Contract Audit
Smart Fast

Smart Contract
Security Audit Report

Num: 09061546236776

Date: 2022-09-06

Welcome to SmartFast!

0x01 Summary Information

The SmartFast (SF, for short) platform received this smart contract security audit application and audited the contract in September 2022.

It is necessary to declare that SF only issues this report in respect of facts that have occurred or existed before the issuance of this report, and undertakes corresponding responsibilities for this. For the facts that occur or exist in the future, SF is unable to judge the security status of its smart contract, and will not be responsible for it. The security audit analysis and other content made in this report are based on the documents and information provided to smart analysis team by the information provider as of the issuance of this report (referred to as "provided information"). SF hypothesis: There is no missing, tampered, deleted or concealed information in the mentioned information. If the information that has been mentioned is missing, tampered with, deleted, concealed or reflected does not match the actual situation, SmartFast shall not be liable for any losses and adverse effects caused thereby.

Table 1 Contract audit information

Project	Description
Contract name	Test
Contract type	Ethereum contract
Code language	Solidity
Contract files	arbitrary_send.sol
Contract address	
Auditors	SmartFast team
Audit time	2022-09-06 15:46:23
Audit tool	SmartFast (SF)

Table 1 shows the relevant information of this contract audit in detail. The details and results of the contract security audit will be introduced in detail below.

0x02 Contract Audit Results

2.1 Vulnerability Distribution

The severity of vulnerabilities in this security audit is distributed according to the level of danger:

Table 2 Overview of contract audit vulnerability distribution

Vulnerability security distribution				
High	Medium	Low	Info	Opt
4	13	2	24	15

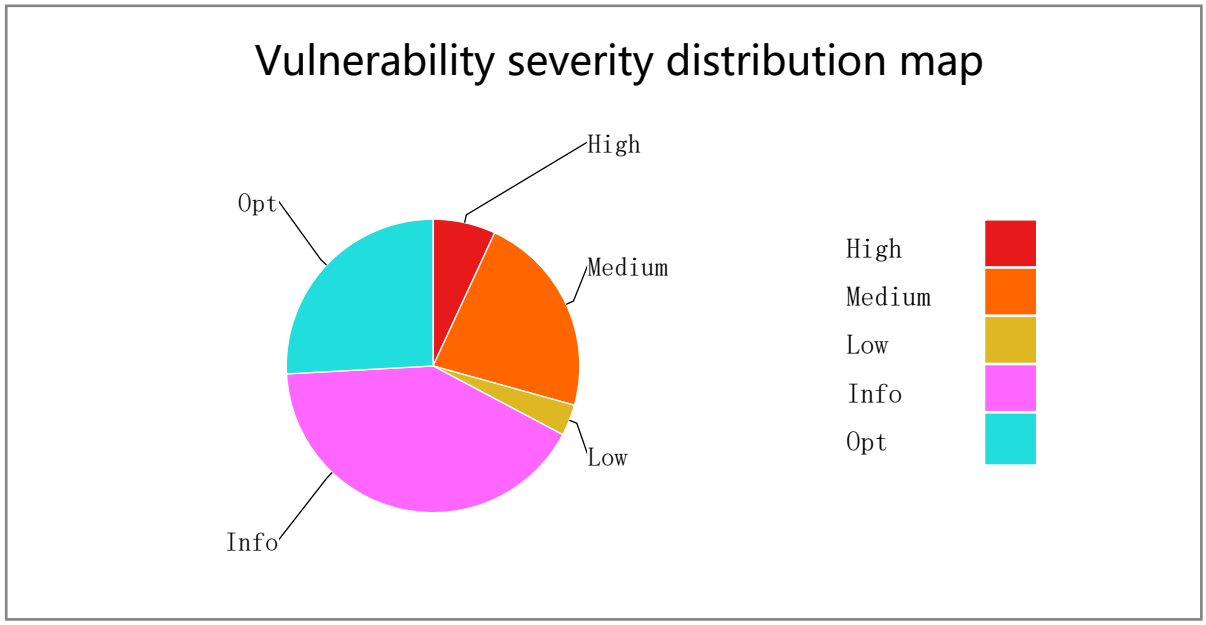

Figure 1 Vulnerability security distribution map

This security audit found 4 High-severity vulnerabilities, 13 Medium-severity vulnerabilities, 2 Low-severity vulnerabilities, 15 Optimization-severity vulnerabilities, and 24 places that need attention.

