SERVER-SIDE ATTACKS

Page 10 / Exploiting SSTI - Jinja2

Exploiting SSTI - Jinja2

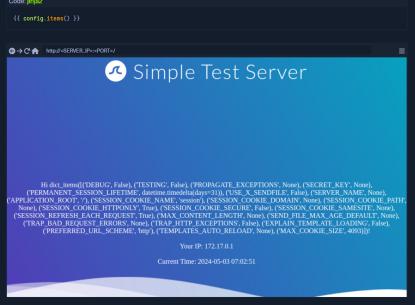
Now that we have seen how to identify the template engine used by a web application vulnerable to SSTI, we will move on to the exploitation of SSTI. In this section, we will assume that we have successfully identified that the web application uses the Jinja template engine. We will only focus on the SSTI exploitation and thus assume that the SSTI confirmation and template engine identification have already been done in a

Jinja is a template engine commonly used in Python web frameworks such as Flask or Django. This section will focus on a Flask web application. The payloads in other web frameworks might thus be slightly different.

In our payload, we can freely use any libraries that are already imported by the Python application, either directly or indirectly. Additionally, we may be able to import additional libraries through the use of the import statement.

Information Disclosure

We can exploit the SSTI vulnerability to obtain internal information about the web application, including configuration details and the web application's source code. For instance, we can obtain the web application's configuration using the following SSTI payload:



Since this payload dumps the entire web application configuration, including any used secret keys, we can prepare further attacks using the obtained information. We can also execute Python code to obtain information about the web application's source code. We can use the following SSTI payload to dump all available built-in functions:



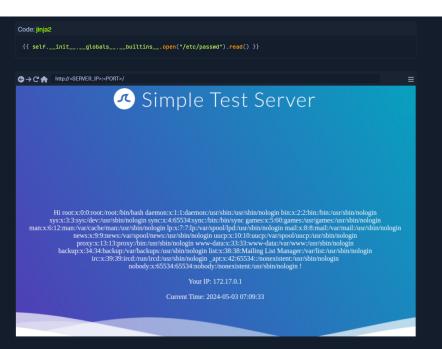
HI ('_name_': builtins', '_doc_': "Built-in functions, exceptions, and other objects.\n'nNoteworthy: None is the 'nil' object; Ellipsis represent
...' in slices.", '_package_': ", '_loader_': <class '_frozen_importlib.BuiltinImporter'>, '_spec_': ModuleSpec(name=builtins', loader=<clas
'_frozen_importlib.BuiltinImporter'>, '_spic_': Note | force |

Local File Inclusion (LFI)

We can use Python's built-in function open to include a local file. However, we cannot call the function directly; we need to call it from the __builtins__ dictionary we dumped earlier. This results in the following payload to include the file /etc/passwd:







Remote Code Execution (RCE)

To achieve remote code execution in Python, we can use functions provided by the os library, such as system or popen. However, if the web application has not already imported this library, we must first import it by calling the built-in function import. This results in the following SSTI payload:

