Template Engines: A Methodology for Assessing Server-Side Code Execution Vulnerabilities

Lorenzo Pisu,

Giorgio Giacinto

PRA lab @ University Of Cagliari, Italy

Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

SERICS
SECURITY AND RIGHTS IN THE CYBERSPACE

About us

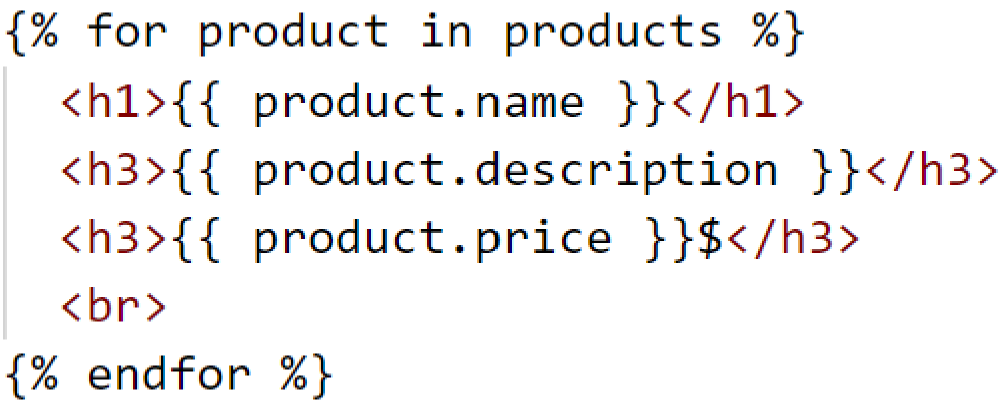
- **PRA Lab** is a research group focused on machine learning for security applications. The cybersecurity division includes
  - Web Security
  - Malware detection, analysis and classification
  - Network Security
  - Vulnerability and threat detection

About us

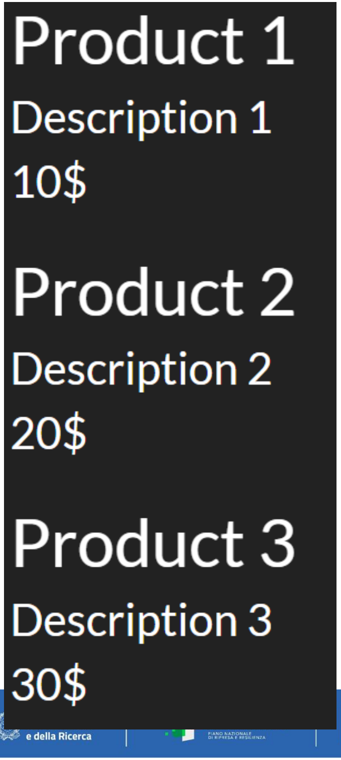
- **Srdnlen** is a CTF team on the top 50 of the global scoreboard of CTFTime
- We participate in international cybersecurity competitions with various topics
  - Web security
  - Software security
  - Forensics
  - Cryptography
- We publish our results on www.srdnlen.it

Template Engines - Use Case

- **Template Engines** are used to dynamically generate pages, their usage is nowadays essential
  - To generate dynamic dashboards with user data
  - To list products in ecommerce
  - Blogs, forums, social networks



```
{% for product in products %}
  <h1>{{ product.name }}</h1>
  <h3>{{ product.description }}</h3>
  <h3>{{ product.price }}$</h3>
  <br>
{% endfor %}
```



Product 1

Description 1

10$

Product 2

Description 2

20$

Product 3

Description 3

30$

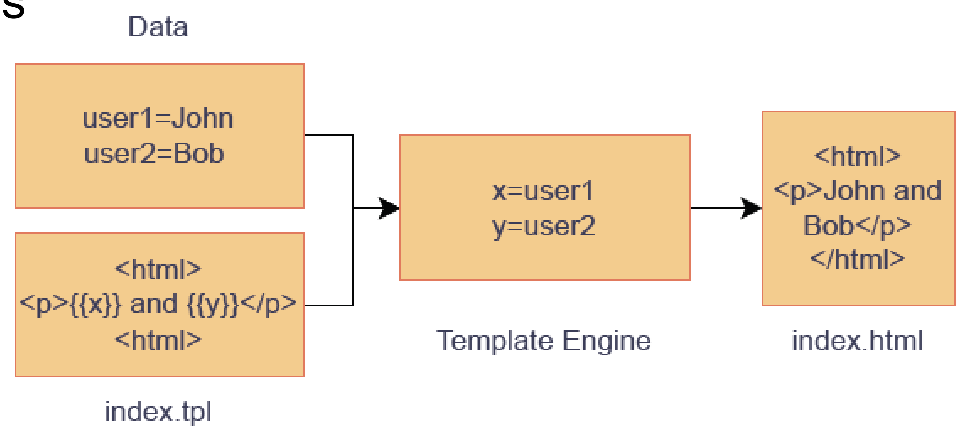
Finanziato dall'Unione europea NextGenerationEU
e della Ricerca
SERICS

Template Engines

Template Engines are software components, typically provided as libraries or modules

They **parse and manipulate** strings or files according to predefined syntactic rules

They apply **tokenization**, breaking strings or files into structured representations. This process allows binding data to **placeholders**, applying transformations, and executing conditional logic and loops



Data

user1=John
user2=Bob

<html>
<p>{{x}} and {{y}}</p>
<html>

index.tpl

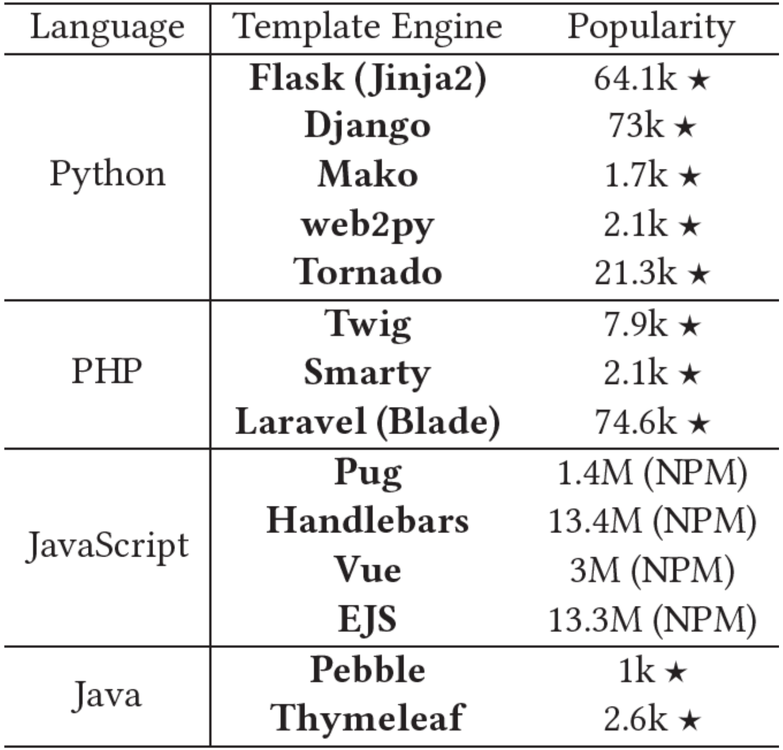
x=user1
y=user2

Template Engine

<html>
<p>John and Bob</p>
</html>

index.html

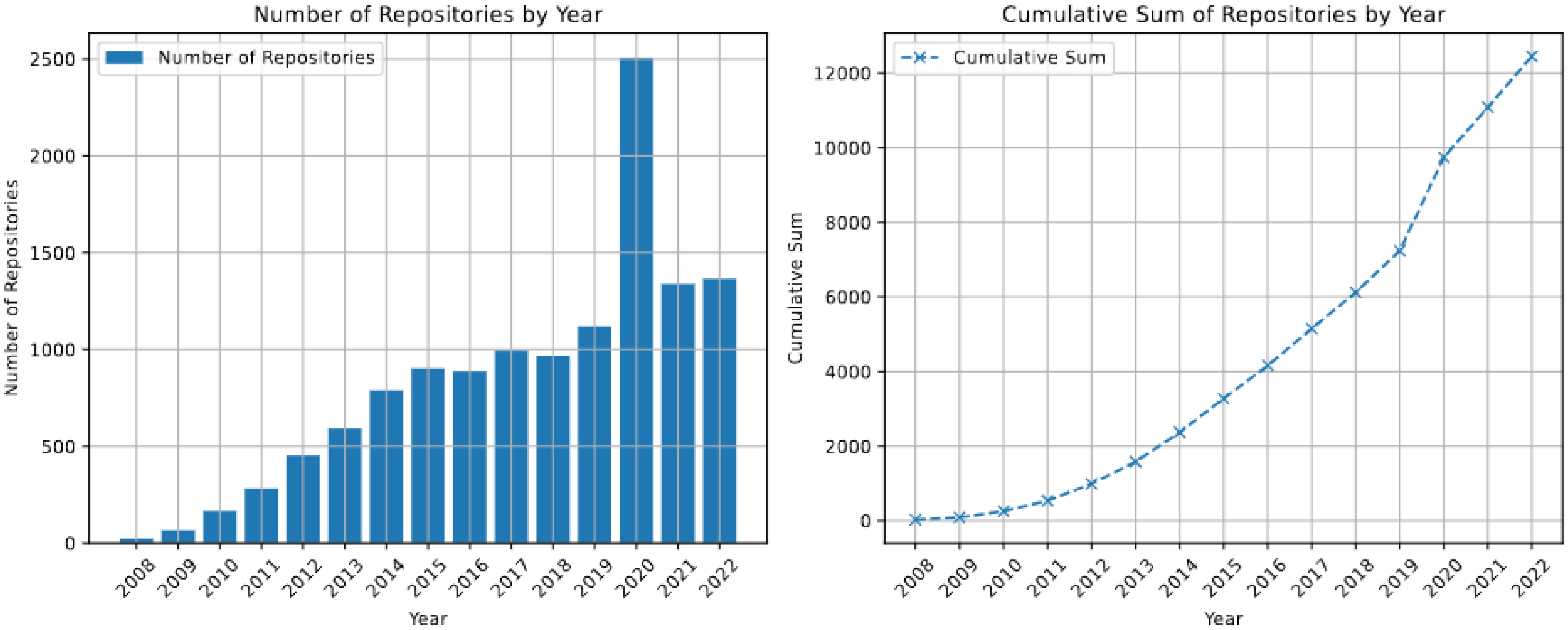
Template Engines - Popularity



Language	Template Engine	Popularity
Python	**Flask (Jinja2)**	64.1k ★
	Django	73k ★
	Mako	1.7k ★
	web2py	2.1k ★
	Tornado	21.3k ★
PHP	**Twig**	7.9k ★
	Smarty	2.1k ★
	Laravel (Blade)	74.6k ★
JavaScript	**Pug**	1.4M (NPM)
	Handlebars	13.4M (NPM)
	Vue	3M (NPM)
	EJS	13.3M (NPM)
Java	**Pebble**	1k ★
	Thymeleaf	2.6k ★

Popularity of template engines in terms of GitHub stars and NPM weekly downloads (JavaScript)

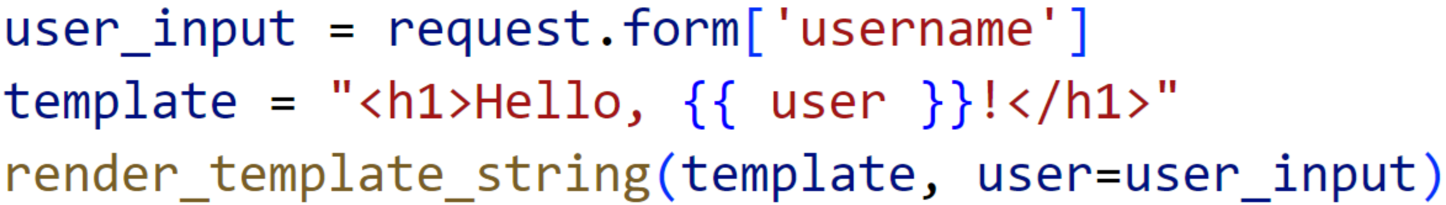
Template Engines



Number of repositories resulting from the query search “template engine” on GitHub

