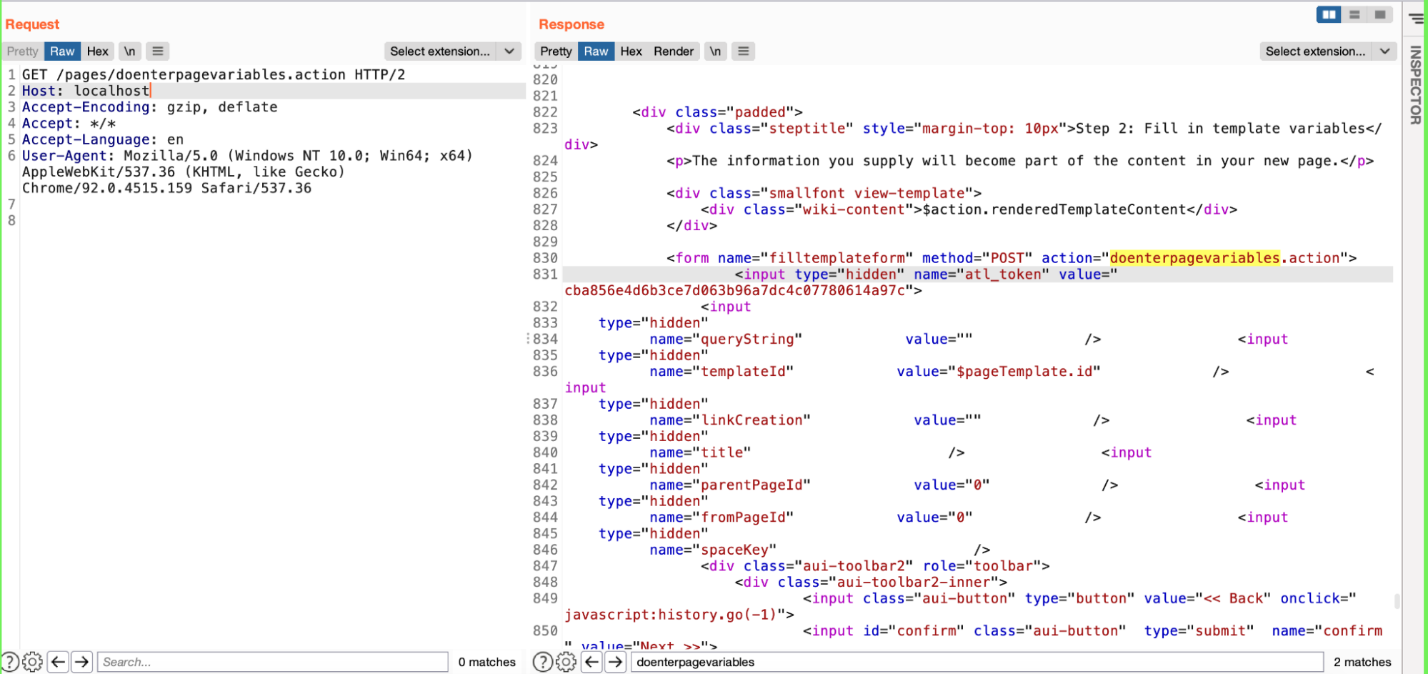
Confluence sử dụng framework để map URL tới các lớp Java, tạo ra cái được gọi là “action”. Action URLs kết thúc bằng “.action” và được xác định trong tệp xwork.xml ở confluence-<version>.jar và trong tệp atlassian-plugin.xml trong các tệp JAR của các plugin đi kèm. Mỗi action chứa ít nhất một thuộc tính name, xác định tên hành động, một thuộc tính class, xác định class Java thực hiện action và ít nhất một phần tử result quyết định Velocity template sẽ hiển thị sau khi action được gọi dựa trên result của action. Các giá trị phổ biến được trả về từ các action là “error”, “input”, “success”, nhưng bất kỳ kết quả nào cũng có thể được sử dụng nếu nó phù hợp với phần tử result trong XWork XML. Các action có thể chứa thuộc tính phương thức cho phép gọi một phương thức cụ thể của class Java đã chỉ định. Khi không có lệnh nào được chỉ định, phương thức doDefault() sẽ được gọi. Dưới đây là mục action cho createpage-entervariables action:  


Phương thức doEnter() của class com.atlassian.confluence.pages.actions.PageVariablesAction xử lý các yêu cầu tới doenterpagevariables.action và trả về các giá trị “error”, “input”, “success”. Điều này dẫn tới Velocity template sẽ được hiển thị.

Ở đây, giá trị thuộc tính “name” của một phần tử action tương ứng với một đường dẫn /<nameValue>.action và phần tử chứa template nào sẽ được hiển thị như một phần của phản hồi dựa trên error/success,... Vì vậy theo ví dụ này chỉ cần truy cập /pages/doenterpagevariables.action sẽ hiển thị tệp velocity template.



Chúng ta có thể thấy velocity template được hiển thị như thế nào trong trang HTML



Nhập tên thẻ trong template làm tham số và nhận thấy rằng các giá trị được lấy từ tham số request và được trả lại trong response.

A screen shot of a computer code

Description automatically generated

Cách sử dụng của #tag : #tag (“attribute1”, “attribute2”, “attribute3”)

Các attribute này trong quá trình render sẽ được lấy tại AbstrctTagDirective.applyAttributes()

A screenshot of a computer

Description automatically generated

#tag ("Hidden" "name='queryString'" "value='ahihihi'")

Sau khi các attribyte được lấy ra bởi phương thức AbstrctTagDirective.applyAttributes(), nó được truyền vào AbstrctUITag.doEndTag() 🡪 AbstractUITag.evaluateParams()

A screen shot of a computer

Description automatically generated

Sau đó tiếp tục được truyền vào WebWorkTagSupport.findValue() 🡪 OgnlValueFinder.findValue() với dạng expression

A computer screen shot of a computer screen

Description automatically generated

Sau đó truyền vào SafeExpressionUtil.isSafeExpression()

A screen shot of a computer

Description automatically generated

Tại đây expression đã được truyền vào compile, kiểm tra blacklist và đẩy vào cache. Qua bước compile sẽ được compile sang dạng ASTConst khi được đánh giá thì sẽ return về chuỗi đã được compile từ trước.

A screenshot of a computer program

Description automatically generated

Khi expression được thêm dấu “’” từ input thì đã bị escape sang dạng html entity

A screen shot of a computer

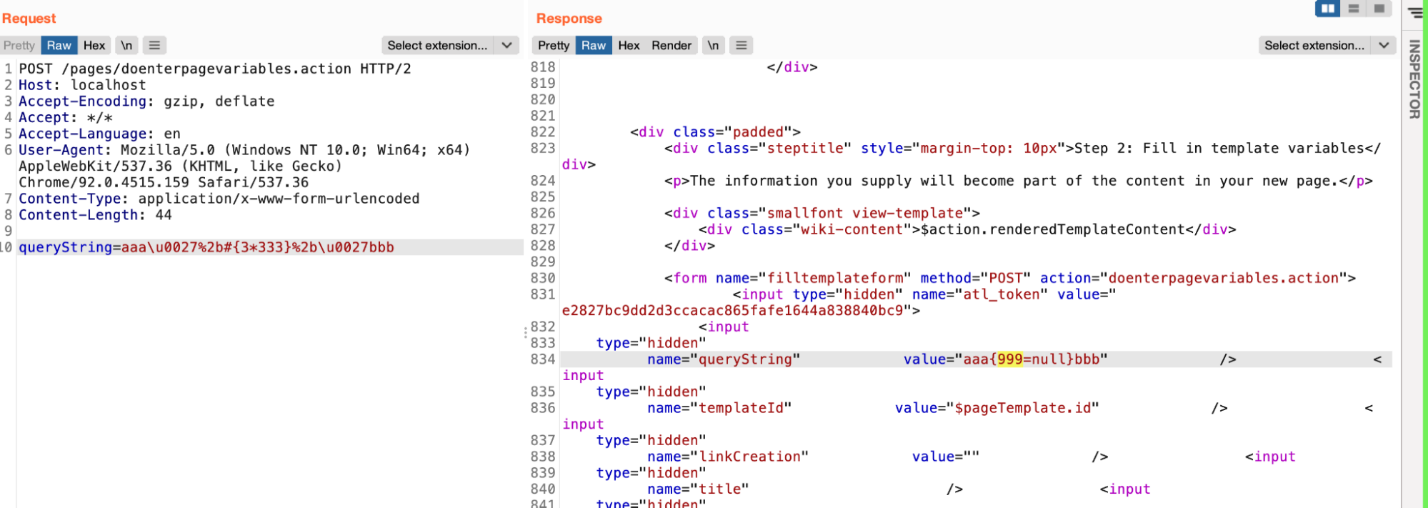
Description automatically generated

Điểm escape là tại HtmlAnnotationEscaper.annotatedValueInsert()

A screenshot of a computer

Description automatically generated

Tại đây phương thức ognl.JavaCharStream.readChar() được gọi và đánh giá các ký tự Unicode escape, do đó khi truyền “\u0027” vào thì nó sẽ convert sang “’”. Do đó có thể escape và nối thêm OGNL expression

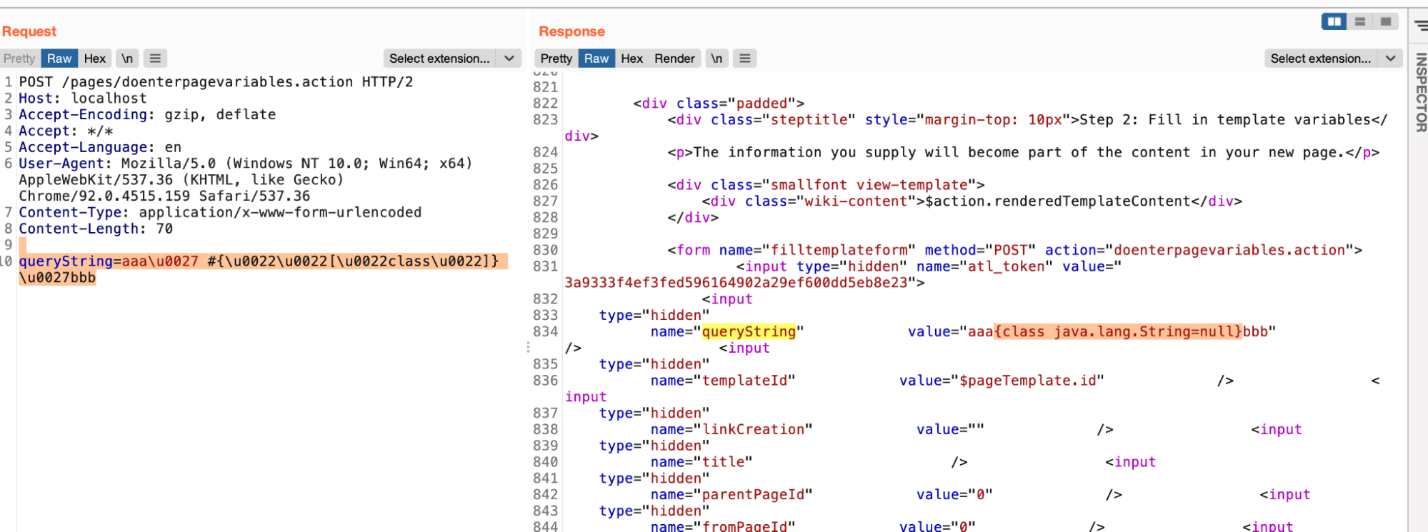


Nhưng phải vượt qua được blacklist của OGNL như sau



Có thể vượt qua bằng cách sử dụng Array accessors thay vì sử dụng phương thức “  
getClass” hoặc thuộc tính “.class”.

queryString=aaa\u0027%2b#{\u0022\u0022[\u0022class\u0022]}%2b\u0027bbb



# Cách thức phát hiện

Giám sát tất cả các yêu cầu lưu lượng HTTP trong đó thành phần đường dẫn của URI yêu cầu chứa một trong các chuỗi trong cột " URI Path" của bảng sau:

|  |  |
| --- | --- |
| URI Path | Vulnerable Parameters |
| /users/darkfeatures.action | featureKey |
| /users/enabledarkfeature.action | featureKey |
| /users/disabledarkfeature.action | featureKey |
| /login.action | token |
| /dologin.action | token |
| /signup.action | token |
| /dosignup.action | token |
| /pages/createpage-entervariables.action | queryString, linkCreation |
| /pages/doenterpagevariables.action | queryString |
| /pages/createpage.action | queryString |
| /pages/createpage-choosetemplate.action | queryString |
| /pages/docreatepagefromtemplate.action | queryString |
| /pages/docreatepage.action | queryString |
| /pages/createblogpost.action | queryString |
| /pages/docreateblogpost.action | queryString |
| /pages/copypage.action | queryString |
| /pages/docopypage.action | queryString |
| /plugins/editor-loader/editor.action | syncRev |

Nếu tìm thấy yêu cầu như vậy thì nên kiểm tra phương thức HTTP request. Nếu phương thức yêu cầu là POST, thì tìm Vulnerable Parameters tương ứng từ bảng trên trong phần body của HTTP request và nếu phương thức yêu cầu là GET, thì tìm các parameter trong request-URI của HTTP request. Kiểm tra xem giá trị của bất kỳ Vulnerable Parameters nào có chứa chuỗi “\u0027” hoặc dạng mã hóa URL của chuỗi đó hay không. Nếu vậy, lưu lượng truy cập sẽ được coi là độc hại và một cuộc tấn công khai thác lỗ hổng này có thể đang diễn ra.