Анализ недостатков десериализации в приложениях на языке программирования PHP

Коваленко Диана Владимировна – студентка магистратуры Московского государственного университета имени М.В. Ломоносова.

Аннотация: В статье автор рассматривает недостатки десериализации в приложениях на языке PHP. Представлен метод автоматизированного анализа критичности недостатков данного типа. На основе этого метода создан инструмент для автоматизированного анализа кода приложения. В статье описан принцип работы инструмента и приведены результаты его экспериментального исследования.

Ключевые слова: Недостаток, уязвимость, анализ исходного кода, десериализация, PHP, магические методы PHP, статический анализ.

В объектно-ориентированных языках известно понятие состояния объекта. Под состоянием подразумевается набор текущих значений полей объекта. Для сохранения и передачи исходного состояния объекта с возможностью его дальнейшего восстановления используется обратимое преобразование объекта в специальное представление. Это преобразование называется **сериализацией**. Обратная операция, восстанавливающая объект в оперативной памяти, называется **десериализацией**
.

В веб-приложениях сериализация используется для сохранения состояния объектов, связанных с пользователями этого приложения, например, объекта сессии пользователя. В том числе сериализованные объекты, связанные с конкретным пользователем, могут передаваться пользователю (например, в cookie), а затем десериализовываться при последующих обращениях клиента в веб-приложение. Необходимость в подобных решениях возникает, например, в микросервисной архитектуре, т.е. тогда, когда серверная часть веб-приложения состоит из нескольких обособленных приложений без общего состояния. Используя ошибки в реализации приложений, злоумышленники могут получить возможность манипулировать десериализованными объектами, то есть объектами в оперативной памяти сервера. В зависимости от формата сериализации возникают различные риски информационной безопасности [6].

Форматы сериализации объектов с магическими методами

Магический метод – метод класса, вызов которого происходит автоматически при различных событиях, происходящих с объектом во время выполнения программы. В этой статье рассматривается формат сериализации языка PHP, который позволяет сериализовать только данные объекта (отсутствует возможность сериализации исполняемого кода), но при этом реализовывать в коде класса этого объекта т.н. магические методы, которые автоматически вызываются при сериализации или десериализации объекта.

В примере кода ниже присутствует вызов unserialize от данных, полученных в запросе от пользователя. А также определен класс Logger, деструктор которого записывает текущее значение поля $content в файл с названием $filename.

```
class Logger {
```

Листинг 1

Атакующий может сформировать строку, при десериализации которой будет создан объект класса Logger со значениями полей, например:

```
$filename = "shell.php";
```

```
$content = "<?php system($_GET
```

Листинг 2

Деструктор класса обязательно выполнится при уничтожении объекта в памяти, а значит, будет создан файл "shell.php".

Вызов unserialize от строки, сформированной клиентом, является недостатком кода, но

не обязательно является уязвимостью. Если в области видимости интерпретатора PHP в момент десериализации нет класса, объект которого может быть использован для атаки, как класс Logger в примере, то злоумышленник не сможет выполнить произвольный код или другие интересующие его действия с помощью данного недостатка. Поэтому для ответа на вопрос, к какой уязвимости приводит недостаток десериализации, требуется найти все классы, которые могут быть использованы злоумышленниками для манипуляции логикой приложения.

Анализ критичности недостатка десериализации

В случае десериализации недостатком является возможность вызывать в серверной части веб-приложения десериализацию произвольных данных. Задача определения наличия недостатка кода может быть решена с применением статических анализаторов безопасности PHP-приложений. Одной из возможностей статических анализаторов безопасности кода является проверка нарушения принципа невмешательства [10], т.е. ответ на вопрос, могут ли недоверенные пользовательские данные попасть в доверенный канал команд. Применительно к десериализации статические анализаторы отвечают на вопрос, может ли атакующий манипулировать параметром, от которого происходит вызов unserialize, то есть определяют наличие недостатка кода.

Уязвимость – недостаток в системе, который может быть использован злоумышленником для нарушения конфиденциальности, целостности или доступности этой системы. Если недостаток десериализации обнаружен при вызове unserialize в PHP, то для подтверждения наличия уязвимости требуется провести анализ кода веб-приложения на предмет возможности проведения атаки.

Веб-приложения, написанные на PHP на базе фреймворков (например, Laravel или Yii) или CMS, содержат большой объем исходного кода. Например, CMS Drupal версии 8.7.0 использует Composer и содержит 1350 классов и 928 магических методов, что затрудняет анализ недостатка вручную, поэтому появляется необходимость применения автоматизированного анализа.

Периодически публикуется список наиболее критичных проблем безопасности веб-приложений OWASP Top 10. Актуальная публикация [1] датируется 2017 годом. В ней проблемы десериализации вынесены в отдельную категорию – A8. Это обосновано тем, что недостатки десериализации могут приводить к различным разрушительным атакам, в том числе к удаленному выполнению кода. Более того, каждый год официально регистрируется более 10 уязвимостей десериализации в базе CVE.

Существует открытый проект PHPGGC [13], являющийся коллекцией полезных нагрузок для эксплуатации десериализации в некоторых известных PHP фреймворках. Под полезной нагрузкой понимается строка, попадание которой в вызов unserialize в качестве параметра приводит к успешной атаке PHP-приложения. Проект PHPGGC можно использовать для демонстрации недостатка десериализации в приложениях, которые содержат используемые фреймворки, или для анализа приложений, код которых неизвестен (методом перебора всех полезных нагрузок из данного проекта). Однако проект не подразумевает проведение анализа уникального кода и не реализует методов поиска уязвимостей, связанных с десериализацией в PHP.

Команда разработчиков статического анализатора PHP кода Rips Tech опубликовала статью [2], описывающую метод автоматизированного построения объектов, которые могут применяться для атаки на приложение с недостатком десериализации. Описанный подход позволяет подтвердить наличие недостатка десериализации в приложении и выявить связанные с ним уязвимости, но его реализация предполагает функционирование в рамках закрытого коммерческого статического анализатора [5] Rips Tech.

На данный момент в открытом доступе нет решений, которые позволяют анализировать недостаток десериализации в конкретном PHP приложении и автоматизированно подтверждать наличие уязвимости.

Метод автоматизированного анализа недостатков десериализации

Для формирования типовой схемы атаки на приложение, содержащее недостаток десериализации, была проанализирована уязвимость CVE-2015-8562 [14]. В результате анализа данной уязвимости разработан метод анализа недостатков десериализации. Для иллюстрации метода приводится характерный пример кода, содержащего недостаток десериализации. В данном примере атакующий может десериализовать объект класса `Connection` таким образом, что выполнится функция `file_put_contents`, которая запишет в файл ‘`logs.txt`’ текст, заданный атакующим.

class	Connection {

Листинг 3. Пример недостатка десериализаци

Для проверки наличия уязвимости, то есть подтверждения возможности проведения успешной атаки, необходимо составить набор объектов классов (т.н. **цепочку гаджетов** или цепочку эксплуатации недостатка десериализации), в случае десериализации которого в анализируемом приложении, будет выполнен код атакующего.

В данном примере цепочка гаджетов должна состоять из объектов классов Connection , StateManager и State со следующими значениями свойств:

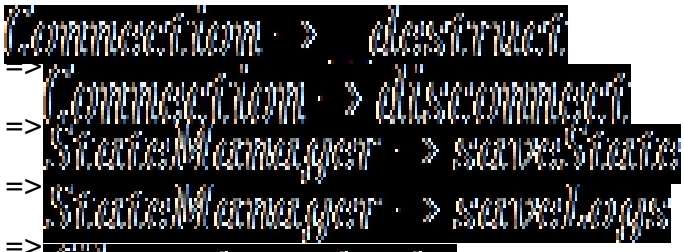
class	Connection {

Листинг 4. Цепочка гаджетов.

Значение строки Sexit message будет записано в файл 'logs.txt', поэтому указывается атакующим в зависимости от целей атаки.

Под **целевой функцией** понимается любая функция, вызов которой представляет интерес для атакующего. Цепочке гаджетов соответствует **цепочка вызовов** функций, которая заканчивается вызовом целевой функции.

В примере целевой функцией является file_put_contents. Цепочка вызовов состоит из функций:



Инструмент автоматизаций анализа критичности недостатка десериализации

Метод анализа недостатка десериализации реализован в инструменте PHP-Chain [15]. Работа инструмента разбита на два этапа: построение цепочек и анализ их применимости. Входными данными для инструмента является исходный код PHP проекта, собранного с помощью Composer, и файл конфигурации, содержащий набор PHP функций, которые могут представлять интерес для атакующего.

1. Построение потенциальных цепочек вызовов

Для построения потенциальных цепочек собирается информация обо всех методах, функциях и их вызовах в коде анализируемого проекта. Формируется структура, которая обладает следующими свойствами:

1. Соответствует иерархии классов проекта [12].
2. В каждом элементе содержится объявление метода или функции (имя, набор параметров вызова).
3. В каждом элементе содержится реализация метода или функции, содержащая список вызовов методов и функций.

После того, как кода проекта сохранен в описанной структуре, строятся цепочки вызовов по следующему алгоритму:

1. Для каждого магического метода, определенного в конфигурации инструмента, собирается список всех функций, вызываемых из данного метода.
2. Для каждой функции из списка определяется, является ли она целевой. Если функция входит в перечень целевых, то зафиксировать создание цепочки и вернуть результат. В противном случае этот алгоритм рекурсивно запускается для данной функции.

3. Если встретился вызов метода объекта, то обнаруживаются все классы, в которых описан метод с таким же названием. Затем этот алгоритм рекурсивно запускается для каждого обнаруженного метода класса.

Если на предыдущем шаге была собрана информация обо всех методах и функциях, определенных в проекте, то алгоритм построения цепочек обеспечивает нахождение всех тех из них, которые не содержат динамических конструкций.

2. Статический анализ потенциальных цепочек

После того, как сформирован набор цепочек вызовов в виде дерева, проводится оценка применимости потенциальных цепочек. Применимость цепочки зависит от количества контролируемых параметров целевого вызова. Для оценки возможности контролирования значений набора переменных целевого вызова решается задача поиска зависимостей по данным. Эта задача сводится к построению обратного среза (backward slicing [8]) для точки кода. Существуют известные алгоритмы ее решения [3]. Эту задачу можно решать с использованием SSA [4], построенного на основе графа потока управления (CFG). В SSA форме каждый блок графа потока управления состоит из операторов, которые содержат информацию об операндах. Если значение операнда зависит от результата выполнения операторов, то операнд содержит список соответствующих операторов. SSA представление переменных подразумевает, что для каждого присваивания значения переменной создается новая версия переменной. Если существует несколько версий одной переменной, то вводится специальный оператор Phi, который содержит информацию о версиях.

Так как SSA форма содержит информацию о том, какие операторы влияют на значение операнда и какие операнды являются параметрами операторов, обход SSA формы позволяет отследить зависимости по данным и построить обратный срез. На основе информации о переменных и операциях в обратном срезе делается вывод о контролируемости переменных целевого вызова.

Для повышения точности анализа применимости цепочки решаются следующие задачи статического анализа кода:

- задача распространения констант [7];
- задача анализа типов для параметров функции;
- поиск недостижимого кода на основе информации о потоке данных, полученной в результате распространения констант;

- для учета операторов ветвлений решается задача построения контекстно-чувствительного обратного среза [9].

А также для учета особенностей целевого вызова при оценке применимости цепочки реализуется статический анализ цепочки вызовов на основе правил [11].

Экспериментальное исследование инструмента

Целью экспериментального исследования является оценка эффективности инструмента для автоматизированного анализа недостатка десериализации. Был выполнен анализ фреймворков проекта PHPGGC на предмет возможности поиска в них цепочек вызовов. Для 80% фреймворков были найдены цепочки вызовов для анализа недостатка десериализации. Для всех найденных цепочек сформированы оценки их применимости, с учетом которых был выполнен ручной поиск цепочки вызовов для составления цепочки гаджетов. Для оценки эффективности автоматизированного анализа недостатка десериализации результаты оценивались следующим образом:

- Был выполнен замер времени, который потребовался аналитику для построения цепочки гаджетов.
- Было зафиксировано, сколько функций вызовов потребовалось пометить аналитику как неприменимые для анализа недостатка десериализации, прежде чем была найдена цепочка, на основе которой ему удалось составить цепочку гаджетов.

Замер времени отражает удобство использования инструмента при анализе недостатке десериализации, а число просмотренных в ходе анализа функций характеризует точность оценок применимости цепочек, полученных в результате статического анализа.

Замер времени ручного анализа цепочек вызовов фреймворков проекта PHPGGC

T – время анализа аналитиком результата, в минутах.

N – число просмотренных аналитиком функций результата.

Таблица 1

Название фреймворка	**Версия**	**T**	**N**
CodeIgniter4	4.0.0-beta.1	30	12
Doctrine	2.1.0		

71

>200

Drupal7

7,63

85

>60

Guzzle

6.3.2

10

0

Monolog

1.23

30

23

Monolog

1.17

40

22

Slim

3.8.1

20

11

SwiftMailer

5.4.8

3

1

SwiftMailer

6.0.0

15

8

SwiftMailer

5.0.1

15

5

SwiftMailer

4.0.0

10

3

Yii

1.1.20

50

>200

ZendFramework

1.12.20

60

>50

На основе данных экспериментального исследования можно сделать вывод о применимости инструмента для автоматизированного анализа недостатка десериализации в приложениях на языке PHP.

Заключение

Недостаток десериализации является одной из 10 наиболее критичных угроз безопасности web-приложений. Анализ критичности недостатка в PHP приложениях представляет собой трудоемкий процесс, который не производится средствами статических анализаторов безопасности PHP кода, выложенными в открытый доступ. В данной работе представлен метод автоматизированного анализа недостатка десериализации. А также реализован инструмент, автоматизирующий анализ критичности недостатка. В ходе экспериментального исследования была показана применимость инструмента, однако в дальнейшем он может быть доработан с целью уменьшения времени, требующегося аналитику для обработки результатов работы инструмента.

Список литературы

1. Топ-10 OWASP – 2017. Десять самых критичных угроз безопасности веб-приложений. – Текст : электронный // The Open Web Application Security Project : [сайт]. – URL: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017-ru.pdf (дата обращения: 29.05.2020).
2. Johannes, Dahse Code reuse attacks in PHP: Automated POP chain generation / Dahse Johannes, Krein Nikolai, Holz Thorsten. – Текст : непосредственный // Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. – CCS '14, 2014. – С. 42–53.
3. Mark, Weiser Program slicing / Weiser Mark. – Текст : непосредственный // ICSE '81: Proceedings of the 5th international conference on Software engineering. – 1981. – С. 439–449.
4. Robert, Oehlmann Static Single-Assignment for Program Slicing on Binary Intermediate Language / Oehlmann Robert. – Текст : электронный // Hamburg University of Technology

(TUHH) : [сайт]. – URL: https://www.sts.tuhh.de/pw-and-m-theses/2013/oehlm13.pdf (дата обращения: 29.05.2020).

5. Johannes, Dahse A Static Source Code Analyser for Vulnerabilities in PHP Scripts / Dahse Johannes. – Текст : электронный // RIPS : [сайт]. – URL: https://www.nds.ruhr-uni-bochum.de/media/nds/attachments/files/2010/09/rips-paper.pdf (дата обращения: 29.05.2020).

6. Abdelazim, Mohammed Deserialization Vulnerability / Mohammed Abdelazim. – Текст : электронный // Exploit Database : [сайт]. – URL: https://www.exploit-db.com/docs/english/44756-deserialization-vulnerability.pdf (дата обращения: 29.05.2020).

7. Anders, Møller Static Program Analysis / Møller Anders. – Текст : электронный // au.dk : [сайт]. – URL: https://cs.au.dk/~amoeller/spa/spa.pdf (дата обращения: 29.05.2020).

8. Касьянов, В. Н. Slicing: Срезы программ и их использование / В. Н. Касьянов, И. Л. Мирзуитова. – Сибирское отделение РАН, 2002. – 116 с. – Текст : непосредственный.

9. Joxan, Jaffar Path-Sensitive Backward Slicing / Jaffar Joxan, Murali Vijayaraghavan, A. N. Jorge, Santosa Andrew. – Текст : непосредственный // Conference: Proceedings of the 19th international conference on Static Analysis. – 2012.

10. Chiara, Bodei Techniques for Security Checking: Non-Interference vs Control Flow Analysis / Bodei Chiara, Degano Pierpaolo, Focardi Riccardo, Martinelli Fabio. – Текст : непосредственный // Electronic Notes in Theoretical Computer Science 62. – : 2002.

11. Octavian, Udrea Rule-based static analysis of network protocol implementations / Udrea Octavian, Lumezanu Cristian, Lumezanu Cristian, S. F. Jeffrey. – Текст : непосредственный // Information and Computation 206. – : DBLP, 2008. – С. 130-157.

12. Frank, TipPeter Class Hierarchy Specialization / TipPeter Frank, F. Sweeney. – Текст : непосредственный // Acta Informatica. – 2000. – № 36(12).

13. Репозиторий проекта PHPGGC. – Текст : электронный // Github : [сайт]. – URL: https://github.com/ambionics/phpggc (дата обращения: 29.05.2020).

14. Joomla! 1.5 < 3.4.5 - Object Injection Remote Command Execution. – Текст : электронный // Exploit Database : [сайт]. – URL: https://www.exploit-db.com/exploits/38977 (дата обращения: 29.05.2020).

15. Репозиторий инструмента – PHP-Chain. – Текст : электронный // Github : [сайт]. – URL: https://github.com/dsp25no/php-chain (дата обращения: 29.05.2020).

{social}