**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**Trường Công nghệ thông tin và Truyền thông**

──────── \* ────────

Ảnh có chứa văn bản, ký hiệu

Mô tả được tạo tự động

**BÁO CÁO PROJECT III**

**Đề tài: Tìm hiểu về lỗ hổng Insecure Deserialization và ảnh hưởng trong thực tế**

Học phần: Project III

Họ và tên: Bùi Trung Quân

MSSV: 20194142

Giảng viên hướng dẫn: Bùi Quốc Trung

Học kỳ: 2022-2

Mục lục

[**I.** **Giới thiệu** 2](#_Toc135947703)

[**1.** **Đặt vấn đề** 2](#_Toc135947704)

[**2.** **Mục tiêu nghiên cứu** 3](#_Toc135947705)

[**II.** **Insecure Deserialization: Khái niệm và cơ chế hoạt động** 4](#_Toc135947706)

[**1.** **Các định nghĩa** 4](#_Toc135947707)

[Serialization 4](#_Toc135947708)

[Deserialization 4](#_Toc135947709)

[Insecure Deserialization 5](#_Toc135947710)

[Magic Method 6](#_Toc135947711)

[Gadget chain, Sink và Source 6](#_Toc135947712)

[Formatter 7](#_Toc135947713)

[**2.** **Cơ chế hoạt động** 7](#_Toc135947714)

[Mô tả cơ chế hoạt động trong ngôn ngữ PHP 7](#_Toc135947715)

[Phân tích hoạt động của các gadget chain phổ biến trong CSharp 12](#_Toc135947716)

[**III.** **Insecure Deserialization trong thực tế** 18](#_Toc135947717)

[**1.** **Lỗ hổng CVE-2022-29108** 18](#_Toc135947718)

[**2.** **Lỗ hổng CVE-2023-21744** 19](#_Toc135947719)

[**IV.** **Các biện pháp phòng ngừa và đảm bảo an toàn dữ liệu** 22](#_Toc135947720)

[**1.** **Nguyên tắc thiết kế an toàn cho quá trình giải mã** 22](#_Toc135947721)

[**2.** **Phương phác xác thực dữ liệu đầu vào** 22](#_Toc135947722)

[**V.** **Kết luận** 23](#_Toc135947723)

[**1.** **Tổng kết nội dung** 23](#_Toc135947724)

[**2.** **Hướng phát triển và nghiên cứu tiếp theo** 23](#_Toc135947725)

[**Tài liệu tham khảo** 24](#_Toc135947726)

# **Giới thiệu**

## **Đặt vấn đề**

Trong thời đại 4.0 hiện nay, công nghệ thông tin được tích hợp và áp dụng rộng rãi trong nhiều lĩnh vực đang trở thành một xu hướng tất yếu. Tuy nhiên, sự phát triển này cũng đồng nghĩa với việc xuất hiện rất nhiều nguy cơ về an ninh thông tin và bảo mật dữ liệu. Do đó, việc tìm hiểu các lỗ hổng bảo mật phổ biến và những ảnh hưởng của chúng trong thực tế là vô cùng quan trọng.

## **Mục tiêu nghiên cứu**

Trong báo cáo này, tôi tập trung vào một lỗ hổng đáng chú ý, đó là *“Insecure Deserialization”* (Giải mã cấu trúc không an toàn). Thời gian gần đây, nó đã trở thành một trong những điểm yếu phổ biến và nguy hiểm trong các ứng dụng web hoặc phần mềm khác. Trên thực tế, lỗ hổng này đã được tận dụng trong nhiều cuộc tấn công mạng và để lại những hậu quả đáng kể. Các cuộc tấn công này cho phép thực thi mã độc, thay đổi dữ liệu, hoặc xâm nhập vào các hệ thống mục tiêu. Qua đó, việc hiểu rõ cơ chế hoạt động cũng như tác động của nó trong thực tế sẽ giúp chúng ta nâng cao khả năng bảo vệ thông tin và hạn chế được các cuộc tấn công.