Template Engines - Secure Usage

In Jinja2 (Python) the following code renders a template



```
user_input = request.form['username']
template = "<h1>Hello, {{ user }}!</h1>"
render_template_string(template, user=user_input)
```

Example:

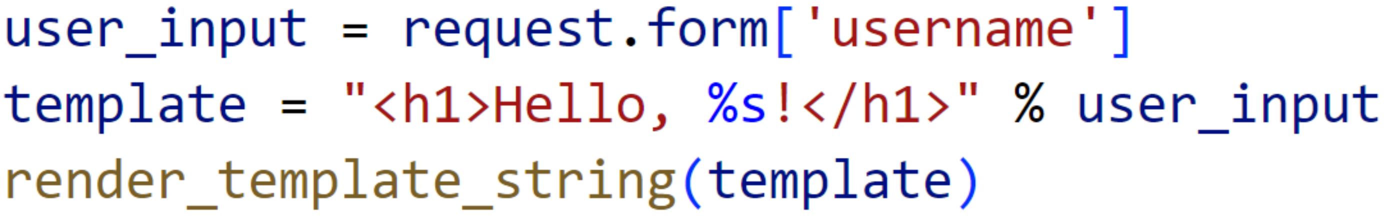
if **username=John** the output is **Welcome, John!**

if **username={{7*7}}** the output is **Welcome, {{7*7}}!**

Template Engines - Vulnerable Usage

The template is embedding directly the user input

This is dangerous since the user is now allowed to execute template directives



```
user_input = request.form['username']
template = "<h1>Hello, %s!</h1>" % user_input
render_template_string(template)
```

Example

If **username=John** the output is **Welcome, John!**

If **username={{7*7}}** the output is **Welcome, 49!**

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
SERICS

Server-Side Template Injection (SSTI)

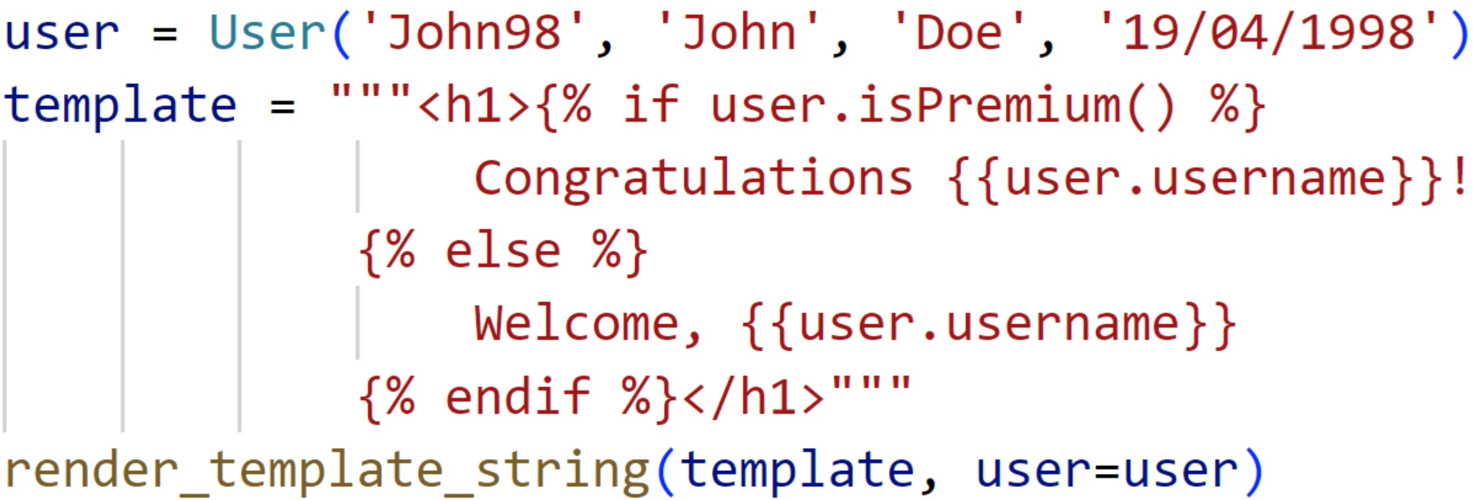
- Discovered in 2015, but possibly already present
- **Different types**, similarly to XSS and SQLi
  - Non-persistent
  - Persistent
  - Non-Blind
  - Blind
- Many possible consequences
  - **Sensitive data leaks**
  - **Unauthorized access**
  - **DoS attacks**
  - **Cross-Site Scripting**
  - Remote Code Execution

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS

SSTI to RCE - Why?

