

CSCM28: Security Vulnerabilities and Penetration Testing – Coursework 1

Analysis of the Insecure Deserialization Vulnerability

Luke Hengstenberg

878876

11/03/2020

Table of Contents

I.	Introduction.....	1
II.	Overview.....	1
III.	Example Exploits.....	2
IV.	Possible Solutions.....	4
V.	Conclusion: Prominence today.....	5
VI.	References.....	7

Table of Figures

1)	Figure 1: Screenshot of generated session cookie from [12].....	2
2)	Figure 2: Screenshot of code deserializing cookie from [12].....	3
3)	Figure 3: Screenshot of hex editor viewing cookie from [12].....	3
4)	Figure 4: Screenshot of hex editor with updated admin cookie from [12].....	3
5)	Figure 5: Screenshot of serialization command creating .NET object from [13].....	4
6)	Figure 6: Screenshot of API call from [13].....	4

I. Introduction

Serialization is the process in which an object is converted into a stream of bytes for storage or transmission [1]. Serialization is used to save the objects state so it can be recreated when it is needed by some application or service through a reversing deserialization process. Deserialization becomes a vulnerability when serialized objects are accepted without correctly verifying the source, meaning raw data is retrieved from the file or network socket to reconstruct an object state that has been altered or falsified to give the attacker some additional access, information or permissions [2]. The insecure deserialization vulnerability can affect any web or software application where the serialization process is public knowledge or easy to deduce and/or where there exist flaws in or lack of integrity checks prior to accepting serialized objects.

Examples include:

- Insecure Java Deserialization function used by Cisco Security Manager releases prior 4.18, allowing an “unauthenticated remote attacker” to gain root privileges and “execute arbitrary commands on an affected device” [3]. Similar Java deserialization vulnerability affecting JetBrains TeamCity before version 2019.1.4 [4], Cisco Unity Express (CUE) releases prior 9.0.6 [5], Cisco Secure Access Control System (ACS) prior to 5.8 patch 9 [6].
- Vulnerabilities in WordPress Carts Guru 1.4.5 plugin via a cartsguru-source cookie to event handler php classes [7], and WordPress Virim 0.4 plugin via s_values, t_values, or c_values in graph.php [8], both allowing attacker controllable data to be passed to an unserialize function with no authentication required.
- Similarly to the WordPress cases the structure of the download.php class in inoERP 4.15 allowed queries to be sent to an unserialize function, these queries were then translated into SQL queries enabling SQL injection attacks [9].

It is clear from these examples that insecure deserialization is still appearing and posing a danger to modern applications. The vulnerability is easily overlooked and can be introduced by third-party plugins without the knowledge of the developer. This report explores insecure deserialization as follows:

- Section II provides a technical overview of what insecure deserialization is and why it exists.
- Section III uses examples to illustrate how the vulnerability is exploited by an attacker.
- Section IV investigates a range of possible solutions to patch the vulnerability and tighten security.
- Section V concludes with an evaluation of the vulnerability’s prominence in modern day.

II. Overview

To understand what the insecure deserialization vulnerability is and why it exists one must look at how serialization is used in an application. As described in the previous section serialization is where an object is converted into a stream of bytes to store its state in a file, database or memory, until it is required and deserialized [1]. The complexity of modern

applications means system components are usually distributed and must share and store lots of data. A serialized object can be sent and stored much more efficiently and securely than the raw data form if the process to serialize and deserialize it is not vulnerable. Serialization operations are common practice and seen in most architectures that include Client-side MVC, APIs and Microservices [10], with most programming languages providing functions to serialize data. Serialization only becomes a vulnerability when the underlying infrastructure is flawed, allowing user input to be given to the deserialization function by an untrusted source without appropriate verification.

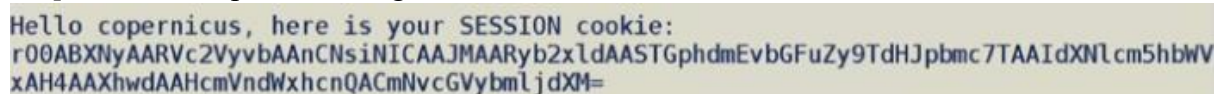
Insecure deserialization vulnerabilities stem from the customizability of serialization processes provided by programming languages, which can easily be misconfigured by the developer or insecure in themselves and therefore abused by an attacker. As seen in the WordPress examples, insecure deserialization may also be accidentally introduced through a weakness in a third-party plugin. Λαβδάνης [11] states that due to the vulnerability allowing unauthorized code to be introduced “almost any malicious intent is possible”, including DoS, data exposure, authentication bypasses, and remote code execution. If the serialization format can be predicted from looking at the serialized objects, then the attacker can modify or replicate a serialized object that executes commands to change user state such as giving them administrator permissions. The attacker can then use an unsecured deserialization function to execute or store the malicious code. If the deserialization function accepts any serialized object then there is the potential to hide any command inside a serialized object, passing it to the function to be remotely executed upon deserialization.

The next section illustrates how the vulnerability can be exploited to give an attacker admin privileges and to remotely execute code.

III. Example Exploits

The first example insecure deserialization exploit was selected from the YouTube video [12] which details how a weakness in Java serialization functions can be used to give an attacker admin privileges through editing a serialized cookie object. An attacker would follow the following steps:

- 1) The attacker would make a normal user account and login to obtain their serialized session cookie generated by the application and stored in the browser. For example, in [12] the user “Copernicus” is given the cookie:



```
Hello copernicus, here is your SESSION cookie:  
r00ABXNyAARVc2VyvbAAnCNsiNICAAMAAJMAARyb2xldAASTGphdmEvdGFuZy9TdHJpbmc7TAAIdXNlcm5hbWV  
xAH4AAxhwdAAHcmVndWxhcncQACmNvcGVybmljdXM=
```

Figure 1: Screenshot of generated session cookie from [12].

- 2) The code RO0 identifies the serialized object as being Java based and therefore an attack can be crafted to target the specific vulnerabilities in Java serialization functions.
- 3) The attacker proceeds to print the cookie to a file called cookie.ser and base64 deserialize it with a standard base64 decoder to view the information it stores:

```
~/serial/2. cookies$ echo r00ABXNyAARVc2VybmAAnCNsiNICAAJMAARyb2xldAASTGphdmEvdGFuZy
9TdHJpbmc7TAAIdXNlcm5hbWVxAH4AAAXhwdAAHcmVndWxhcncQACmNvcGVybmljdXM= | base64 -d > co
okie.ser
```

Figure 2: Screenshot of code deserializing cookie from [12].

- 4) Due to the attacker's ability to replicate the deserialization function the structure of the cookie can now be viewed and modified in a hex editor:

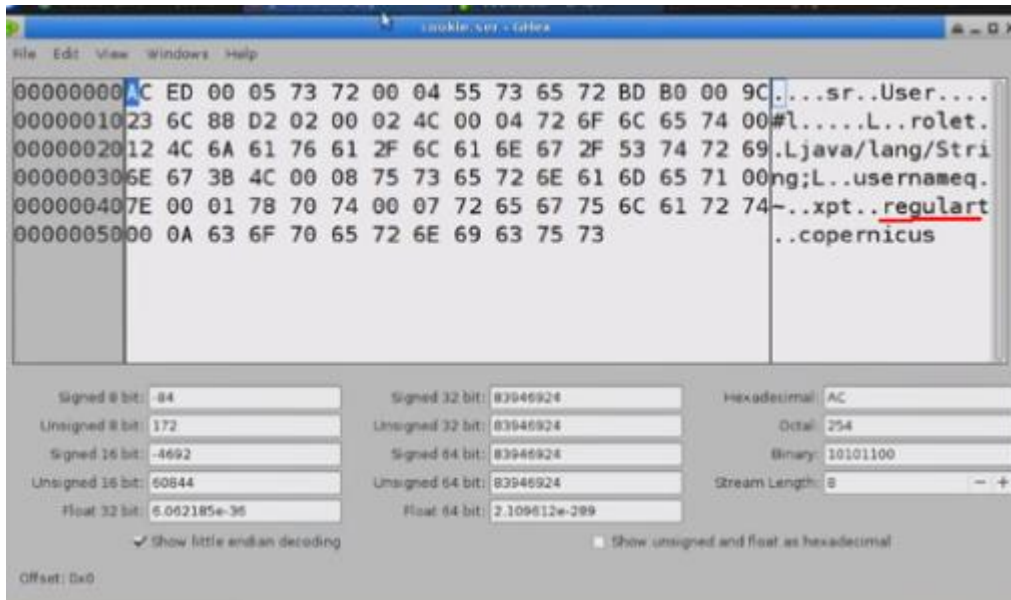


Figure 3: Screenshot of hex editor viewing cookie from [12].

- 5) As seen by the red line in figure 3, Copernicus is identified as a “regular” user. The attacker proceeds to modify this to “administrator”.

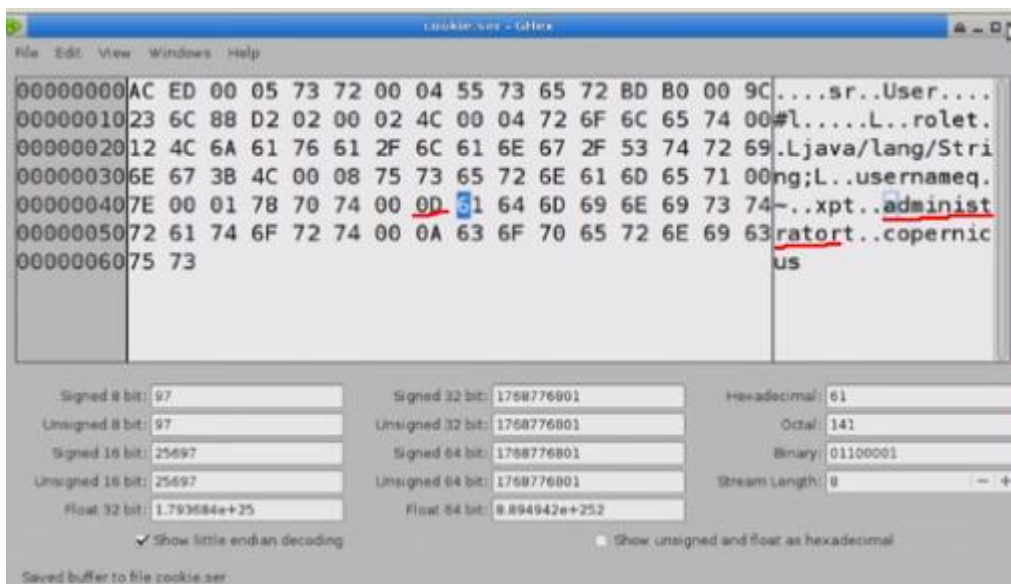


Figure 4: Screenshot of hex editor with updated admin cookie from [12].

Due to the length of the string changing the attacker also adjusts the bytes from 07 to 0D, reflecting the new length.

- 6) Finally, the attacker reserializes the object using a standard base64 encoder and uses the updated session cookie which when deserialized considers the attacker to be an administrator.

Serious damage can now be done with the new privileges the attacker has acquired, allowing further attacks to be conducted.

The second example insecure deserialization exploit was a real-world case in 2018 affecting the RESTful API behind CyberArk Password Vault Web Access [13]. The vulnerability existed because the application used serialized .NET objects as authentication tokens which could be crafted by an attacker to remotely execute code on the web server. Much like the previous example, these tokens contained session information and used base64 encoding. An attacker could follow the following steps based on the proof of concept in [13]:

- 1) The attacker begins by crafting a malicious serialized .NET object using some gadget ([13] used ysoserial.net) containing the command to be executed by the API, which for testing purposes is ping to see if there is a response.

```
$ ysoserial.exe -f BinaryFormatter -g TypeConfuseDelegate -o base64 \
-c "ping 10.0.0.19" > execute-ping.txt
```

Figure 5: Screenshot of serialization command creating .NET object from [13].

- 2) The attacker then invokes an API call and puts the malicious .NET object in its authorization header to execute it against the server.

```
$ curl -s -X GET -k \
--url 'https://10.0.0.6/PasswordVault/WebServices/PIMServices.svc/' \
'Applications/?Location=\&IncludeSublocations=true' \
--header "authorization: $(cat execute-ping.txt)" \
--header 'content-type: application/json'
```

Figure 6: Screenshot of API call from [13].

- 3) The attackers code has now been successfully executed receiving the ICMP packets from the ping response (in this example) or the result of some other code.
- 4) This exploit can give the attacker permanent access through methods such as creating a persistent backdoor. The attacker can craft new requests and execute them without any credentials, using the vulnerability to compromise the entire system.

IV. Possible Solutions

The previously explored sections have made the severity of an insecure deserialization vulnerability evident. Upon finding such a vulnerability action must be taken by the developer to fix the issue imminently. This section discusses a range of possible fixes that can be used to patch an underlying serialization issue, most of which should be used in conjunction rather than individually. Inspiration for these solutions was taken from sources [10, 14, 15] and developed by the researcher.

- 1) **Maintain integrity:** Checking the integrity of serialized objects should be made possible to ensure they have not been tampered with. A mechanism such as digital signatures should be employed so the application knows the serialized object has not been altered by checking if the signature is the same before deserializing it.
- 2) **Avoid/Constrain user input:** Where possible avoid accepting any user input unless it is completely necessary. In cases where user input is required thoroughly validate and constrain the class type before serialization/deserialization.
- 3) **Research application functionality:** Investigate the functions the application is using for deserialization in relation to known vulnerabilities for the programming language they are written in. When a variation of a function is found to be insecure more secure alternatives are usually available and should be adopted.
- 4) **Test third-party plugins:** Consider any third-party plugins that use serialization as possible weak points and conduct thorough testing of serialization/deserialization functions, adopting alternatives if their security is questionable.
- 5) **Minimize remote execution:** Ensure a minimal amount of code is run directly from serialized objects and isolate deserialized code in controlled environments running with low privileges, separate from high privilege environments.
- 6) **Safe authentication:** Use serialized objects to store limited sensitive credentials with a serialization process that is secure, unique and unpredictable. If serialized tokens are used the credentials held should be encrypted and not include permissions. Further integrity checks should be conducted as well as employing other levels of authentication in conjunction.
- 7) **Logging:** Log all cases where failed deserialization attempts have been made and exceptions have been thrown. This alerts you to the fact that you may be the target of an attack and helps give time to prepare for further attack attempts.
- 8) **Monitoring:** Monitor network activity for incoming and outgoing connections and monitor deserialization attempts closely to identify users with suspicious activity.
- 9) **Use security tools:** Utilize popular security tools proven to protect against insecure deserialization attacks such as Hdiv RASP [10]. Ensure these tools have enough visibility to be flexible in identifying and dealing with various threats.
- 10) **Language formats:** Avoid using native binary formats in the application and favour the use of language-agnostic formats such as YAML or JSON [10].

V. Conclusion: Prominence today

The Insecure deserialization vulnerability is a serious security concern that can be easily overlooked if specific testing is not conducted. It can give an attacker the potential to remotely execute specially crafted malicious code to perform an endless variety of different commands. For this reason, it placed in OWASP's top 10 web application security risks of 2017 at number 8 as a new contender [16]. However, its prominence today has arguably decreased with less occurrences being reported. In positive technologies 2018 report [17] it was found that insecure deserialization was only a medium risk vulnerability affecting 2% of tested web applications. In the 2019 report [18] it was omitted from the statistics entirely, whilst most other OWASP top 10 vulnerabilities maintained their prominence.

On the other hand, accurate occurrence statistics are hard to come by (if they exist at all) as the vulnerability is likely to be associated with and recorded under popular attacks such as

RCE (Remote Code Execution) and DoS (Denial of Service) by most security researchers. Overall, the reduction in prominence of the insecure deserialization vulnerability can be seen as a positive for the industry if it means it is being widely recognised and addressed in modern web applications. However recent CVE entries show it is very much alive and still appearing in versions of software in 2020 [19]. The developer should adopt the solutions explored in the previous section and take a great deal of care when an application uses deserialization, especially if it has been provided by an “out of the box” function from a programming language or third-party plugin. This vulnerability is not one that should be underestimated, and the researcher feels it should still be considered one of high risk.

VI. References

- [1] Microsoft, “Serialization (C#),” docs.microsoft.com, Feb. 1, 2020, [Online], Available: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>, [Accessed Mar. 7, 2020].
- [2] V. Dehalwar, A. Kalam, M. L. Kolhe and A. Zayegh, "Review of web-based information security threats in smart grid," in *2017 7th International Conference on Power Systems (ICPS)*., Pune, 2017, pp. 849-853.
- [3] Cisco, “Cisco Security Manager Java Deserialization Vulnerability,” tools.cisco.com, Oct. 2, 2019, [Online], Available: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20191002-sm-java-deserial>, [Accessed Mar. 7, 2020].
- [4] NIST, “National Vulnerability Database, CVE-2019-18364 Detail,” nvd.nist.gov, Nov. 1, 2019, [Online], Available: <https://nvd.nist.gov/vuln/detail/CVE-2019-18364#vulnCurrentDescriptionTitle>, [Accessed Mar. 7, 2020].
- [5] Cisco, “Cisco Unity Express Arbitrary Command Execution Vulnerability,” tools.cisco.com, Nov. 7, 2018, [Online], Available: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20181107-cue>, [Accessed Mar. 7, 2020].
- [6] Cisco, “Cisco Secure Access Control System Java Deserialization Vulnerability,” tools.cisco.com, Mar. 7, 2018, [Online], Available: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180307-acs2>, [Accessed Mar. 7, 2020].
- [7] WPVulnDB, “Carts Guru <= 1.4.4 - Unauthenticated Object Injection,” wpvulndb.com, May. 27, 2019, [Online], Available: <https://wpvulndb.com/vulnerabilities/9292>, [Accessed Mar. 7, 2020].
- [8] WPVulnDB, “Virim - Unauthenticated Object Injection,” wpvulndb.com, May. 27, 2019, [Online], Available: <https://wpvulndb.com/vulnerabilities/9291>, [Accessed Mar. 7, 2020].
- [9] Exploit Database, “inoERP 4.15 - 'download' SQL Injection,” exploit-db.com, Sept. 26, 2019, [Online], Available: <https://www.exploit-db.com/exploits/47426>, [Accessed Mar. 7, 2020].
- [10] D. Blazquez, “Insecure Deserialization: attack examples and mitigation,” hdivsecurity.com, Apr. 15, 2019, [Online], Available: <https://hdivsecurity.com/bornsecure/insecure-deserialization-attack-examples-mitigation/>, [Accessed Mar. 8, 2020].
- [11] Γ. Λαβδάνης, “PHP object injection and JAVA deserialization vulnerabilities in web applications,” in *Master's thesis, Πανεπιστήμιο Πειραιώς*, 2019, pp. 5-6.
- [12] OWASP, “Deserialization: what, how and why [not] - Alexei Kojenov - AppSecUSA 2018,” Youtube, Nov. 23, 2018, [Video file], Available: <https://www.youtube.com/watch?v=t-zVC-CxYjw>, [Accessed Mar. 8, 2020].

- [13] Exploit Database, “CyberArk Password Vault Web Access < 9.9.5 / < 9.10 / 10.1 - Remote Code Execution,” exploit-db.com, Apr. 9, 2018, [Online], Available: <https://www.exploit-db.com/exploits/44429>, [Accessed Mar. 8, 2020].
- [14] Sucuri, “OWASP Top 10 Security Risks & Vulnerabilities,” sucuri.net, 2020, [Online], Available: <https://sucuri.net/guides/owasp-top-10-security-vulnerabilities-2020/>, [Accessed Mar. 9, 2020].
- [15] Detectify, “OWASP TOP 10: Insecure Deserialization,” blog.detectify.com, Mar. 21, 2018, [Online], Available: <https://blog.detectify.com/2018/03/21/owasp-top-10-insecure-deserialization/>, [Accessed Mar. 9, 2020].
- [16] OWASP, “OWASP Top Ten,” owasp.org, 2020, [Online], Available: <https://owasp.org/www-project-top-ten/>, [Accessed Mar. 9, 2020].
- [17] Positive Technologies, “Web application vulnerabilities: statistics for 2018,” ptsecurity.com, Mar. 5, 2019, [Online], Available: <https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-statistics-2019/>, [Accessed Mar. 9, 2020].
- [18] Positive Technologies, “Web Applications vulnerabilities and threats: statistics for 2019,” ptsecurity.com, Feb. 13, 2020, [Online], Available: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>, [Accessed Mar. 9, 2020].
- [19] CVE, “Search Results: Deserialization,” cve.mitre.org, 2020, [Online], Available: <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=deserialization>, [Accessed Mar. 9, 2020].