2.2 Audit Results

There are 119 test items in this security audit, and the test items are as follows (other unknown security vulnerabilities are not included in the scope of responsibility of this audit):

Table 3 Contract audit items

ID	Pattern	Description	Severity	Confidence	Status/Num
1	abiencoderv2-array	ABI encoding error	High	exactly	Pass
2	array-by-reference	Storage parameter usage	High	exactly	Pass
3	multiple-constructors	Multiple constructors in a contract	High	exactly	Pass

4	names-reused	Check contract name reuse	High	exactly	Pass
5	public-mappings-nested	Struct sets the nested structure of struct	High	exactly	Pass
6	rtlo	Overwritten from right to left	High	exactly	Pass
7	shadowing-state	Check state variable hiding	High	exactly	Pass
8	suicidal	Check if anyone can break the contract	High	exactly	Pass
9	uninitialized-state	Check for uninitialized state variables	High	exactly	Pass
10	uninitialized-storage	Check for uninitialized storage variables	High	exactly	Pass
11	unprotected-upgrade	Break the logic of the contract	High	exactly	Pass
12	visibility	Check visibility level error	High\Info	exactly	Info:1
13	redundant-fallback	Check for redundant fallback functions	High\Opt	exactly	Pass
14	arbitrary-send	Check if Ether can be sent to any address	High	probably	High:4
15	continue-in-loop	Continue causes an infinite loop	High	probably	Pass
16	controlled-array-length	Length is allocated directly	High	probably	Pass
17	controlled-delegatecall	The delegate address out of control	High	probably	Pass
18	incorrect-constructor	Check the constructor name error	High	probably	Pass
19	parity-multisig-bug	Check for multi-signature vulnerabilities	High	probably	Pass
20	reentrancy-eth	Re-entry vulnerabilities (Ethereum theft)	High	probably	Pass
21	storage-array	Signed integer array problem	High	probably	Pass
22	weak-prng	Modulo problem of parameters	High	probably	Pass
23	assert-violation	Check for incorrect use of assertions	Medium	exactly	Pass
24	constructor-return	The use of return in the constructor	Medium	exactly	Pass
25	enum-conversion	Scope of enumerated types	Medium	exactly	Pass
26	erc1155-interface	Check the wrong ERC1155 interface	Medium	exactly	Pass
27	erc1410-interface	Check the wrong ERC1410 interface	Medium	exactly	Pass
28	erc20-interface	Check for wrong ERC20 interface	Medium	exactly	Pass
29	erc223-interface	Check the wrong ERC223 interface	Medium	exactly	Pass
30	erc621-interface	Check the wrong ERC621 interface	Medium	exactly	Pass
31	erc721-interface	Check the wrong ERC721 interface	Medium	exactly	Pass
32	erc777-interface	Check the wrong ERC777 interface	Medium	exactly	Pass
33	erc875-interface	Check the wrong ERC875 interface	Medium	exactly	Pass
34	incorrect-equality	Check the strict equality of danger	Medium	exactly	Pass
35	incorrect-signature	Check for incorrect function signatures	Medium	exactly	Pass