Template engines allow to perform seemingly innocent operations

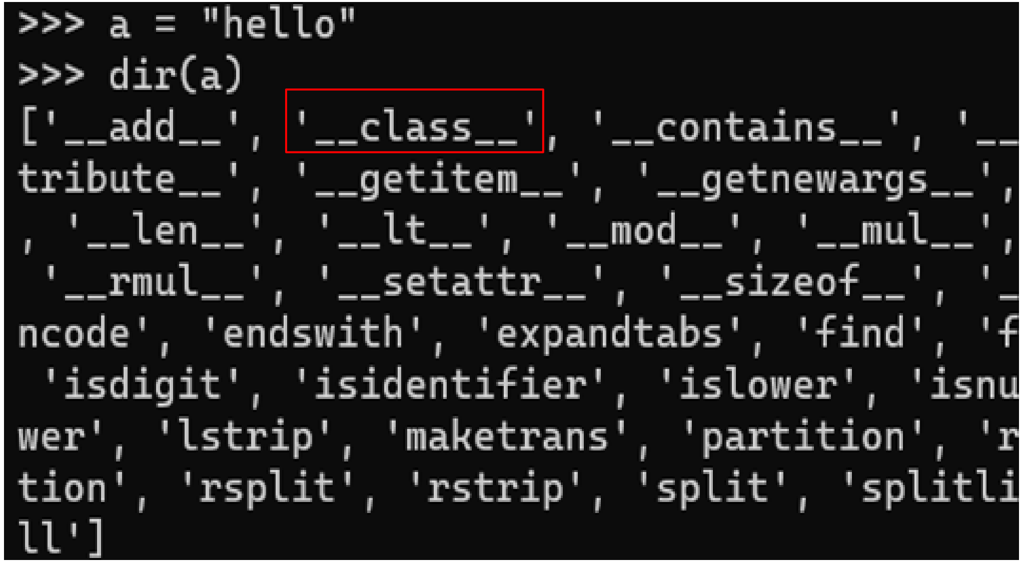
- **Access objects attributes**
- **Call objects functions**



```
user = User('John98', 'John', 'Doe', '19/04/1998')
template = """<h1>{% if user.isPremium() %}
                Congratulations {{user.username}}!
            {% else %}
                Welcome, {{user.username}}
            {% endif %}</h1>"""
render_template_string(template, user=user)
```

SSTI to RCE - Why?

- But they can be dangerous since **introspective** attributes and functions exist



```
>>> a = "hello"
>>> dir(a)
['__add__', '__class__', '__contains__', '__
tribute__', '__getitem__', '__getnewargs__',
, '__len__', '__lt__', '__mod__', '__mul__',
 '__rmul__', '__setattr__', '__sizeof__', '_
ncode', 'endswith', 'expandtabs', 'find', 'f
 'isdigit', 'isidentifier', 'islower', 'isnu
wer', 'lstrip', 'maketrans', 'partition', 'r
tion', 'rsplit', 'rstrip', 'split', 'splitli
ll']
```

SSTI to RCE

Since Jinja2 allows to access introspective attributes, users can inject the following payload to obtain RCE

```
{{config.__class__.__init__.__globals__[‘os’].popen(‘ls’).read()}}
```

Global Object Python Introspective Attributes OS Module Command Exec Output

“**config**” is a **Flask object** that contains configuration parameters

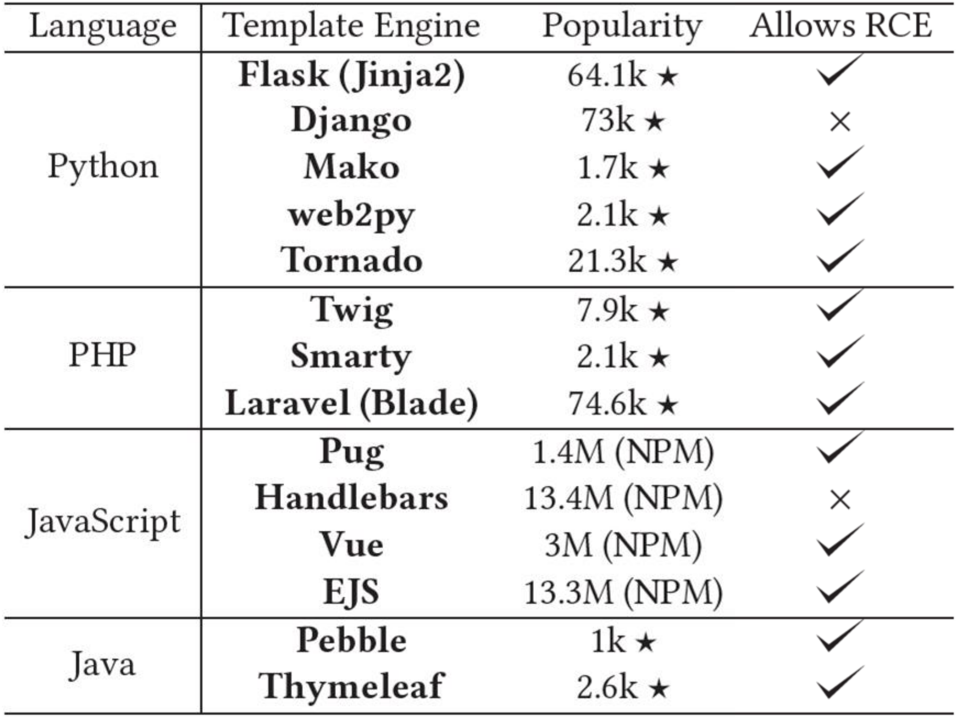
```
{{‘’.__class__.__mro__()[1].__subclasses__()[N](‘ls’, shell=True, stdout=-1)}}
```

Object Python Introspective Attributes Offset Command Execution

N is the **offset** where the ***subprocess.Popen*** class is located, it can change depending on the application

Template Engines - Popularity VS Security

Remember this table? Let’s add one more column



Language	Template Engine	Popularity	Allows RCE
Python	**Flask (Jinja2)**	64.1k ★	✓
	Django	73k ★	×
	Mako	1.7k ★	✓
	web2py	2.1k ★	✓
	Tornado	21.3k ★	✓
PHP	**Twig**	7.9k ★	✓
	Smarty	2.1k ★	✓
	Laravel (Blade)	74.6k ★	✓
JavaScript	**Pug**	1.4M (NPM)	✓
	Handlebars	13.4M (NPM)	×
	Vue	3M (NPM)	✓
	EJS	13.3M (NPM)	✓
Java	**Pebble**	1k ★	✓
	Thymeleaf	2.6k ★	✓

