Comment: There are a large number of bug injection positions in solidiFI that do not match the markings. For Example, the bug fragment in Fig.1 a starts on line 196 and ends on line 203, but it is marked in line 202. The bug fragment in Fig.1 b starts on line 222 and ends on line 229, but it is marked in line 217. The bug fragment in Fig.1 c starts on line 250 and ends on line 257, but it is marked in line 240……

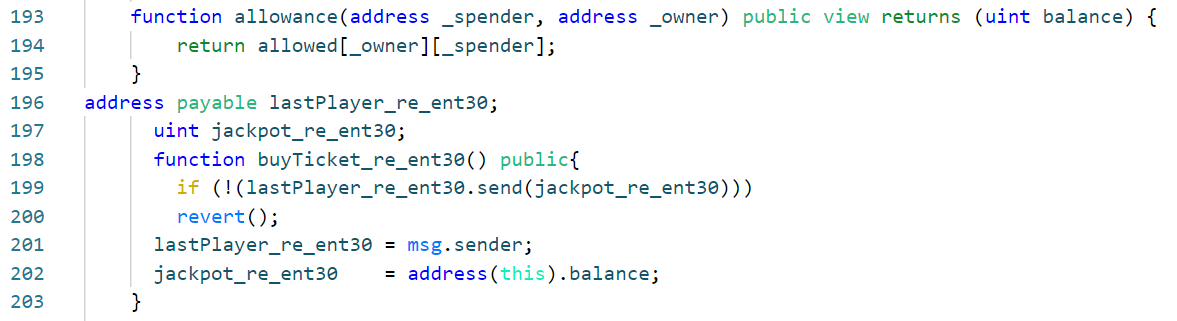


Fig.1 a: An example of incapability of SolidiFI to accurately inject and precisely label bugs.

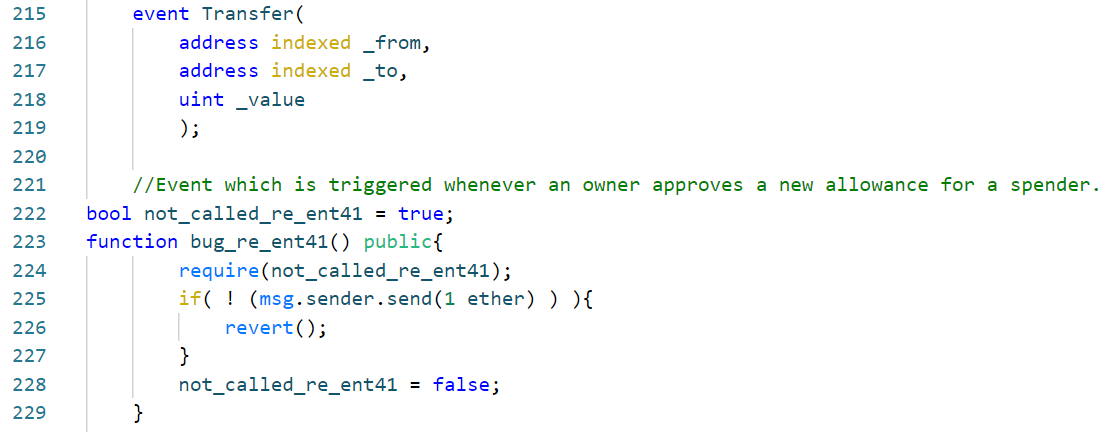


Fig.1 b: An example of incapability of SolidiFI to accurately inject and precisely label bugs.



Fig.1 c: An example of incapability of SolidiFI to accurately inject and precisely label bugs.

Comment: In addition to the unreasonable annotations mentioned above, there are also bug injection fragments with issues that prevent the injected bugs from being activated…..



Fig.2 a: An example of incapability of SolidiFI to accurately inject and precisely label bugs. SolidiFI does not insert any statement into the contract to modify the value of the variable *balances\_re\_ent31* (declared in line 1). This makes *balances\_re\_ent31[msg.sender]* keep the initial value (0) unchanged. If the *require-statement* (line 3) is to be true, \_*weiToWithdraw* must be equal to 0 at this point, which means the ether transferred by *msg.sender* is 0(line 5). However, transferring 0 ether is meaningless. It eventually invalidates the injected bug (line 5).



Fig.2 b: An example of incapability of SolidiFI to accurately inject and precisely label bugs. SolidiFI does not insert any statement into the contract to modify the value of the variable *balances\_re\_ent24* (declared in line 1). This makes *balances\_re\_ent24[msg.sender]* keep the initial value (0) unchanged. If the *require-statement* (line 3) is to be true, \_*weiToWithdraw* must be equal to 0 at this point, which means the ether transferred by *msg.sender* is 0(line 5). However, transferring 0 ether is meaningless. It eventually invalidates the injected bug (line 5).



Fig.2 c: An example of incapability of SolidiFI to accurately inject and precisely label bugs. SolidiFI does not insert any statement into the contract to modify the value of the variable *userBalance\_re\_ent26* (declared in line 1). This makes *userBalance\_re\_ent26 [msg.sender]* keep the initial value (0) unchanged, which means the ether transferred by *msg.sender* is 0(line 5). However, transferring 0 ether is meaningless. It eventually invalidates the injected bug (line 5).



Fig.2 d: An example of incapability of SolidiFI to accurately inject and precisely label bugs. SolidiFI does not insert any statement into the contract to modify the value of the variable *redeemableEther\_re\_ent25* (declared in line 1). This makes *redeemableEther\_re\_ent25 [msg.sender]* keep the initial value (0) unchanged and the *require-statement* (line 4) always throws an exception. It eventually invalidates the injected bug (line 8).