# **Insecure Deserialization: Khái niệm và cơ chế hoạt động**

## **Các định nghĩa**

### Serialization

Serialization là quá trình chuyển đổi dữ liệu từ RAM (Random Access Memory) thành định dạng nhị phân để phục vụ cho việc lưu trữ, có thể lưu trong ổ cứng hoặc là các hệ quản trị cơ sở dữ liệu, …. Đối với mỗi ngôn ngữ lập trình, cách chuyển đổi này sẽ khác nhau và có nhiều định dạng khác nhau, có thể là dạng nhị phân hoặc JSON, XML, …

A picture containing text, screenshot, display, software

Description automatically generated

Figure 1: Serialization trong ngôn ngữ PHP

Trên đây, với biến $string1 khi serialize sẽ được kết quả s:13:”Hello, World!”;. Trong đó ký tự s là đánh dấu cho kiểu dữ liệu string, còn 13 là độ dài của string đó.

### Deserialization

Ngược lại với serialization, đây là quá trình chuyển đổi dữ liệu từ các bộ phận lưu trữ vào RAM để phục vụ cho hoạt động của chương trình.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 2: Deserialization trong PHP

Trong trường hợp này, ký tự O đánh dấu một object, “Demo” là tên class của object đó với 4 ký tự và object này có 1 thuộc tính data có kiểu dữ liệu là string với giá trị là “123”.

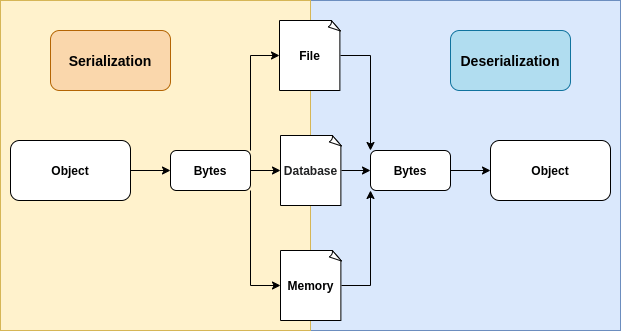


Figure 3: Minh họa các quá trình

### Insecure Deserialization

Insecure Deserialization là khái niệm ám chỉ quá trình Deserialization mà không có sự kiểm soát đầu vào dữ liệu. Điều này dẫn tới việc chúng ta có thể tùy chỉnh đầu vào để tạo ra những object phục vụ cho mục đích tấn công hệ thông.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 4: Insecure Deserialization trong PHP

Ở đây ta có thể thấy class User có thuộc tính mặc định permission = “user” ở phạm vi private và không có setter để thay đổi giá trị. Nhưng với ví dụ trên, chúng ta đã tạo ra một đối tượng User với permission là “admin” một cách trái phép.

### Magic Method

Magic Method là những phương thức được định nghĩa trong từng class, cho phép object của class đó phản ứng lại các events nhất định theo từng trường hợp cụ thể. Có thể kể đến các magic method phổ biến như *Constructor, Destructor, ToString, …*

### Gadget chain, Sink và Source

Gadget chain: Khi thực hiện quá trình Serialization hoặc Deserialization, chúng sẽ gọi đến các magic method tương ứng với quá trình đó. Điều này sẽ tạo ra một luồng thực thi đến các class khác và thực hiện công việc mà ta mong muốn như ghi chạy lệnh, đọc hoặc ghi file.

Source là điểm bắt đầu (phương thức khởi đầu) của một quá trình Deserialization.

Sink là phương thức kết thúc của quá trình. Phương thức này sẽ thực hiện công việc ta mong muốn như đã nói phía trên.

Vậy nói cách khác, Gadget chain là một luồng thực thi từ sink đến source.