Finanziato dall'Unione europea NextGenerationEU

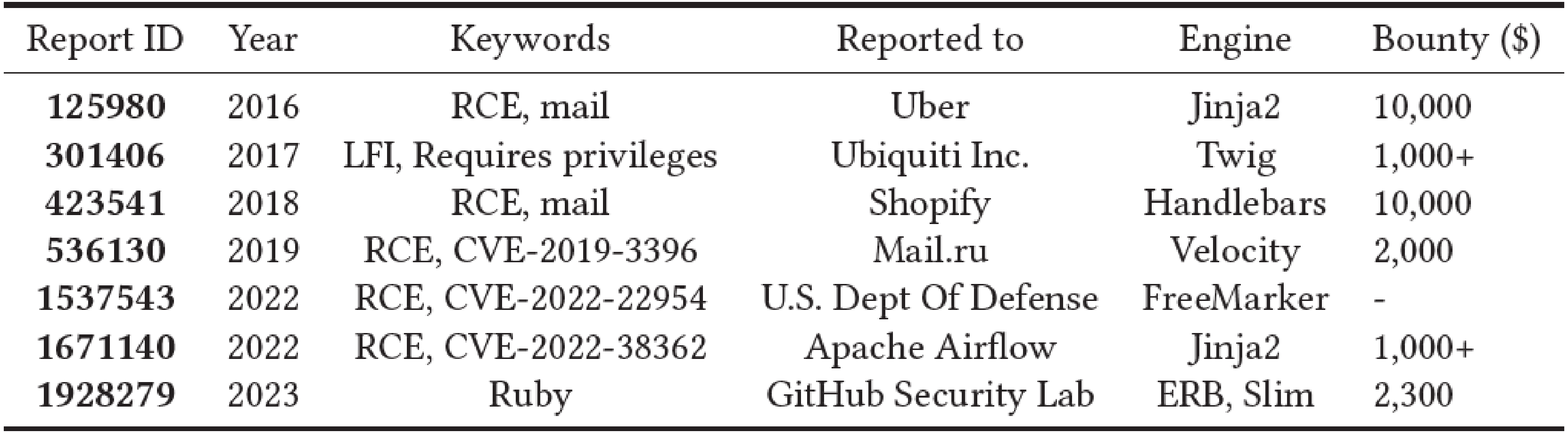
Ministero dell'Università e della Ricerca

Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

SERICS

SSTI - In the Wild

- Most of them lead to RCE
- The bounties can be very high
- Different engines involved



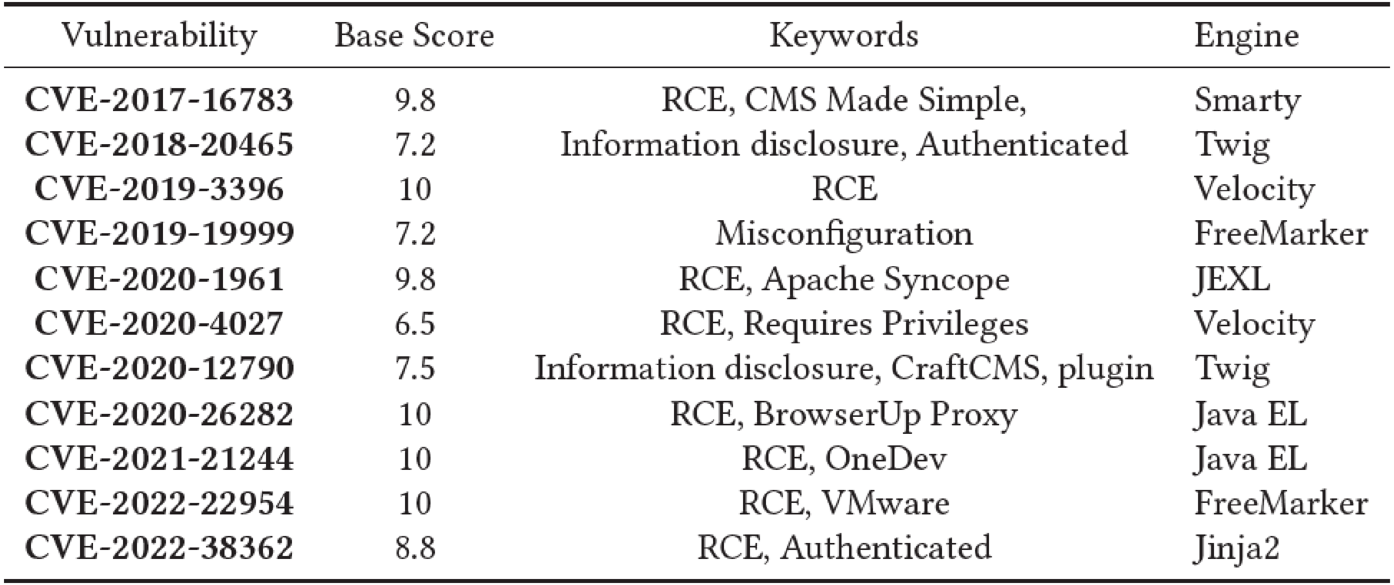
Report ID	Year	Keywords	Reported to	Engine	Bounty ($)
125980	2016	RCE, mail	Uber	Jinja2	10,000
301406	2017	LFI, Requires privileges	Ubiquiti Inc.	Twig	1,000+
423541	2018	RCE, mail	Shopify	Handlebars	10,000
536130	2019	RCE, CVE-2019-3396	Mail.ru	Velocity	2,000
1537543	2022	RCE, CVE-2022-22954	U.S. Dept Of Defense	FreeMarker	-
1671140	2022	RCE, CVE-2022-38362	Apache Airflow	Jinja2	1,000+
1928279	2023	Ruby	GitHub Security Lab	ERB, Slim	2,300

A list of SSTI reports on HackerOne

SSTI - CVEs

SSTI corresponds to CWE-1336 under the CWE-94 (Code Injection)