36	locked-ether	Whether the contract ether is locked	Medium	exactly	Pass
37	mapping-deletion	Delete the mapped structure problem	Medium	exactly	Pass
38	shadowing-abstract	State variables from the abstract contract	Medium	exactly	Pass
39	tautology	Check for tautologies or contradictions	Medium	exactly	Pass
40	default-return-value	Function returns only default value	Medium	probably	Pass
41	boolean-cst	Check for misuse of Boolean constants	Medium	probably	Pass
42	constant-function-state	Check Constant function to change state	Medium	probably	Pass
43	divide-before-multiply	Imprecise order of arithmetic operations	Medium	probably	Pass
44	erc20-approve	ERC-20 advance attack (TOD)	Medium	probably	Pass
45	function-problem	Abnormal termination of contract function	Medium	probably	Pass
46	mul-var-len-arguments	Hash collisions with multiple variable	Medium	probably	Pass
47	reentrancy-no-eth	Re-entry vulnerabilities (no ether theft)	Medium	probably	Pass
48	reused-constructor	Constructor conflict problem in the contract	Medium	probably	Pass
49	tx-origin	Check the dangerous use of tx.origin	Medium	probably	Pass
50	typographical-error	Check for writing errors (=+)	Medium	probably	Pass
51	unchecked-lowlevel	Check for uncensored low-level calls	Medium	probably	Pass
52	unchecked-send	Check unreviewed send	Medium	probably	Medium:13
53	uninitialized-local	Check for uninitialized local variables	Medium	probably	Pass
54	unused-return	Check if there is an unused return value	Medium	probably	Pass
55	writeto-arbitrarystorage	It can be written to any storage location	Medium	probably	Pass
56	integer-overflow	Check for integer overflow	Medium	probably	Low:1
57	costly-loop	Check for too expensive loops	Medium	possibly	Pass
58	shift-parameter-mixup	Check reversible shift operation	Medium	possibly	Pass
59	shadowing-builtin	Check the hiding of built-in symbols	Low	exactly	Pass
60	shadowing-function	Check function hiding	Low	exactly	Pass
61	shadowing-local	Check local variable hiding	Low	exactly	Pass
62	transfer-to-zeroaddress	The withdrawal address is 0x0	Low	exactly	Pass
63	uninitialized-fptr-cst	Uninitialized pointers in constructor	Low	exactly	Pass
64	variable-scope	Check the declaration of variables	Low	exactly	Pass
65	void-cst	Calls to unimplemented constructors	Low	exactly	Pass
66	incorrect-modifier	Restore abnormal Modifier	Low	exactly	Pass
67	assemblycall-rewrite	Assemblycall covers the input	Low	probably	Pass

68	block-other-parameters	Hazardous use variables (block.number etc.)	Low	probably	Pass
69	calls-loop	Check the external call in the loop	Low	probably	Pass
70	events-access	The loss of key access control parameters	Low	probably	Pass
71	events-maths	The loss of key arithmetic parameters	Low	probably	Pass
72	extcodesize-invoke	Check Extcodesize call	Low	probably	Pass
73	fllback-outofgas	The fallback function is too complicated	Low	probably	Pass
74	incorrect-blockhash	Incorrect use of Blockhash function	Low	probably	Pass
75	incorrect-inheritance-order	The inherited variables conflict	Low	probably	Pass
76	missing-zero-check	Check the use of zero addresses	Low	probably	Low:1
77	reentrancy-benign	Reentrant vulnerabilities (continuous calls)	Low	probably	Pass
78	reentrancy-events	Reentrant vulnerabilities (events)	Low	probably	Pass
79	timestamp	The dangerous use of block.timestamp	Low	probably	Pass
80	signature-malleability	The signature contains an existing signature	Low	possibly	Pass
81	assembly	Unsafe use of assembly	Info	exactly	Pass
82	assert-state-change	Incorrect use of assert()	Info	exactly	Pass
83	delete-dynamic-arrays	The deletion of the dynamic storage array	Info	exactly	Pass
84	deprecated-standards	Solidity deprecated instructions	Info	exactly	Pass
85	erc20-indexed	ERC20 event parameter is missing indexed	Info	exactly	Pass
86	erc20-throw	ERC20 throws an exception	Info	exactly	Pass
87	length-manipulation	Unsafe operation to check array length	Info	exactly	Pass
88	low-level-calls	Check low-level calls	Info	exactly	Info:13
89	msgvalue-equals-zero	The judgment of msg.value and zero	Info	exactly	Pass
90	naming-convention	The naming follows the Solidity format	Info	exactly	Info:9
91	pragma	Undeclared multiple compiled versions	Info	exactly	Pass
92	solc-version	Check for incorrect Solidity version	Info	exactly	Info:1
93	unimplemented-functions	Functions overloaded in the contract	Info	exactly	Pass
94	upgrade-050	Code for Solidity 0.5.x upgrade	Info	exactly	Pass
95	function-init-state	State variables initialized by functions	Info	exactly	Pass
96	complex-function	Check complex functions	Info	probably	Pass
97	hardcoded	Check the legitimacy of the address	Info	probably	Pass
98	overpowered-role	The permissions are too concentrated	Info	probably	Pass
99	reentrancy-limited-events	Reentrancy vulnerabilities (limited events)	Info	probably	Pass