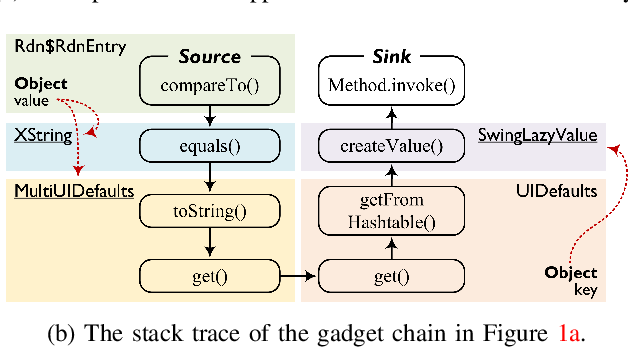


Figure 5: Một Gadget chain trong Java cho phép gọi phương thức Method.invoke (thực thi method bất kỳ)

### Formatter

Formatter là một đối tượng dùng để quyết định dữ liệu như nhị phân, JSON, XML cũng như cách thức chuyển đổi dữ liệu cho các quá trình De(serialization). Ngoài ra trong một số ngôn ngữ lập trình, formatter cũng quy định các điều kiện để đánh giá quá trình giải mã có thể thực thi đúng mục đích hay không.

## **Cơ chế hoạt động**

Đối với từng ngôn ngữ lập trình, quá trình Deserialization sẽ được thực hiện theo cách thức khác nhau nhưng khái quát lại thì cơ chế chung của quá trình này là gọi đến cách magic method được định nghĩa trong class qua đó dựa vào ngữ cảnh để tạo ra luồng thực thi tới các hàm độc hại. Trong tài liệu này, tôi sẽ lấy ví dụ về 2 ngôn ngữ phổ biến là PHP và CSharp.

### Mô tả cơ chế hoạt động trong ngôn ngữ PHP

Chúng ta xét với mã nguồn sau.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 6: file exp.php

Lớp Image chứa các thuộc tính đặc trưng của tệp hình ảnh như filename, path. Lớp còn cung cấp thuộc tính sizeGetter có tác dụng lấy kích thước của tệp hình ảnh.

A screen shot of a computer program

Description automatically generated with medium confidence

Figure 7: file GetImageSize.php

Lớp GetImageSize nhận vào đường dẫn của tệp tin. Phương thức execute trả về kích thước của tệp tin với đường dẫn filepath.

A screen shot of a computer program

Description automatically generated with medium confidence

Figure 8: file GetFileContent.php

Tương tự như GetImageSize, lớp GetFileContent phục vụ chức năng đọc nội dung tệp tin từ đường dẫn filepath. Trong mã nguồn trên, chúng ta có thể thấy rằng lớp GetFileContent không hề có tham chiếu tới từ lớp Image hay GetImageSize.

A screen shot of a computer program

Description automatically generated with low confidence

Ở đây, ta tạo ra một đối tượng Image với thuộc tính path là ./sample.jpg. Tại dòng 33, khi nhận được lệnh echo thì phương thức \_\_toString của đối tượng được thực thi (magic method này được kích hoạt khi có lệnh ép kiểu string); sau đó \_\_toString gọi đến phương thức getSize; getSize gọi đến phương thức execute của thuộc tính sizeGetter. Trước đó, sizeGetter đã được gán với một đối tượng GetImageSize. Do đó, Ta có được luồng thực thi như sau.

Image.\_\_toString()

-> Image.getSize()

-> GetImageSize.execute()

Quá trình này trả về kích thước của tệp tin ./sample.jpg.

Như đã nói phía trên, Insecure Deserialization có thể tạo ra một object bất kỳ phục vụ cho mục đích của kẻ tấn công. Ở đây, nếu gán thuộc tính sizeGetter thành một object của GetFileContent thì sẽ xảy ra điều gì?

Image.\_\_toString()

-> Image.getSize()

-> GetFileContent.execute()

A picture containing text, screenshot, multimedia software, software

Description automatically generated

Với đoạn mã sau, quá trình deserialization sẽ tạo một đối tượng Image với 1 thuộc tính sizeGetter (cách thuộc tính khác sẽ được gán giá trị null). Tiếp tục tạo thêm một đối tượng GetFileContent với thuộc tính filepath có giá trị /etc/passwd, sau đó gán nó cho sizeGetter. Dựa theo flow vừa rồi, phương thức GetFileContent.execute sẽ trả về nội dung của file /etc/passwd và được hiển thị thông qua câu lệnh echo.

A screenshot of a computer program

Description automatically generated with medium confidence

Theo cách này, chúng ta có thể đọc được một nội dung của tập tin bất kỳ trên hệ thống mà không phụ thuộc vào xử lý logic của mã nguồn. Trong trường hợp này Image.\_\_toString được coi là source còn GetFileContent.execute sẽ là sink của gadget chain.

### Phân tích hoạt động của các gadget chain phổ biến trong CSharp

Deserialization trong ngôn ngữ PHP thường không phổ biến vì ngôn ngữ này không có các class mặc định, class chỉ được định nghĩa trong các thư viện hoặc framework lớn. Các thư viện này phục vụ cho ứng dụng theo mô hình hướng đối tượng (OOP). Do đó việc tìm một class có khả năng tấn công thường gây ra nhiều khó khăn và phụ thuộc nhiều vào môi trường của chương trình.

Trong mục này, chúng ta sẽ tìm hiểu về các gadget trong ngôn ngữ CSharp. Đây là ngôn ngữ hướng đối tượng, do đó có rất nhiều class mặc định được tải vào khi chạy chương trình. Một trong số đó là ObjectDataProvider thuộc namespace System.Windows.Data. Lớp này có tác dụng thực thi một non-static method trong class bất kỳ, đây là một sink lý tưởng trong quá trình tìm kiếm gadget chain. Đối với CSharp, formatter sẽ được sử dụng để định dạng kiểu dữ liệu đầu vào cũng như cách thức chuyển đổi kiểu dữ liệu đó. Trường hợp này ta sẽ sử dụng formatter XamlReader để làm source cho gadget, formatter này có định dạng Xaml (XML base). Ta sẽ có luồng thực thi như sau.

A screen shot of a computer program

Description automatically generated with low confidence

Nội dung method InvokeMethodOnInstance như sau.

A picture containing text, software, multimedia software, screenshot

Description automatically generated

Hàm này cho phép gọi tới phương thức bất kỳ với tên là MethodName của đối tượng \_objectInstance. Ta sẽ lựa chọn gọi đến phương thức Start của object System.Diagnostics, phương thức này cho phép thực thi một lệnh bất kỳ trong command prompt. Ở ví dụ này, ta sẽ chạy lệnh cmd /c calc.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 9: Kết quả pop-up calculator

MethodName được gán là Start còn ObjectInstance được gán là object của System.Diagnostics.Process, ngoài ra ProcessStartInfo có FileName là cmd và Arguments là /c calc.

Do khuyến cáo về bảo mật và đặc thù chức năng nên Xaml formatter thường ít được sử dụng trong các ứng dụng. Thực tế BinaryFormatter là formatter phổ biến cho quá trình De(serialization). Formatter này biến đổi dữ liệu dưới dạng nhị phân và thường được mã hóa Base64 để giảm thiểu kích thước của đoạn dữ liệu. Nội dung sau đây sẽ phân tích cách kết hợp gadget ObjectDataProvider với BinaryFormatter và tác dụng của magic method trong CSharp.

Xét với class TextFormattingRunProperties, class này định nghĩa một constructor với 2 tham số có kiểu dữ liệu là SerializationInfo và StreamingContext. Magic method này sẽ được gọi đến khi thực hiện deserialization object. Trong đó tham số SerializationInfo có thể được thay đổi được giá trị tùy ý trong lúc serialization, quá trình này được thực hiện từ phía người dùng mà không cần phụ thuộc vào mã nguồn chạy trên máy chủ ứng dụng.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 10: Magic method của class

Phương thức GetObjectFromSerializationInfo được gọi đến ngay tại dòng đầu tiên với các tham số lần lượt là “ForegroundBrush” và info – giá trị của SerializationInfo đã được cài đặt từ trước.

A screen shot of a computer program

Description automatically generated with low confidence

Tiếp theo GetObjectFromSerializationInfo thực hiện lấy giá trị string “ForegroundBrush” từ info và truyền trực tiếp vào XamlReader.Parse. Ta có luồng thực thi như sau.

A screenshot of a computer program

Description automatically generated with medium confidence

Nếu biến @string có giá trị như trong ví dụ trên thì có thể thực thi được lệnh trên command prompt để hiển thị được chương trình calculator. Trong trường hợp này phương thức XamlReader.Parse sẽ đóng vai trò là sink còn source chính là TextFormattingRunProperties.ctor. Chúng ta sẽ tạo một class mới cho quá trình serialization.

A picture containing text, screenshot

Description automatically generated

MiddleClass triển khai interface ISerializable, phương thức GetObjectData sẽ được thực thi mỗi khi serialize một đối tượng của nó. Tại dòng 19 và 20, class TextFormattingRunProperties sẽ được thêm vào SerializationInfo để báo hiệu cho formatter biết nó đang thực hiện chuyển đổi từ kiểu dữ liệu nào. Sau đó payload trong ví dụ trên được thêm vào SerializationInfo với chỉ mục là “ForegroundBrush” phù hợp với luồng thực thi đã đề ra. Thực hiện sesrialize và mã hóa Base64 ta được kết quả như sau.

A screen shot of a computer

Description automatically generated with medium confidence

Trong thực tế đây sẽ là giá trị dữ liệu đầu vào mà kẻ tấn công sử dụng. Tiếp tục thực hiện deserialization với kết quả trên và thực hiện lệnh pop-up calculator thành công.

A screenshot of a computer

Description automatically generated

Figure 11: Kết quả sau khi tùy chỉnh payload

TextFormattingRunProperties và ObjectDataProvider là hai gadget thường được sử dụng trong tấn công thực tế. Ngoài ra, chúng còn là tiền đề để phát triển những gadget chain mới sau này.

# **Insecure Deserialization trong thực tế**

Thời gian gần đây, kỹ thuật này được ứng dụng rất nhiều trong các cuộc tấn công mạng vào những hệ thống của tổ chức lớn. Chúng ta có thể kể đến một vài trường hợp như lỗ hổng CVE-2022-29108 được phát hiện vào tháng 5 năm 2022 và lỗ hổng CVE-2023-21744 được phát hiện vào tháng 1 năm 2023. Hai lỗ hổng này đều được đánh giá có mức độ ảnh hưởng nghiêm trọng tới hệ thống Sharepoint Server (ứng dụng quản lý trang web) của Microsoft. Chúng ta sẽ đi vào phân tích hai lỗ hổng này.

## **Lỗ hổng CVE-2022-29108**

Lỗ hổng xảy ra tại class ChartAdminPageBase namespace của Sharepoint.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 12: Đoạn code gây lỗi tại lớp ChartAdminPageBase

Tại dòng 93, đầu vào của quá trình deserialization được lấy từ phương thức FetchBinaryData với tham số là CustomSessionStateKey – một chuỗi định danh được sinh ra khi tạo một nội dung mới trên cơ sở dữ liệu và được trả về cho người dùng. Phương thức này chỉ thực hiện công việc đơn giản là lấy nội dung trong cơ sở dữ liệu với chuỗi định danh là tham số được truyền vào. Dễ dàng nhận thấy được rằng việc tấn công vào điểm yếu này là hoàn toàn khả thi khi kẻ tấn công có thể tạo một bản ghi chứa payload độc hại và tiếp tục truyền vào mã định danh của nội dung đó để thực hiện insecure deserialization. Trong quá trình này máy chủ không thực hiện bất kỳ công đoạn kiểm tra đầu vào nào trên dữ liệu của kẻ tấn công. Lỗ hổng này được nhà phát triển cập nhật bản vá vào tháng 5 năm 2022.

A picture containing screenshot, text

Description automatically generated

Figure 13: Mã nguồn sau khi vá lỗi

Tại dòng 98, đối tượng SafeSerialization được gán vào thuộc tính Binder. Đối tượng này sẽ thực hiện kiểm tra các kiểu dữ liệu trong SerializationInfo có nằm trong whitelist (danh sách các kiểu dữ liệu an toàn) expectedTypes hay không sau đó mới thực hiện quá trình deserialization. Tuy nhiên, đây là bản vá chưa hoàn chỉnh khi mà mã nguồn này tiếp tục bị tấn công sau đó.

## **Lỗ hổng CVE-2023-21744**

Lỗ hổng này là bypass của CVE-2022-29108 (phiên bản biến thể). Whitelist đã đề cập phía trên chứa các kiểu dữ liệu được cho là an toàn để phục vụ quá trình deserialization, trong đó có ChartWebPart.

A screen shot of a computer program

Description automatically generated with medium confidence

Figure 14: Nội dung whitelist

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 15: Kiểu dữ liệu DataSet được khai báo trong ChartWebPart

Đi sâu hơn vào class này, ta có thể thấy thuộc tính actionResult được định nghĩa có kiểu dữ liệu là DataSet. Đây là một kiểu dữ liệu được đánh dấu không an toàn và trên thực tế người ta đã tìm ra gadget chain dành cho kiểu dữ liệu này với luồng thực thi như sau.

DataSet.ctor()

-> DataSet.DeserializeDataSet()

-> DataSet.DeserializeDataSetSchema()

-> BinaryFormatter.Deserialize()

Như trên chúng ta có thể sử dụng gadget TextFormattingRunProperties với BinaryFormatter làm sink còn DataSet.ctor là source cho luồng tấn công này. Ngoài ra class này còn khai báo thêm kiểu dữ liệu DataTable cho thuộc tính providerTable, đây cũng là một kiểu dữ liệu có khả năng nằm trong gadget chain của cách thức tấn công này. Lỗ hổng này được phát hiện và cập nhật bản vá tại tháng 1 năm 2023.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Figure 16: Mã nguồn của bản vá hoàn chỉnh

Tại dòng 100, đối tượng DataSetSurrogateSelector được gán cho thuộc tính SurrogateSelector, điều này vẫn cho phép formatter thực hiện giải mã kiểu dữ liệu DataSet hoặc DataTable nhưng đã được khống chế đầu vào từ SerializationInfo.

Cả 2 lỗ hổng phía trên đều cho phép kẻ tấn công thực thi lệnh từ xa (Remote code execution) và được đánh giá ảnh hưởng ở mức nghiêm trọng. Những lỗ hổng sử dụng kỹ thuật insecure deserialization đều được đánh giá có mức rủi ro cao, gây mất an toàn thông tin và không đảm bảo tính toàn vẹn dữ liệu của ứng dụng. Ngoài ra, kỹ thuật này còn được sử dụng trong nhiều cuộc tấn công vào các hệ thống lớn khác như Exchange Mail Server, MoveIT File Transfer, …

# **Các biện pháp phòng ngừa và đảm bảo an toàn dữ liệu**

## **Nguyên tắc thiết kế an toàn cho quá trình giải mã**

Do mức độ ảnh hưởng lớn của kỹ thuật tấn công này, khi thiết kết phần mềm chúng ta cần tuân theo một số quy tắc sau.

