Arbitrary Jump with Function Type Variable

Description

Solidity supports function types. That is, a variable of function type can be assigned with a reference to a function with a matching signature. The function saved to such variable can be called just like a regular function.

The problem arises when a user can arbitrarily change the function type variable and thus execute random code instructions. As Solidity doesn't support pointer arithmetic's, it's impossible to change such variable to an arbitrary value. However, if the developer uses assembly instructions, such as `mstore` or assign operator, in the worst-case scenario an attacker is able to point a function type variable to any code instruction, violating required validations and required state changes.

Remediation

The use of assembly should be minimal. A developer should not allow a user to assign arbitrary values to function type variables.

Example:

Code:

```
pragma solidity ^0.4.25;

contract FunctionTypes {

    constructor() public payable { require(msg.value != 0); }

    function withdraw() private {
        require(msg.value == 0, 'dont send funds!');
        address(msg.sender).transfer(address(this).balance);
    }

    function frwd() internal
        { withdraw(); }

    struct Func { function () internal f; }

    function breakIt() public payable {
        require(msg.value != 0, 'send funds!');
        Func memory func;
        func.f = frwd;
        assembly { mstore(func, add(mload(func), callvalue)) }
        func.f();
    }
}

```

Explanation:

If the smart contract uses certain assembly instructions, mstore for example, an attacker may be able to point the function variable to any other function. This may give the attacker the ability to break the functionality of the contract, and perhaps even drain the contract funds.

Since inline assembly is a way to access the EVM at a low level, it bypasses many important safety features. So, it's important to only use assembly if it is necessary and properly understood.