100	reentrancy-limited-gas	Reentry (send and transfer, with eth)	Info	probably	Pass
101	reentrancy-limited-gas-no-eth	Reentrance (send and transfer, no eth)	Info	probably	Pass
102	similar-names	Detect similar variables	Info	probably	Pass
103	too-many-digits	Too many number symbols	Info	probably	Pass
104	private-not-hidedata	Check the use of private visibility	Info	possibly	Pass
105	safemath	Check the use of SafeMath	Info	possibly	Pass
106	array-instead-bytes	The byte array can be replaced with bytes	Opt	exactly	Pass
107	boolean-equal	Check comparison with boolean constant	Opt	exactly	Pass
108	code-no-effects	Check for invalid codes	Opt	exactly	Pass
109	constable-states	State variables can be declared as constants	Opt	exactly	Pass
110	event-before-revert	Check if event is called before revert	Opt	exactly	Pass
111	external-function	Public functions can be declared as external	Opt	exactly	Opt:15
112	extra-gas-inloops	Check for additional gas consumption	Opt	exactly	Pass
113	missing-inheritance	Detect lost inheritance	Opt	exactly	Pass
114	redundant-statements	Detect the use of invalid sentences	Opt	exactly	Pass
115	return-struct	Multiple return values (struct)	Opt	exactly	Pass
116	revert-require	Check Revert in if operation	Opt	exactly	Pass
117	send-transfer	Check Transfe to replace Send	Opt	exactly	Pass
118	unused-state	Check unused state variables	Opt	exactly	Pass
119	costly-operations-loop	Expensive operations in the loop	Opt	probably	Pass

0x03 Contract Code

3.1 Code and Vulnerability Lables

In the corresponding position of each contract code, security vulnerabilities and coding specification issues have been marked in the form of comments. The comment labels start with //StFt. For details, please refer to the following contract code content.

```
contract Test{

  address destination;
  address owner;

  mapping (address => uint) balances;

  constructor(){ //StAs //visibility
    balances[msg.sender] = 0;
```

```
owner = msg.sender;
}

function direct() public{ //bad //StAs //external-function
    msg.sender.send(address(this).balance); //StAs //arbitrary-send、unchecked-send、low-level-calls
}

function init() public{ //StAs //external-function
    destination = msg.sender;
}

modifier isowner() {
address aa = msg.sender;
require(owner == aa);
_;
}

function indirect() public{ //bad //StAs //external-function
    destination.send(address(this).balance); //StAs //arbitrary-send、unchecked-send、low-level-calls
}

function directceshi_1() public{ //StAs //naming-convention、external-function
require(owner == msg.sender);
    destination.send(address(this).balance); //StAs //unchecked-send、low-level-calls
}

function directceshi_2() isowner public{ //StAs //naming-convention、external-function
    destination.send(address(this).balance); //StAs //unchecked-send、low-level-calls
}

function directceshi_3() public{ //StAs //naming-convention、external-function
    destination.send(msg.value); //StAs //unchecked-send、low-level-calls
}

function directceshi_4() public{ //StAs //naming-convention、external-function
    destination.send(balances[msg.sender]); //StAs //unchecked-send、low-level-calls
}

function directceshi_5() public{ //StAs //naming-convention、external-function
    destination.send(0); //StAs //unchecked-send、low-level-calls
}

function directceshi_6() public{ //StAs //naming-convention、external-function
uint avalue = 0;
    destination.send(avalue); //StAs //unchecked-send、low-level-calls
}

// these are legitimate calls
// and should not be detected
function repay() payable public{ //StAs //external-function
    msg.sender.transfer(msg.value);
}

function withdraw() public{ //StAs //external-function
    uint val = balances[msg.sender];
    msg.sender.send(val); //StAs //unchecked-send、low-level-calls
```

```
    }

    function withdraw_direct() public{ //StAs //naming-convention、external-function
        uint val = msg.value;
        msg.sender.send(val); //StAs //unchecked-send、low-level-calls
    }

    function withdraw_inderect() public{ //bad //StAs //naming-convention、external-function
        uint val = address(this).balance;
        msg.sender.send(val); //StAs //arbitrary-send、unchecked-send、low-level-calls
    }

    function indirect_ceshi1() public{ //bad //StAs //naming-convention、external-function
    address cc = msg.sender;
    cc = owner;
    require(cc == owner);
    address dd = msg.sender; //StAs //missing-zero-check
        uint val = address(this).balance;
        dd.send(val); //StAs //arbitrary-send、unchecked-send、low-level-calls
    }

    function buy() payable public{ //StAs //external-function
        uint value_send = msg.value;
        uint value_spent = 0 ; // simulate a buy of tokens
        uint remaining = value_send - value_spent; //StAs //integer-overflow
        msg.sender.send(remaining); //StAs //unchecked-send、low-level-calls
    }

}
```

0x04 Contract Audit Details

4.1 visibility

Vulnerability description

The default function visibility level in the contract is public, and in interfaces-external, the default visibility level of state variables is internal. In the contract, the fallback function can be external or public. In the interface, all functions should be declared as external functions. Clearly define function visibility to prevent confusion. Specifically, before version 0.5.0: look for fallback functions without explicit visibility declaration and delete them, look for fallback functions that are neither external nor public, look for constructors with external or private visibility, in the interface Find internal and private functions; after version 0.5.0: Find non-external functions in the interface.

Audit results: 【Info:1】

For this pattern, the specific problems in the contract are as follows:

(Informational)constructor(){ (tests/arbitrary_send.sol)#8

Security advice

The visibility should be modified according to the requirements of the compiler.

4.2 arbitrary-send

Vulnerability description

The call to the function that sends Ether to an arbitrary address has not been reviewed.

Audit results: 【High:4】

For this pattern, the specific problems in the contract are as follows:

(High)Test.direct() (tests/arbitrary_send.sol#13-15) sends eth to arbitrary user

Dangerous calls:

- msg.sender.send(address(this).balance) (tests/arbitrary_send.sol#14)

(High)Test.indirect() (tests/arbitrary_send.sol#27-29) sends eth to arbitrary user

Dangerous calls:

- destination.send(address(this).balance) (tests/arbitrary_send.sol#28)

(High)Test.withdraw_inderect() (tests/arbitrary_send.sol#73-76) sends eth to arbitrary user

Dangerous calls:

- msg.sender.send(val) (tests/arbitrary_send.sol#75)

(High)Test.indirect_ceshi1() (tests/arbitrary_send.sol#78-85) sends eth to arbitrary user

Dangerous calls:

- dd.send(val) (tests/arbitrary_send.sol#84)

Security advice

Ensure that no user can withdraw unauthorized funds.

4.3 unchecked-send

Vulnerability description

Similar to unchecked-lowlevel, it is explained here that the return value of send and Highlevelcall is not checked.

Audit results:【Medium:13】

For this pattern, the specific problems in the contract are as follows:

(Medium)Test.buy() (tests/arbitrary_send.sol#87-92) unchecks send return value:

- msg.sender.send(remaining) (tests/arbitrary_send.sol#91)

(Medium)Test.direct() (tests/arbitrary_send.sol#13-15) unchecks send return value:

- msg.sender.send(address(this).balance) (tests/arbitrary_send.sol#14)

(Medium)Test.directceshi_1() (tests/arbitrary_send.sol#31-34) unchecks send return value:

- destination.send(address(this).balance) (tests/arbitrary_send.sol#33)

(Medium)Test.directceshi_2() (tests/arbitrary_send.sol#36-38) unchecks send return value:

- destination.send(address(this).balance) (tests/arbitrary_send.sol#37)

(Medium)Test.directceshi_3() (tests/arbitrary_send.sol#40-42) unchecks send return value:

- destination.send(msg.value) (tests/arbitrary_send.sol#41)

(Medium)Test.directceshi_4() (tests/arbitrary_send.sol#44-46) unchecks send return value:

- destination.send(balances[msg.sender]) (tests/arbitrary_send.sol#45)

(Medium)Test.directceshi_5() (tests/arbitrary_send.sol#48-50) unchecks send return value:

- destination.send(0) (tests/arbitrary_send.sol#49)

(Medium)Test.directceshi_6() (tests/arbitrary_send.sol#52-55) unchecks send return value:

- destination.send(avalue) (tests/arbitrary_send.sol#54)

(Medium)Test.indirect() (tests/arbitrary_send.sol#27-29) unchecks send return value:

- destination.send(address(this).balance) (tests/arbitrary_send.sol#28)

(Medium)Test.indirect_ceshi1() (tests/arbitrary_send.sol#78-85) unchecks send return value:

- dd.send(val) (tests/arbitrary_send.sol#84)

(Medium)Test.withdraw() (tests/arbitrary_send.sol#63-66) unchecks send return value:

- msg.sender.send(val) (tests/arbitrary_send.sol#65)

(Medium)Test.withdraw_direct() (tests/arbitrary_send.sol#68-71) unchecks send return value:

- msg.sender.send(val) (tests/arbitrary_send.sol#70)

(Medium)Test.withdraw_inderect() (tests/arbitrary_send.sol#73-76) unchecks send return value:

- msg.sender.send(val) (tests/arbitrary_send.sol#75)

Security advice

Make sure to check or record the return value of send.

4.4 integer-overflow

Vulnerability description

When an arithmetic operation reaches the maximum or minimum size of the type, overflow/underflow will occur. For example, if a number is stored in the uint8 type, it means that the number is stored as an 8-bit unsigned number, ranging from 0 to 2^8-1. In computer programming, when an arithmetic operation attempts to create a value, an integer overflow occurs, and the value can be represented by a given number of bits-greater than the maximum value or less than the minimum value.

Audit results: 【Low:1】

For this pattern, the specific problems in the contract are as follows:

(Low)Test.buy() (tests/arbitrary_send.sol#87-92) may have integer overflow

Possible nodes:

- remaining = value_send - value_spent (tests/arbitrary_send.sol#90)

Security advice

Use Safemath to perform integer arithmetic or verify calculated values.

4.5 missing-zero-check

Vulnerability description

Check the verification of the zero address.

Audit results: 【Low:1】

For this pattern, the specific problems in the contract are as follows:

(Low)Test.indirect_ceshi1().dd (tests/arbitrary_send.sol#82) lacks a zero-check on :

- dd.send(val) (tests/arbitrary_send.sol#84)

Security advice

Check that the address is not zero.

4.6 low-level-calls

Vulnerability description

Label low-level methods such as call, delegatecall, and callcode, because these methods are easily exploited by attackers.

Audit results: 【Info:13】

For this pattern, the specific problems in the contract are as follows:

(Informational)Low level call in Test.direct() (tests/arbitrary_send.sol#13-15):

- msg.sender.send(address(this).balance) (tests/arbitrary_send.sol#14)

(Informational)Low level call in Test.indirect() (tests/arbitrary_send.sol#27-29):

- destination.send(address(this).balance) (tests/arbitrary_send.sol#28)

(Informational)Low level call in Test.directceshi_1() (tests/arbitrary_send.sol#31-34):

- destination.send(address(this).balance) (tests/arbitrary_send.sol#33)

(Informational)Low level call in Test.directceshi_2() (tests/arbitrary_send.sol#36-38):

- destination.send(address(this).balance) (tests/arbitrary_send.sol#37)

(Informational)Low level call in Test.directceshi_3() (tests/arbitrary_send.sol#40-42):

- destination.send(msg.value) (tests/arbitrary_send.sol#41)

(Informational)Low level call in Test.directceshi_4() (tests/arbitrary_send.sol#44-46):

- destination.send(balances[msg.sender]) (tests/arbitrary_send.sol#45)

(Informational)Low level call in Test.directceshi_5() (tests/arbitrary_send.sol#48-50):

- destination.send(0) (tests/arbitrary_send.sol#49)

(Informational)Low level call in Test.directceshi_6() (tests/arbitrary_send.sol#52-55):

- destination.send(avalue) (tests/arbitrary_send.sol#54)

(Informational)Low level call in Test.withdraw() (tests/arbitrary_send.sol#63-66):

- msg.sender.send(val) (tests/arbitrary_send.sol#65)

(Informational)Low level call in Test.withdraw_direct() (tests/arbitrary_send.sol#68-71):

- msg.sender.send(val) (tests/arbitrary_send.sol#70)

(Informational)Low level call in Test.withdraw_inderect() (tests/arbitrary_send.sol#73-76):

- msg.sender.send(val) (tests/arbitrary_send.sol#75)

(Informational)Low level call in Test.indirect_ceshi1() (tests/arbitrary_send.sol#78-85):

- dd.send(val) (tests/arbitrary_send.sol#84)

(Informational)Low level call in Test.buy() (tests/arbitrary_send.sol#87-92):

- msg.sender.send(remaining) (tests/arbitrary_send.sol#91)

Security advice

Avoid low-level calls. Check whether the call is successful. If the call is to sign a contract, please check whether the code exists.

4.7 naming-convention

Vulnerability description

Check whether the naming written in the contract is standardized, because the naming is messy and not easy to understand and manage.

Audit results: 【Info:9】

For this pattern, the specific problems in the contract are as follows:

(Informational)Function Test.directceshi_1() (tests/arbitrary_send.sol#31-34) is not in mixedCase

(Informational)Function Test.directceshi_2() (tests/arbitrary_send.sol#36-38) is not in mixedCase

(Informational)Function Test.directceshi_3() (tests/arbitrary_send.sol#40-42) is not in mixedCase

(Informational)Function Test.directceshi_4() (tests/arbitrary_send.sol#44-46) is not in mixedCase

(Informational)Function Test.directceshi_5() (tests/arbitrary_send.sol#48-50) is not in mixedCase

(Informational)Function Test.directceshi_6() (tests/arbitrary_send.sol#52-55) is not in mixedCase

(Informational)Function Test.withdraw_direct() (tests/arbitrary_send.sol#68-71) is not in mixedCase

(Informational)Function Test.withdraw_inderect() (tests/arbitrary_send.sol#73-76) is not in mixedCase

(Informational)Function Test.indirect_ceshi1() (tests/arbitrary_send.sol#78-85) is not in mixedCase

Security advice

Please follow Solidity [naming conventions](https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions).

4.8 solc-version

Vulnerability description

Solidity source files indicate the version of the compiler that can be compiled with them. It is recommended to indicate a clear version, because future versions of the compiler may handle certain language constructs in ways that developers cannot foresee.

Audit results: 【Info:1】

For this pattern, the specific problems in the contract are as follows:

(Informational)solc-0.4.24 is not recommended for deployment

Security advice

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing. Solidity version recommended: 0.5.11-0.5.13, 0.5.15-0.5.17, 0.6.8, 0.6.10-0.6.11.

4.9 external-function

Vulnerability description

Functions with public visibility modifiers are not called internally. Changing the visibility level to an external level can improve the readability of the code. In addition, in many cases, functions that use external visibility modifiers cost less gas than functions that use public visibility modifiers.

Audit results: 【Opt:15】

For this pattern, the specific problems in the contract are as follows:

(Optimization)direct() should be declared external:
- Test.direct() (tests/arbitrary_send.sol#13-15)

(Optimization)init() should be declared external:
- Test.init() (tests/arbitrary_send.sol#17-19)

(Optimization)indirect() should be declared external:
- Test.indirect() (tests/arbitrary_send.sol#27-29)

(Optimization)directceshi_1() should be declared external:
- Test.directceshi_1() (tests/arbitrary_send.sol#31-34)

(Optimization)directceshi_2() should be declared external:
- Test.directceshi_2() (tests/arbitrary_send.sol#36-38)

(Optimization)directceshi_3() should be declared external:
- Test.directceshi_3() (tests/arbitrary_send.sol#40-42)

(Optimization)directceshi_4() should be declared external:
- Test.directceshi_4() (tests/arbitrary_send.sol#44-46)

(Optimization)directceshi_5() should be declared external:
- Test.directceshi_5() (tests/arbitrary_send.sol#48-50)

(Optimization)directceshi_6() should be declared external:
- Test.directceshi_6() (tests/arbitrary_send.sol#52-55)

(Optimization)repay() should be declared external:
- Test.repay() (tests/arbitrary_send.sol#59-61)

(Optimization)withdraw() should be declared external:
- Test.withdraw() (tests/arbitrary_send.sol#63-66)

(Optimization)withdraw_direct() should be declared external:
- Test.withdraw_direct() (tests/arbitrary_send.sol#68-71)

(Optimization)withdraw_inderect() should be declared external:

- Test.withdraw_inderect() (tests/arbitrary_send.sol#73-76)

(Optimization)indirect_ceshi1() should be declared external:

- Test.indirect_ceshi1() (tests/arbitrary_send.sol#78-85)

(Optimization)buy() should be declared external:

- Test.buy() (tests/arbitrary_send.sol#87-92)

Security advice

Use the "external" attribute for functions that are never called from within the contract.

Thanks for using the SmartFast contract audit platform!