



**Penetration Testing Report**

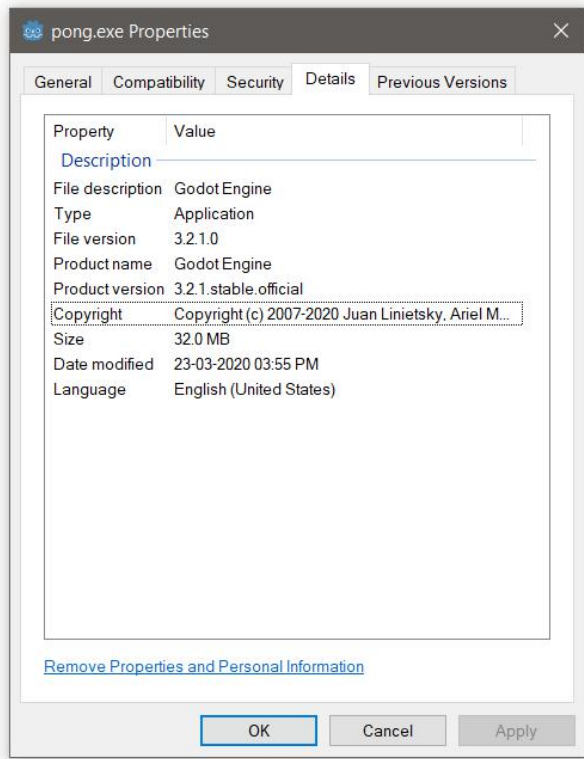
**For**

**“Pong”**

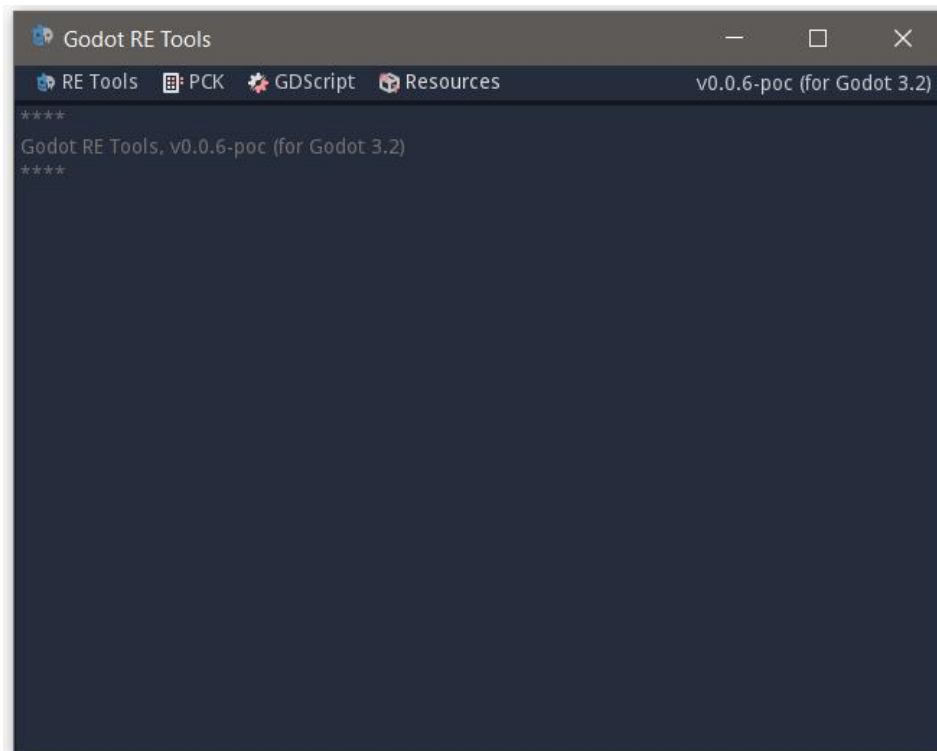
<b>S.NO.</b>	<b>Title</b>	<b>#</b>
1.	Challenge Category	Reverse Engineering
2.	Challenge Related Files	pong.exe
3.	File Link / Target IP	N/A