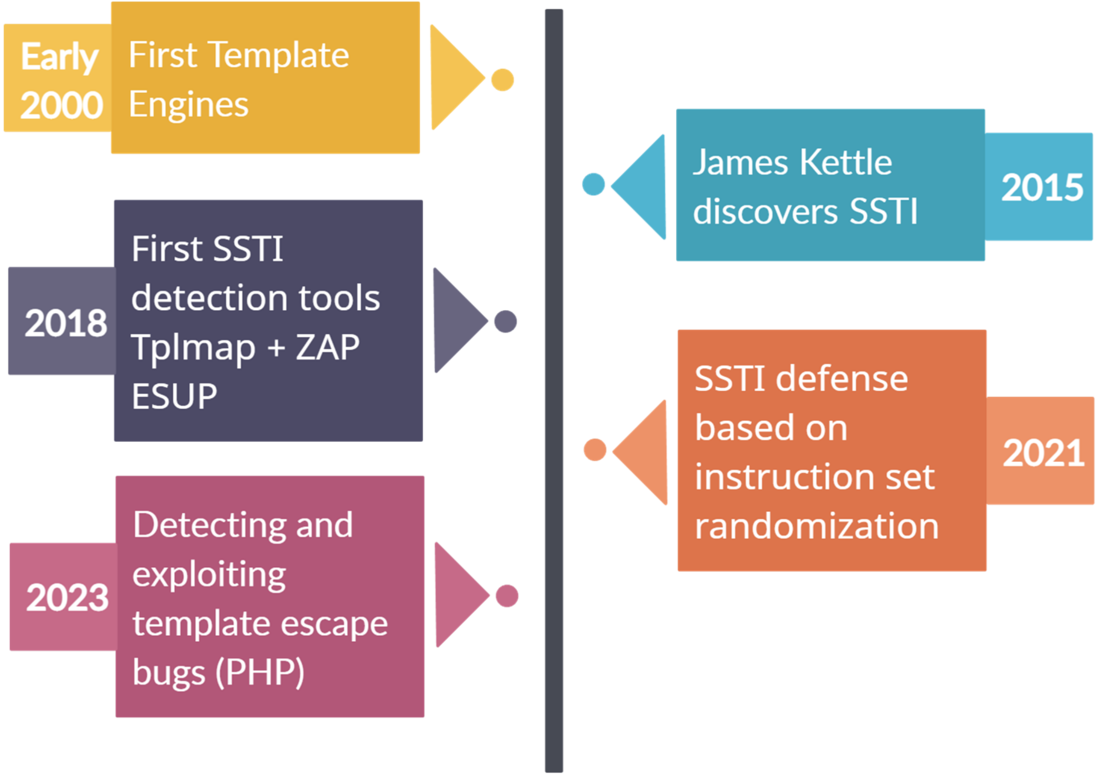
The base score of SSTI CVEs is very high on average and RCE is often present



Vulnerability	Base Score	Keywords	Engine
CVE-2017-16783	9.8	RCE, CMS Made Simple,	Smarty
CVE-2018-20465	7.2	Information disclosure, Authenticated	Twig
CVE-2019-3396	10	RCE	Velocity
CVE-2019-19999	7.2	Misconfiguration	FreeMarker
CVE-2020-1961	9.8	RCE, Apache Syncope	JEXL
CVE-2020-4027	6.5	RCE, Requires Privileges	Velocity
CVE-2020-12790	7.5	Information disclosure, CraftCMS, plugin	Twig
CVE-2020-26282	10	RCE, BrowserUp Proxy	Java EL
CVE-2021-21244	10	RCE, OneDev	Java EL
CVE-2022-22954	10	RCE, VMware	FreeMarker
CVE-2022-38362	8.8	RCE, Authenticated	Jinja2

A list of CVEs related to SSTI

SSTI - Seminal Works



Early 2000 First Template Engines

2015 James Kettle discovers SSTI

2018 First SSTI detection tools Tplmap + ZAP ESUP

2021 SSTI defense based on instruction set randomization

2023 Detecting and exploiting template escape bugs (PHP)

Template Engines - Scenarios

- **Unintentional**
  - The web developer introduces SSTI unintentionally
  - In this case avoiding SSTI is the main focus

- **Intentional**
  - CMS
  - Bulk emails
  - Website as a service (Github Pages)
  - Is essential to select a secure template engine

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS

Demo

Two examples

- A simple website with **unintentional** SSTI (Python Jinja2)
- A CMS with **intentional** SSTI (who uses the CMS should not be allowed to access the underlying machine - Python Jinja2 vs Django)

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS

The Importance of Selection

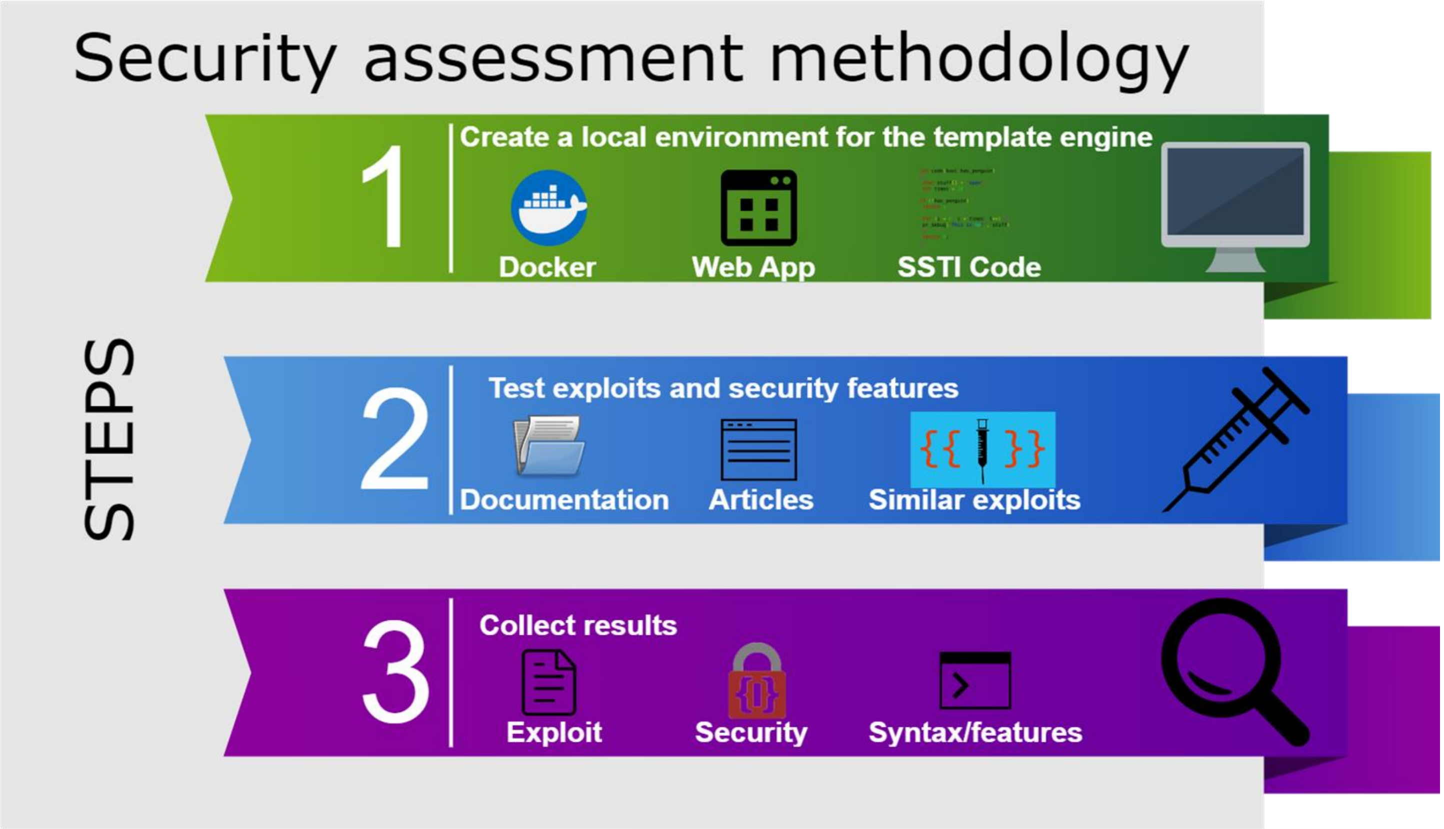
The demo showed how important is to select a template engine properly