* Hạn chế quyền của quá trình giải mã, chỉ cho phép truy cập vào các tài nguyên cần thiết. Điều này giúp giới hạn tiềm năng của cuộc tấn công và giảm thiếu tác động của lỗ hổng nếu có.
* Thực hiện kiểm tra và xác thực dữ liệu trước khi deserialize. Việc này bao gồm kiểm tra tính hợp lệ của định dạng, kiểm tra giá trị và kiểu dữ liệu.
* Áp dụng whitelist trong quá trình để chỉ chấp nhận giải mã các lớp và đối tượng xác định trước. Các lớp và đối tượng này sẽ được coi là an toàn trong ngữ cảnh giải mã. Điều này giúp loại bỏ các đối tượng có thể gây nguy hiểm và giảm thiếu khả năng tấn công.
* Theo dõi và cập nhật bản vá bảo mật cho các thư viện, framework cũng như toàn bộ sản phẩm. Đảm bảo rằng hệ thống đang sử dụng phiên bản mới nhất và có các bản vá để giảm thiếu lỗ hổng.
* Hạn chế thực hiện deserialization, chỉ cho phép thực hiện khi có chức năng hợp lý và hạn chế các chức năng nhạy cảm.
* Áp dụng kiểm soát kích thước dữ liệu để ngăn chặn các cuộc tấn công làm gián đoạn hoạt động của hệ thống.

## **Phương phác xác thực dữ liệu đầu vào**

Đối với mỗi formatter sẽ có những cách thức giải mã khác nhau, chính vì vậy ứng với từng formatter sẽ có những thuộc tính riêng biệt để thực hiện kiểm tra dữ liệu đầu vào; ví dụ như BinaryFormatter sẽ có thuộc tính Binder và SurrogateSelector như đã nói phía trên. Nhà phát triển cần phải sử dụng hợp lý các thuộc tính này trong việc xác thực dữ liệu của ngữ cảnh giải mã để giảm thiểu khả năng tấn công cũng như tác động của lỗ hổng.

Bên cạnh đó, việc kiểm tra nguồn gốc và tính toàn vẹn của dữ liệu đầu vào cũng rất quan trọng. Điều này đảm bảo sự tin cậy của dữ liệu cũng như hạn chế được các ngoại lệ có thể xảy ra, đảm bảo hiệu suất và tài nguyên hệ thống.

# **Kết luận**

## **Tổng kết nội dung**

Trên đây là một số nội dung liên quan đến lỗ hổng Insecure Deserialization và các biện pháp phòng ngừa. Chúng ta đã tìm hiểu về khái niệm và cơ chế hoạt động của Insecure Deserialization, cũng như những ảnh hưởng tiêu cực mà nó có thể gây ra đối với hệ thống và dữ liệu.

Để giảm thiếu rủi ro từ lỗ hổng, ta cần tuân thủ các nguyên tắc thiết kế cùng với việc áp dụng các biện pháp phòng ngừa hợp lý. Điều này bao gồm sử dụng mô hình whitelist, xác thực dữ liệu, kiểm soát quyền truy cập, sử dụng các bản vá bảo mật và kiểm soát kích thước khối dữ liệu.

Qua bài viết này, hy vọng người đọc đã có cái nhìn tổng quan về lỗ hổng Insecure Deserialization và nhận thức được tầm quan trọng của việc áp dụng các phương pháp an toàn để bảo vệ hệ thống khỏi những mối đe dọa tiềm ẩn.

## **Hướng phát triển và nghiên cứu tiếp theo**

Đây mới chỉ là những phân tích kỹ thuật về Insecure Deserialization, trong tương lai chúng ta có thể phát triển những hướng đi như.

* Nghiên cứu, phân tích công cụ tự động phát hiện và kiểm tra lỗ hổng trong mã nguồn ứng dụng.
* Nghiên cứu các kỹ thuật tấn công mới để hiểu sâu hơn về cách thức tấn công cũng như các biện pháp phòng ngừa triệt để.
* Xây dựng các framework và thư viện an toàn cho ngôn ngữ lập trình ngằm hỗ trợ việc thực hiện quá trình giải mã, bao gồm các tính năng xác thực và kiểm tra, kiểm soát dữ liệu đầu vào.
* Tìm hiểu các phương pháp tối ưu hóa quá trình giải mã để cải thiện hiệu suất cũng như tài nguyên của hệ thống trong lúc thực thi.

# **Tài liệu tham khảo**

* <https://www.php.net/manual/en/index.php>.
* <https://learn.microsoft.com/>.
* <https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf>.
* <https://github.com/pwntester/ysoserial.net>.
* <https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2022-29108>.
* <https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2023-21744>.