## PROCEDURE

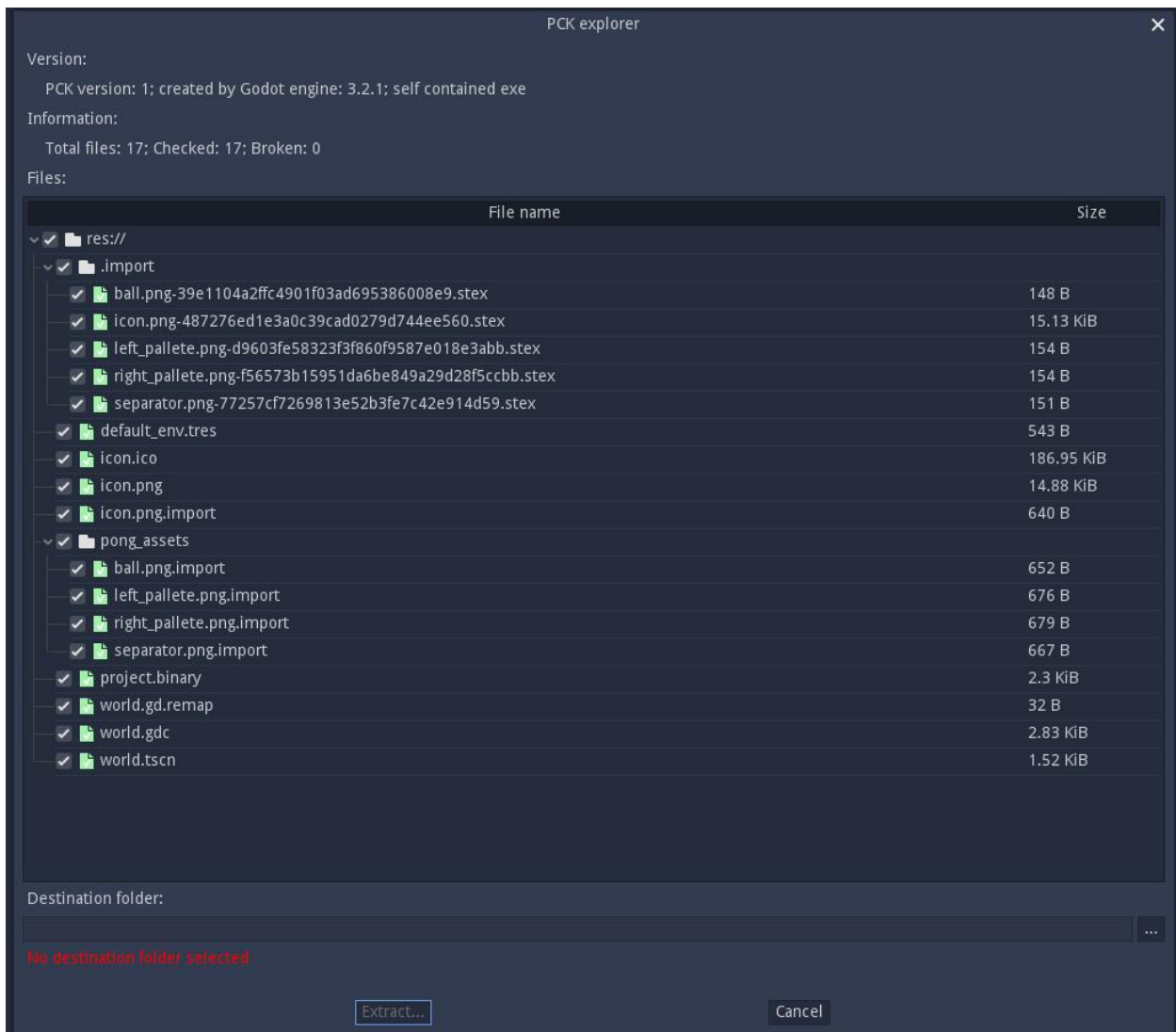
1. Properties of the file shows that the game is build on Godot 3.2.1.



2. Using the [Godot Reverse Engineering Tools](#).



### 3. Explore the PCK archive from pong.exe.



### 4. Decompile the world.gdc file to world.gd.

### 5. Analyze that GDScript in in Godot.

```
69 >|
70 >| if (leftSc == 147):
71 >| >| get_node("leftFlag").set_text("SEV7ZYzcnktYm")
72 >| if (rightSc == 1470):
73 >| >| get_node("rightFlag").set_text("FsbC1jMHVudHN9")
74 >| >|
```

## 6. Decoding the Base64.

The screenshot shows a web-based Base64 decoder. On the left, there is a 'Text' input field containing the Base64 string 'SEV7ZXYzcnktYmFsbC1jMHVudHNS'. In the center, there is a 'Base64' dropdown menu set to 'Base64 (RFC 3548, RFC 4648)'. Below the dropdown, it indicates 'Decoded 21 bytes'. On the right, there is a 'Text' output field containing the decoded string 'HE{ev3ry-ball-c0unts}'. The interface includes 'VIEW', 'ENCODE', and 'DECODE' buttons.

### Flags:

S.No.	Flag - No.	Flag
1.	Flag 1	HE{ev3ry-ball-c0unts}