- Some popular engines are known to allow RCE
- What if I’m using a template engine that is less popular or custom?

We need a general methodology to assess if a template engine allows RCE or not

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani | SERICS

Template Analysis



Security assessment methodology

STEPS

1 Create a local environment for the template engine
- Docker
- Web App
- SSTI Code

2 Test exploits and security features
- Documentation
- Articles
- Similar exploits

3 Collect results
- Exploit
- Security
- Syntax/features

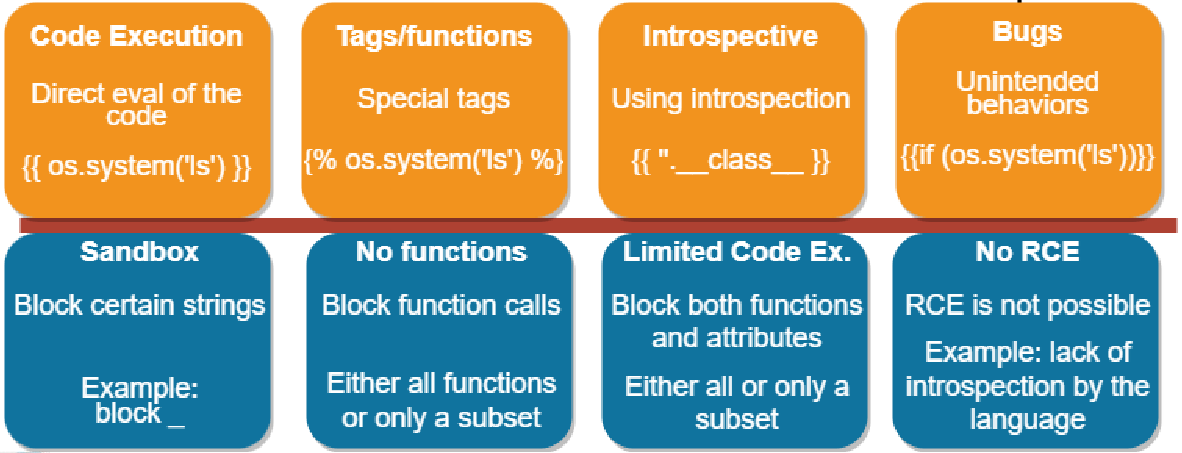
RCE Paths and Security Features

4 RCE exploit types

- Direct code execution
- Tags or functions for code execution
- Introspective
- Bugs or vulnerabilities

4 security features types

- Sandbox
- No function calls
- Limited code execution
- No RCE paths



Code Execution

Direct eval of the code

{{ os.system('ls') }}

Tags/functions

Special tags

{% os.system('ls') %}

Introspective

Using introspection

{{ ''.__class__ }}

Bugs

Unintended behaviors

{{if (os.system('ls'))}}

Sandbox

Block certain strings

Example: block _

No functions

Block function calls

Either all functions or only a subset

Limited Code Ex.

Block both functions and attributes

Either all or only a subset

No RCE

RCE is not possible

Example: lack of introspection by the language

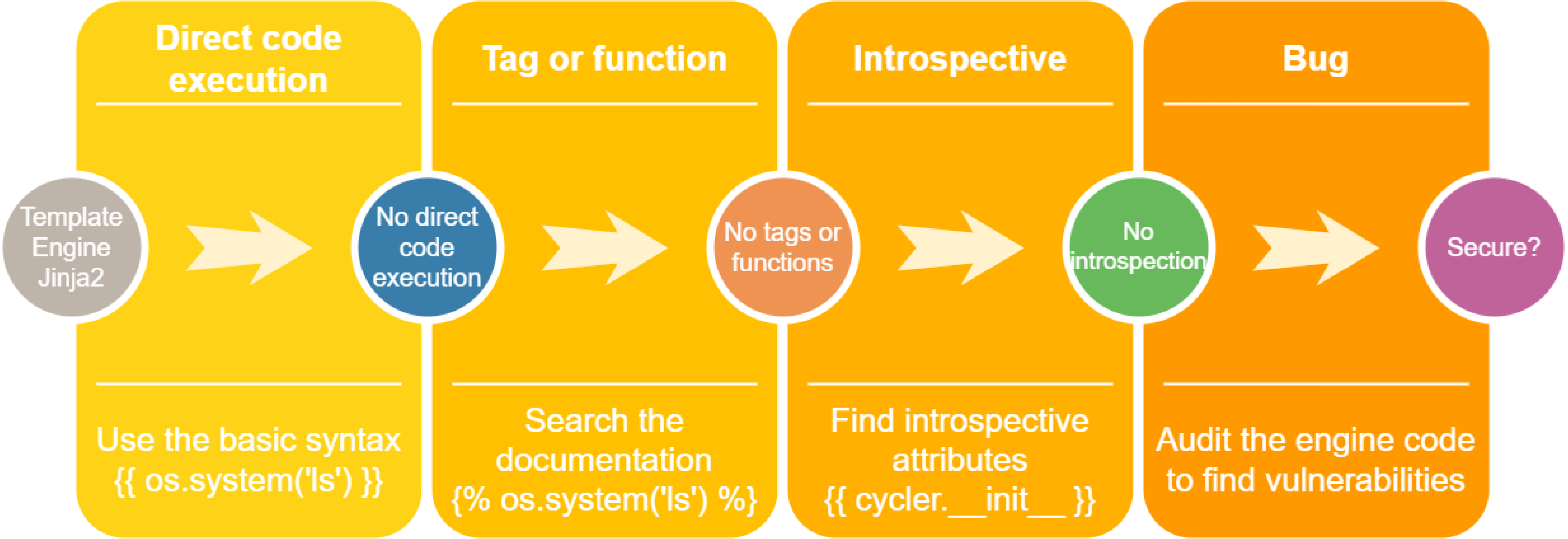
Making the Tests

- The time needed to find and test 34 engines was of 4 weeks
- Some tests could take up to 3-4 hours whilst others 2-3 days (Java was especially difficult)

Steps involved:

a. **Search** the template engine documentation/repo, it contains usage examples of the engine
b. **Write** an SSTI vulnerable piece of code
c. **Host** the web application/execute the vulnerable code
d. **Test** exploits and security

Finding an RCE Path - Jinja2 Example + Demo



Template Engine Jinja2

Direct code execution

Use the basic syntax {{ os.system('ls') }}

No direct code execution

Tag or function

Search the documentation {% os.system('ls') %}

No tags or functions

Introspective

Find introspective attributes {{ cycler.__init__ }}

No introspection

Bug

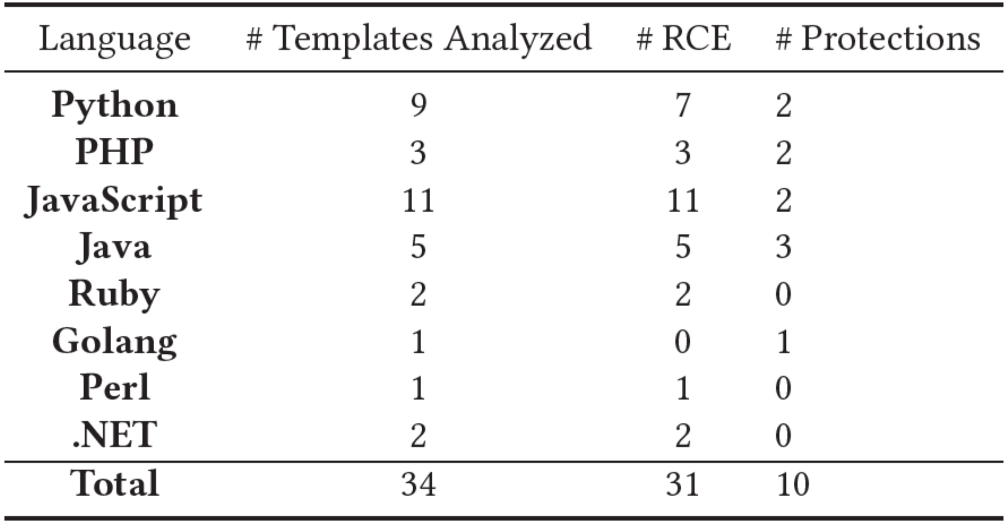
Audit the engine code to find vulnerabilities

Secure?

Finanziato dall'Unione europea NextGenerationEU Ministero dell'Università e della Ricerca Italiadomani SERICS

Results

- We analyzed 34 template engines in 8 different programming languages
  - 9 were never analyzed before and 8 allowed RCE
  - 31 allow or allowed RCE



Language	# Templates Analyzed	# RCE	# Protections
Python	9	7	2
PHP	3	3	2
JavaScript	11	11	2
Java	5	5	3
Ruby	2	2	0
Golang	1	0	1
Perl	1	1	0
.NET	2	2	0
Total	34	31	10

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS

Results - details

Language	Name	Delimiters	Already analyzed	Known RCE	RCE exploit	RCE Exploit kind	Security features
Python	Jinja2	{{ }}	✓	✓	✓	Introspective	-
	Cheetah	$ and #	✗	✗	✓	Tag for code execution	-
	Django	{{ }}	✓	✗	✗	-	Limited code exec.
	Genshi and Kid	${}	✗	✗	✓	Tag for code execution	-
	Mako	<% %> and $	✓	✓	✓	Tag for code execution	-
	web2py	{{= }}	✗	✗	✓	Introspective	-
	Tornado	{{ }} and {% %}	✓	✓	✓	Tag for code execution	-
	Chameleon	${ }	✗	✗	✓	Introspection	-
	Pyratemp	@! !@	✗	✗	✗	-	Sandbox

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
SERICS

The Future of SSTI and Template Engines

- Automatic ways to find RCE
  - Difficult, too many programming languages
- Developing solutions to mitigate RCE in template engines
  - Sandboxes can be escaped
- Developing template engines that do not allow RCE
  - Again, no sandboxes
  - Removing functions or attributes access has an impact
- Developing tools to detect SSTI that are not dependent on the engine

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS

Thank You!

lorenzo.pisu@unica.it

giacinto@unica.it

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERICS