VETEOS: Statically Vetting EOSIO Contracts for the "Groundhog Day" Vulnerabilities

Levi Taiji Li[1], Ningyu He[2], Haoyu Wang[3], Mu Zhang[1]

[1]University of Utah

[2]Peking University

[3]Huazhong University of Science and Technology

IMAGINE U

EOSIO Smart Contracts

358545893

Current Block

3996 Peak

Live TPS

Total EOS Wallets

6,145,869

↑ 43,502 / 30 days

0.71%

Staked / Unstaked

257,749,759 EOS

Staked

36,431 EOS

Voters

98,396 voters

151,296,190 EOS

(13% of supply)

RAM usage

261.5 GB / 390.2 GB

RAM price per KB:
0.1023 EOS / kb ($0.08 / kb)

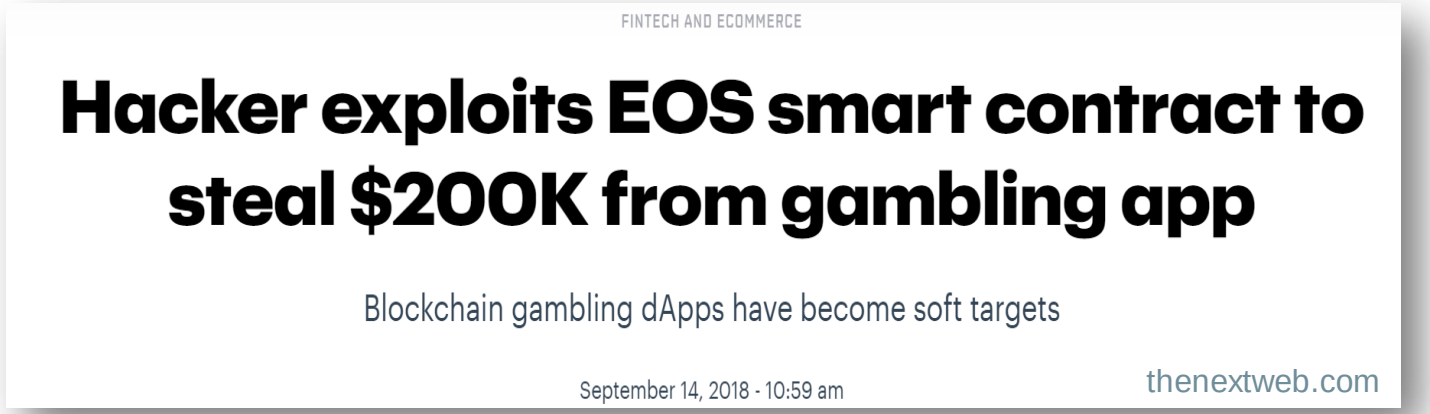
CPU scaling

5% used now

CPU price:
0.0001 EOS/ms/day
($0.00 USD/ms/day)

IMAGINE U

Attack Events



FINTECH AND ECOMMERCE

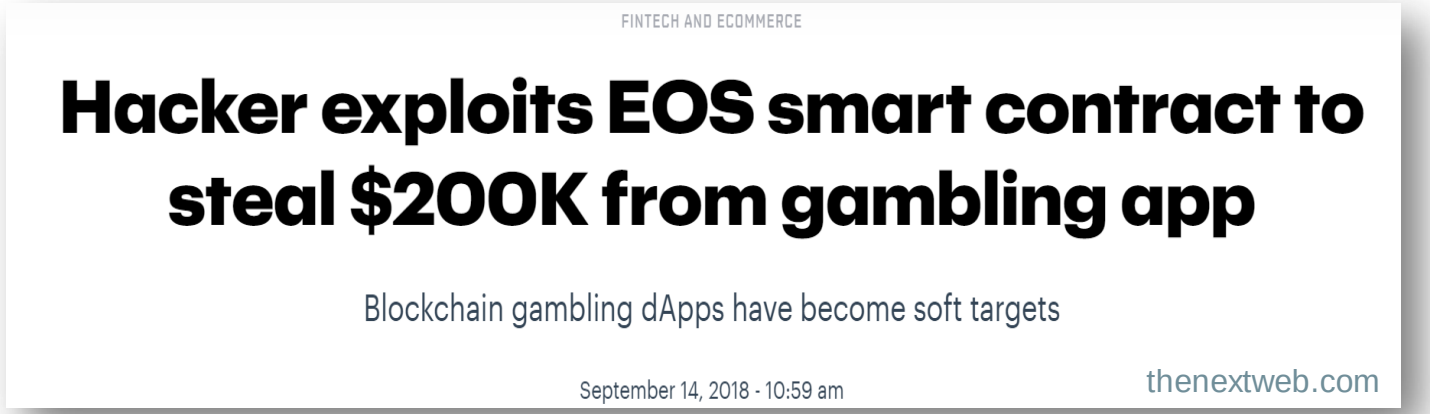
Hacker exploits EOS smart contract to steal $200K from gambling app

Blockchain gambling dApps have become soft targets

September 14, 2018 - 10:59 am

thenextweb.com

Attack Events

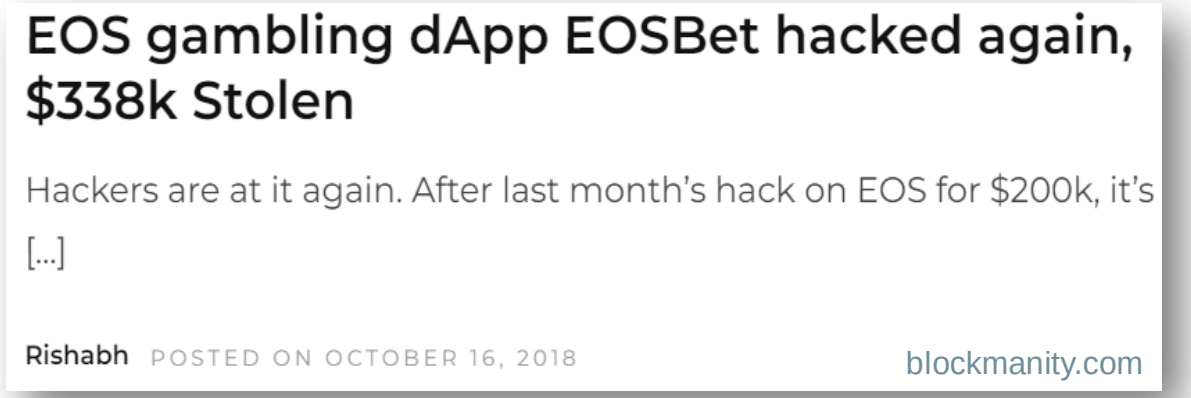

FINTECH AND ECOMMERCE

Hacker exploits EOS smart contract to steal $200K from gambling app

Blockchain gambling dApps have become soft targets

September 14, 2018 - 10:59 am

thenextweb.com


EOS gambling dApp EOSBet hacked again, $338k Stolen

Hackers are at it again. After last month's hack on EOS for $200k, it's [...]

Rishabh POSTED ON OCTOBER 16, 2018

blockmanity.com

Attack Events

FINTECH AND ECOMMERCE

Hacker exploits EOS smart contract to steal $200K from gambling app

Blockchain gambling dApps have become soft targets

September 14, 2018 - 10:59 am

thenextweb.com

EOS gambling dApp EOSBet hacked again, $338k Stolen

Hackers are at it again. After last month's hack on EOS for $200k, it's [...]

Rishabh POSTED ON OCTOBER 16, 2018

blockmanity.com

Hacked target: EOS.WIN 2018-11-11

Description of the event: The game contract was attacked by the attacker lockonthecha.

Amount of loss: 20,000 EOS **Attack method:** Random number attack

View Reference Sources

Hacked target: EOS.Win 2018-09-14

Description of the event: The attacker exchanged true EOS token with fake token within the vulnerability in the code,winning without betting

Amount of loss: 4,000 EOS **Attack method:** Code Vulnerability

Hacked target: EOS.Win 2018-09-02

Description of the event: On September 2, 2018, the EOS WIN random number was cracked and 2000 EOS was lost, this attack was not disclosed to the public.

Amount of loss: 2,000 EOS **Attack method:** Random number attack

View Reference Sources

hacked.slowmist.io

IMAGINE U

Attack Events

FINTECH AND ECOMMERCE

Hacker exploits EOS smart contract to s

[SlowMist Hacked Statistical]:

The total amount of money lost by blockchain hackers is about $ 27,927,302.55 ;

EOS ga
$338k Stolen

Hackers are at it again. After last month's hack on EOS for $200k, it's [...]

Rishabh POSTED ON OCTOBER 16, 2018

blockmanity.com

Hacked target: EOS.WIN 2018-11-11

Description of the event: The game contract was attacked by the attacker lockonthecha.

Amount of loss: 20,000 EOS **Attack method:** Random number attack

2018-09-14

th fake

Hacked target: EOS.WIN 2018-09-02

Description of the event: On September 2, 2018, the EOS WIN random number was cracked and 2000 EOS was lost, this attack was not disclosed to the public.

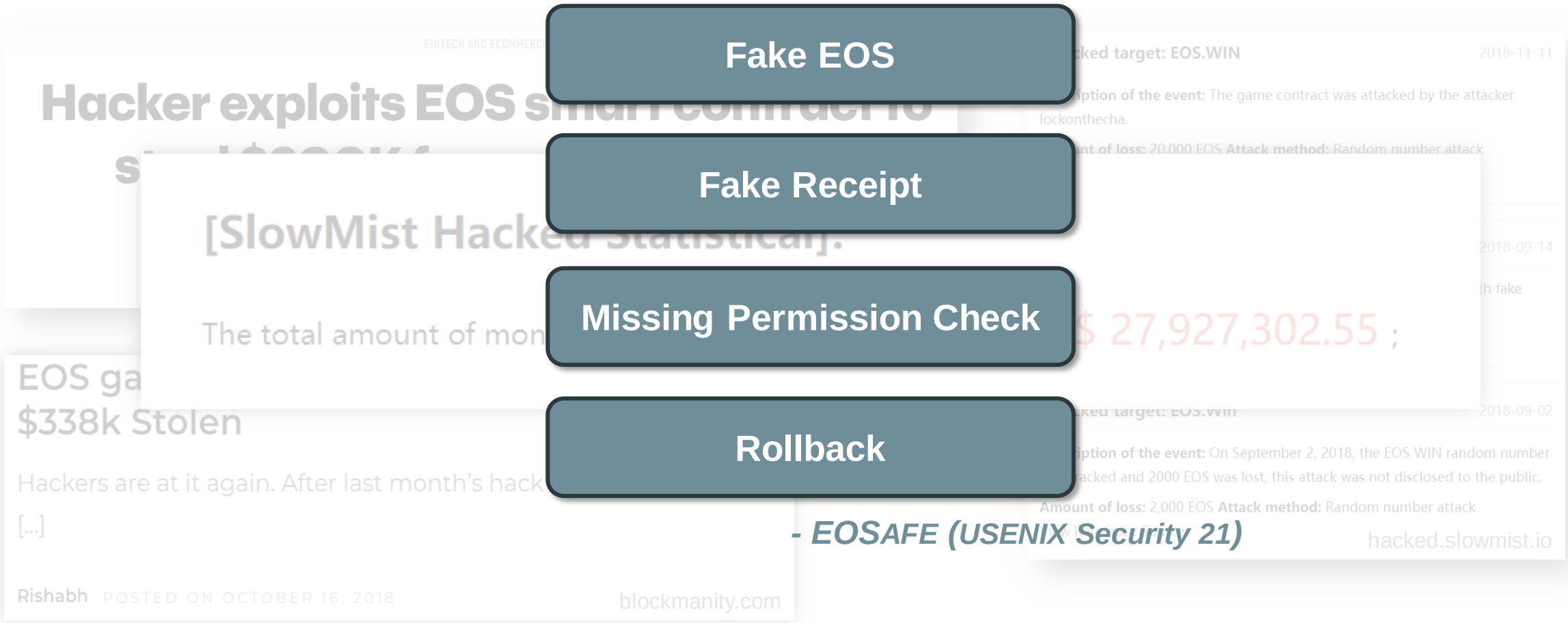
Amount of loss: 2,000 EOS **Attack method:** Random number attack

View Reference Sources

hacked.slowmist.io

IMAGINE U

Vulnerabilities



Vulnerabilities

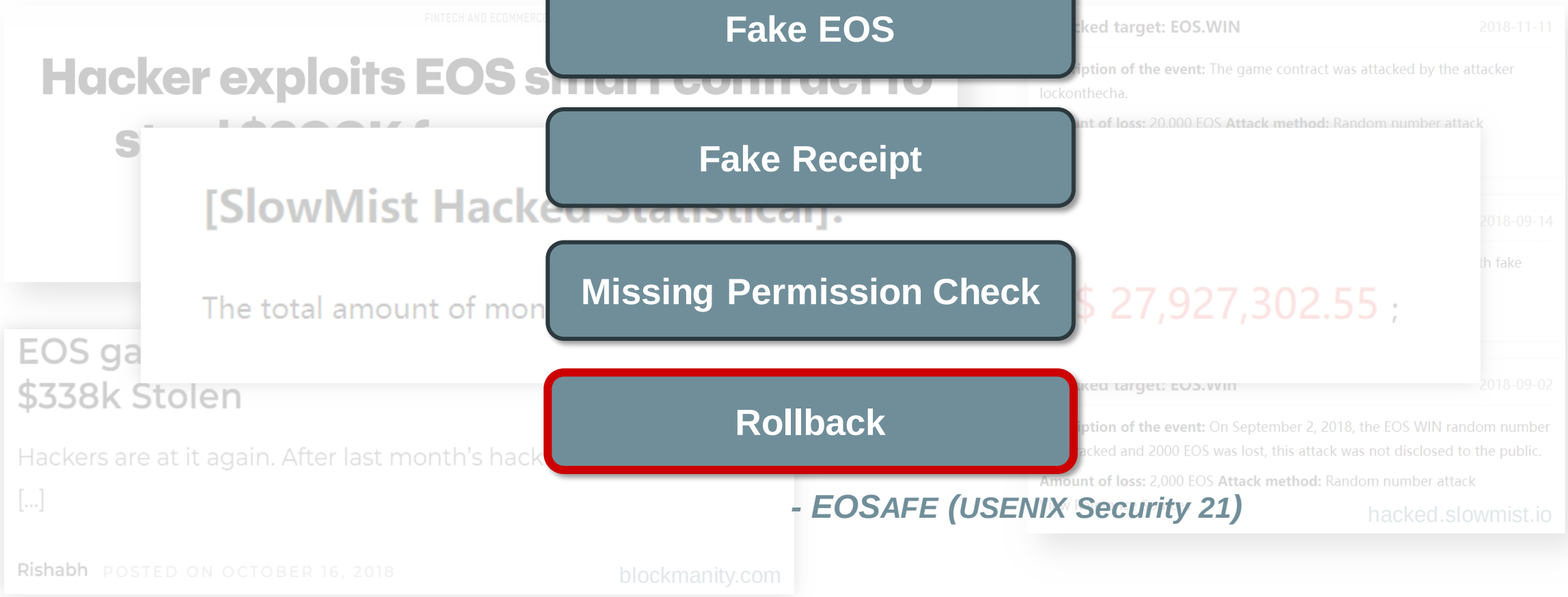
Fake EOS

Fake Receipt

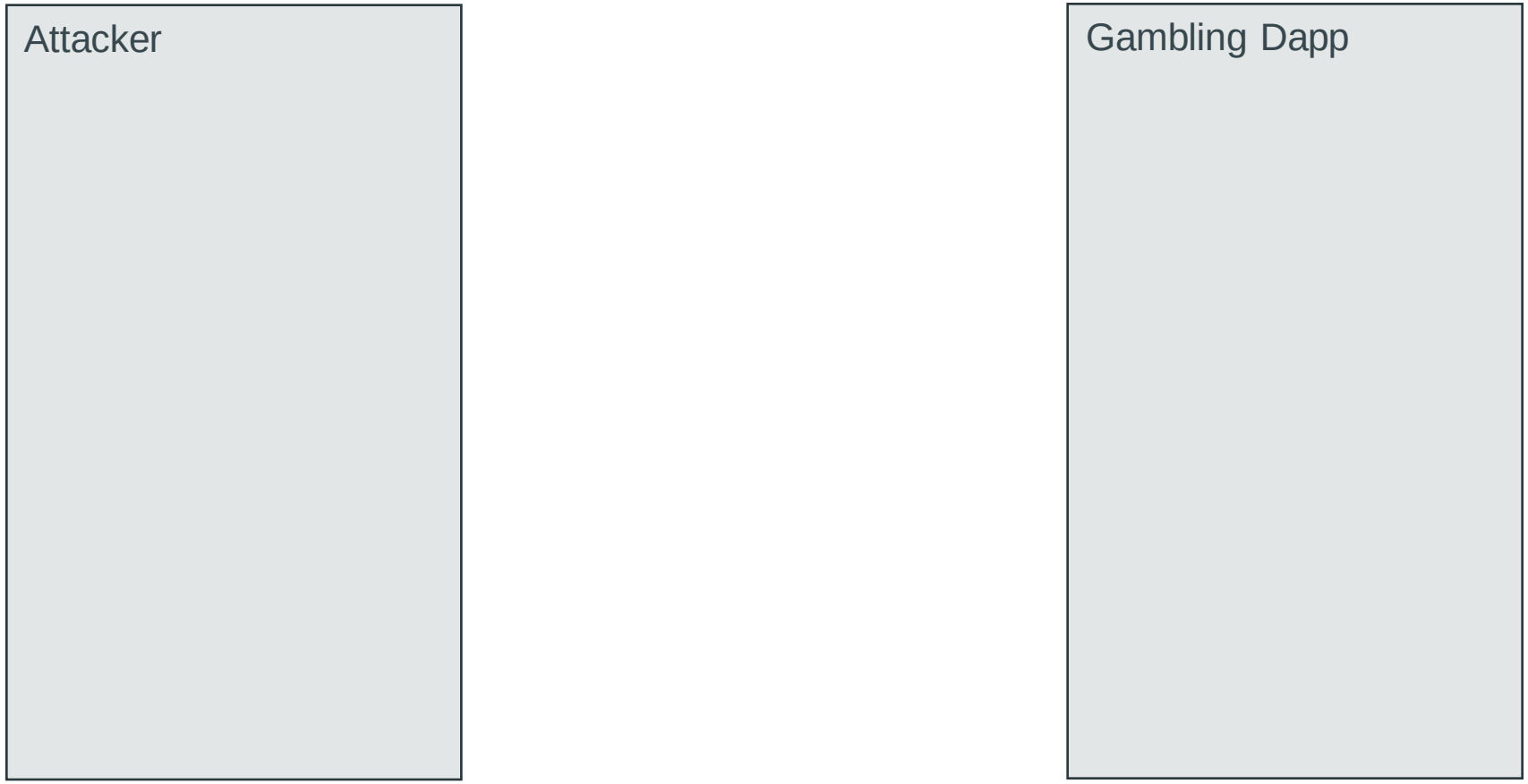
Missing Permission Check

Rollback

- EOSAFE (USENIX Security 21)

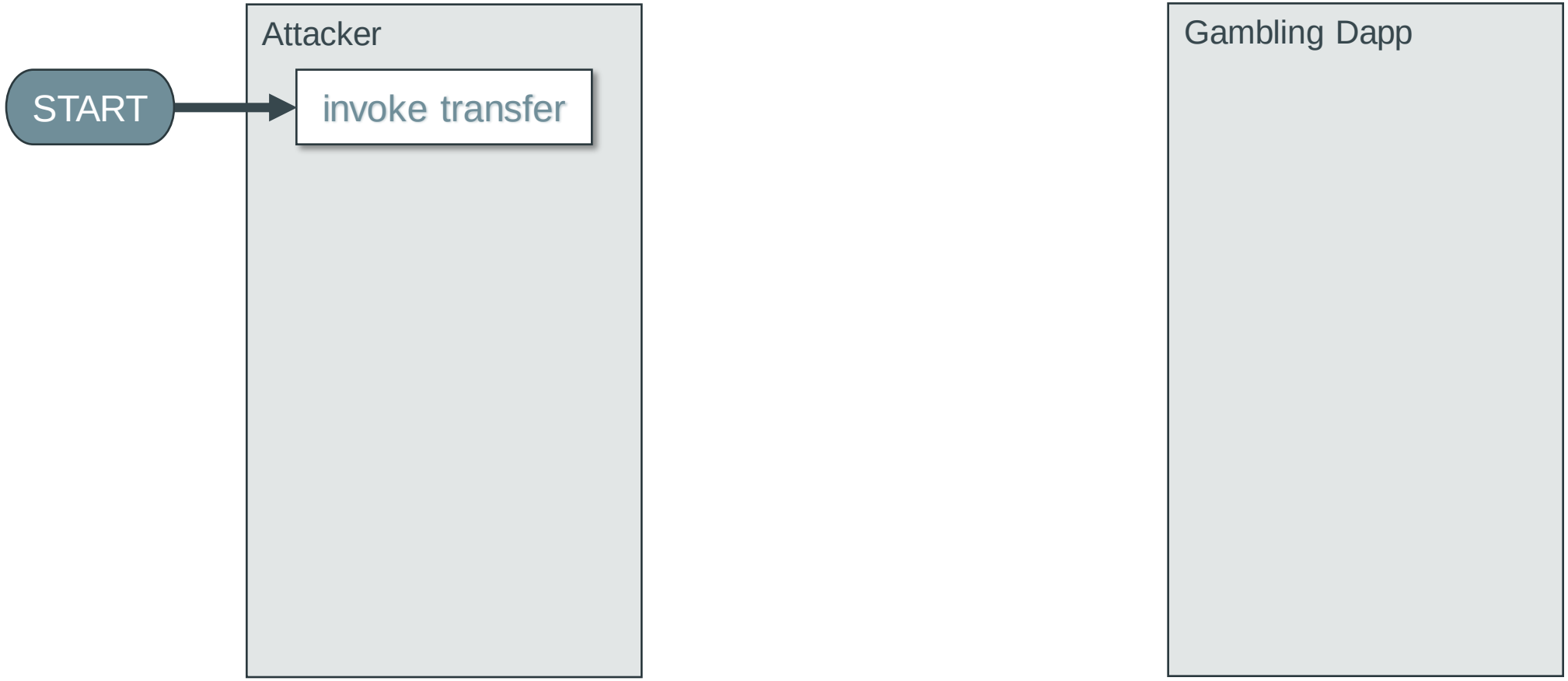


Rollback Vulnerability



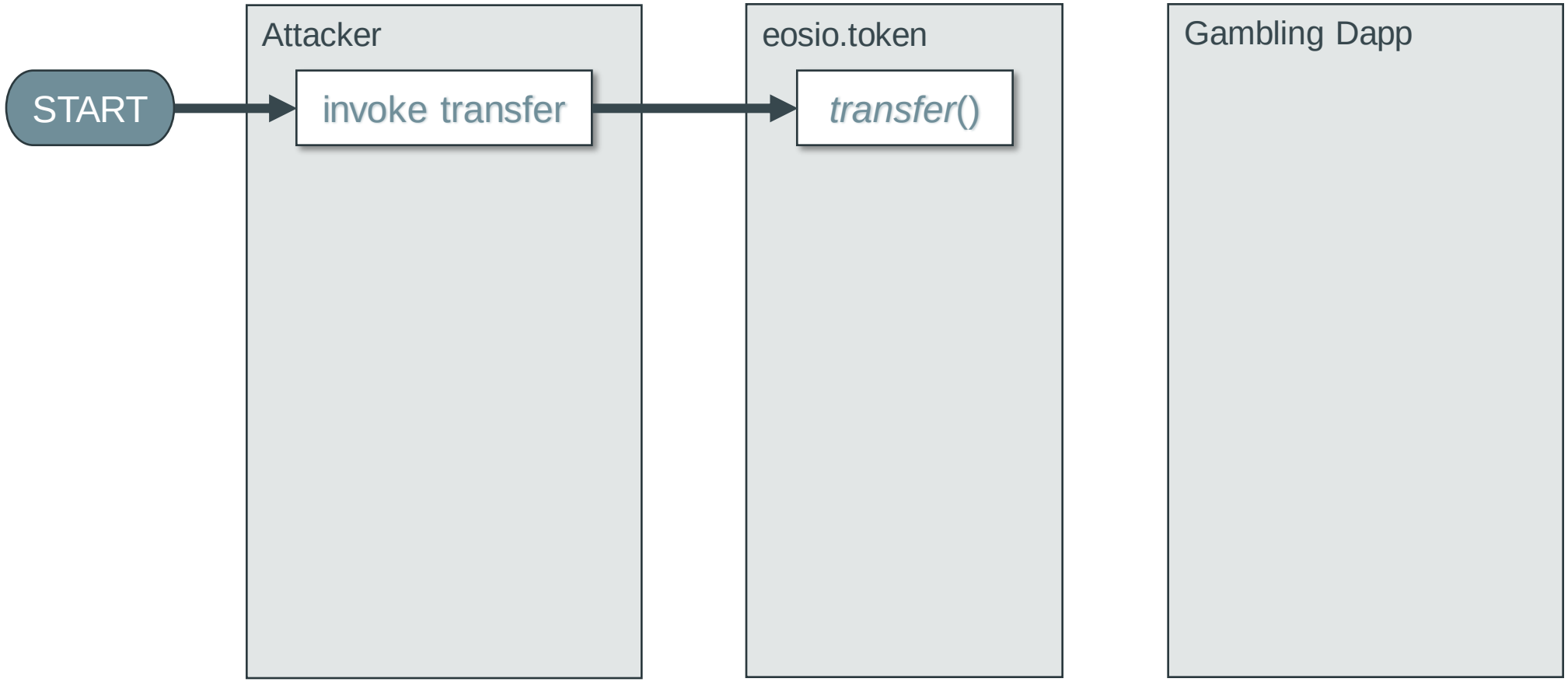
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



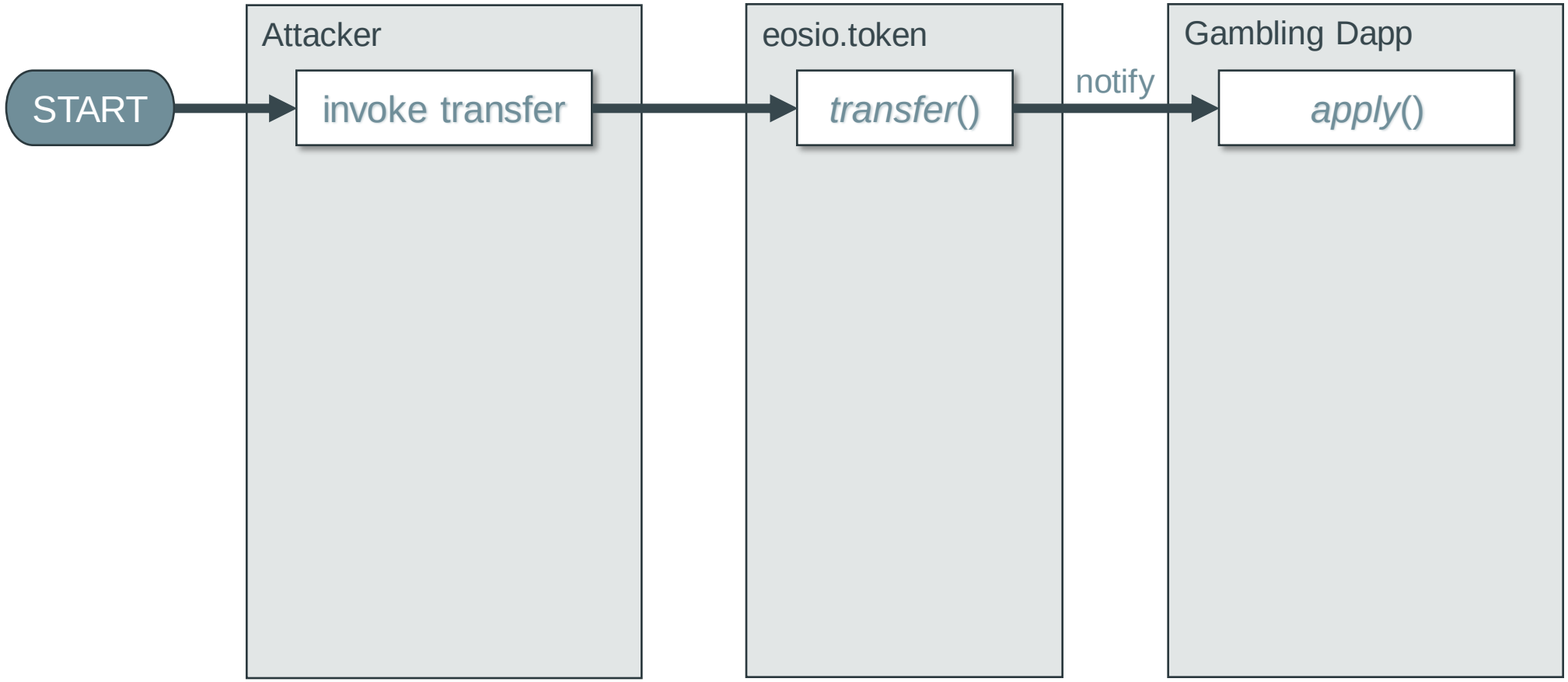
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



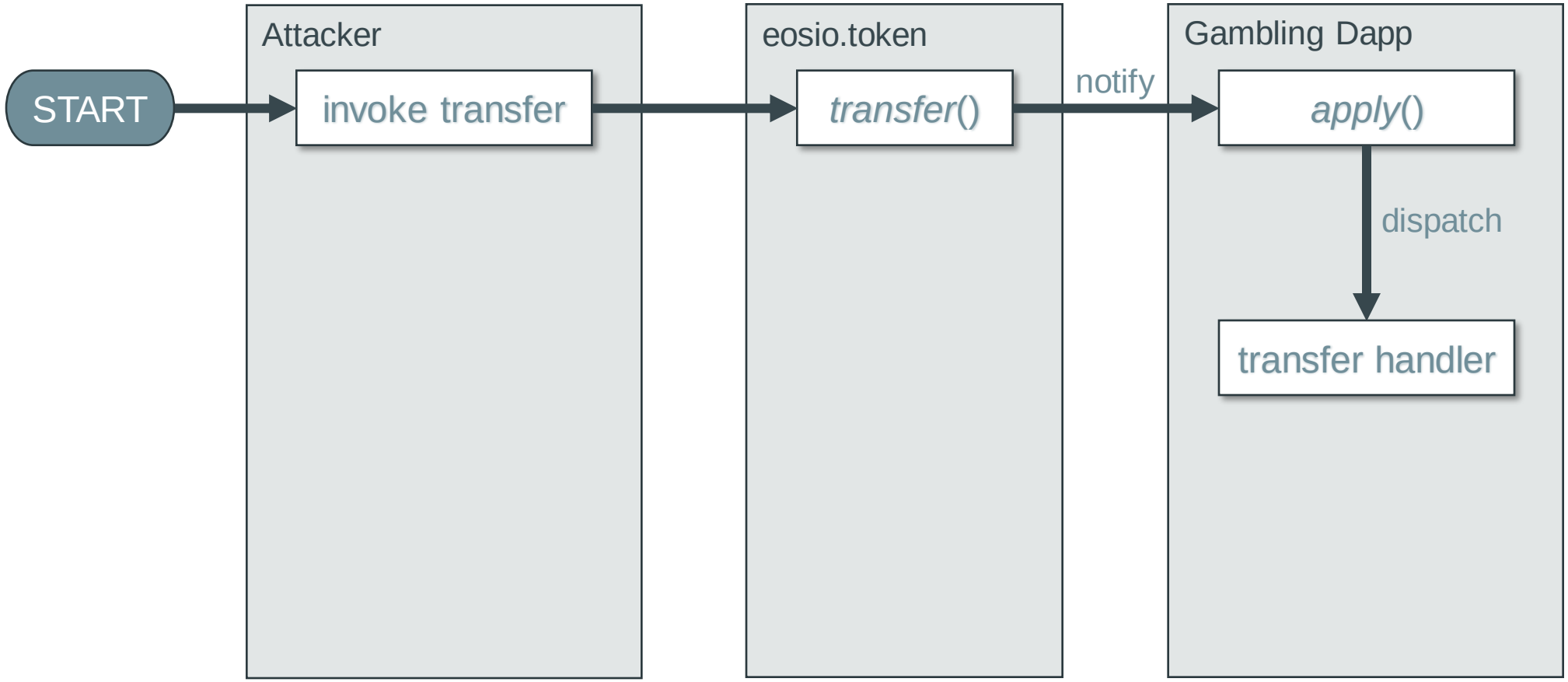
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



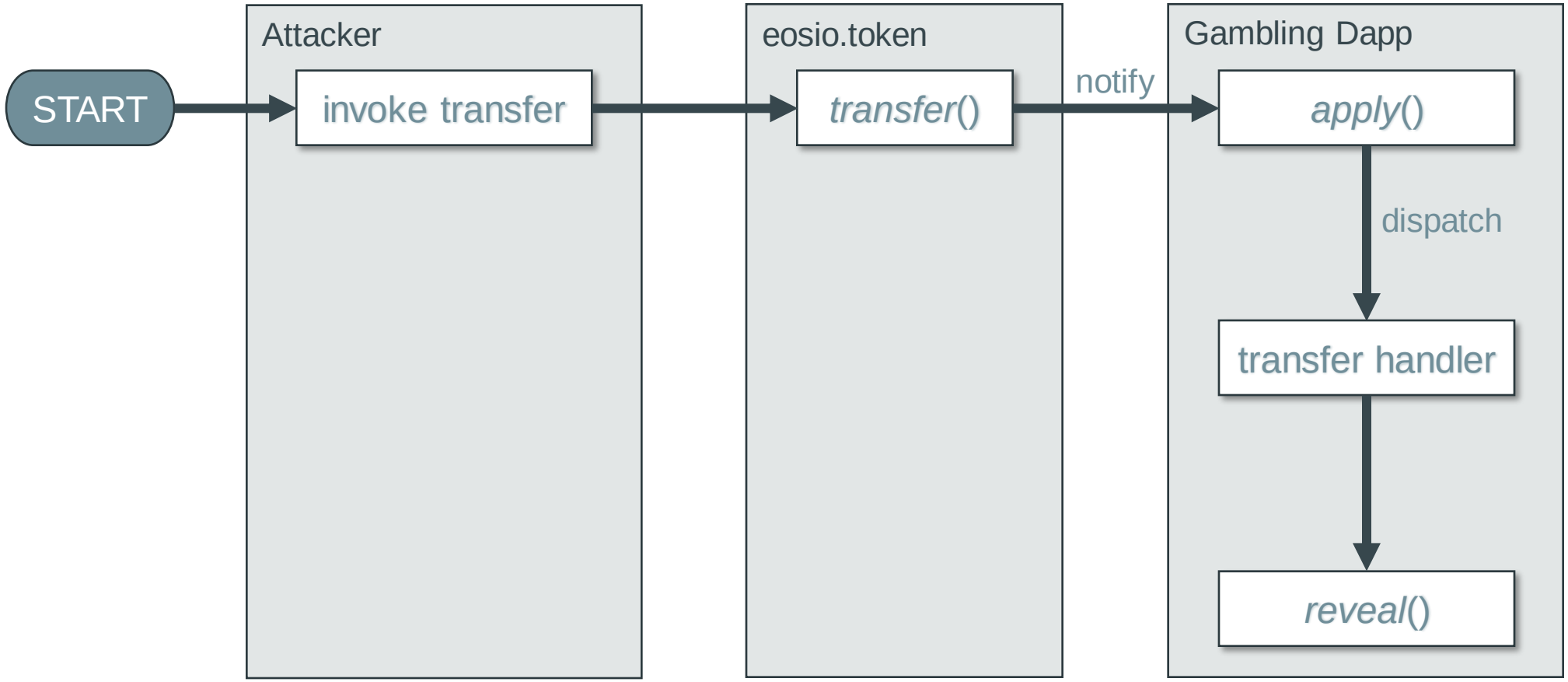
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



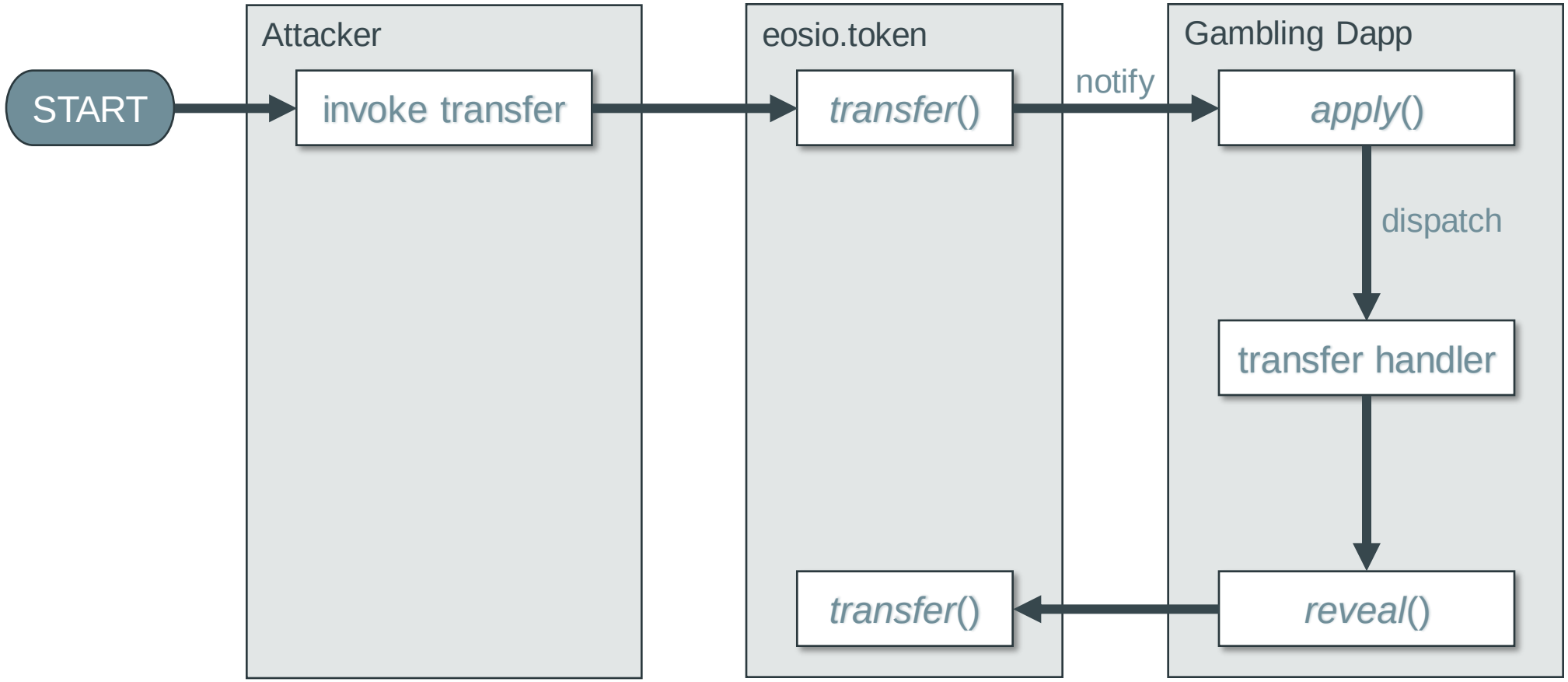
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



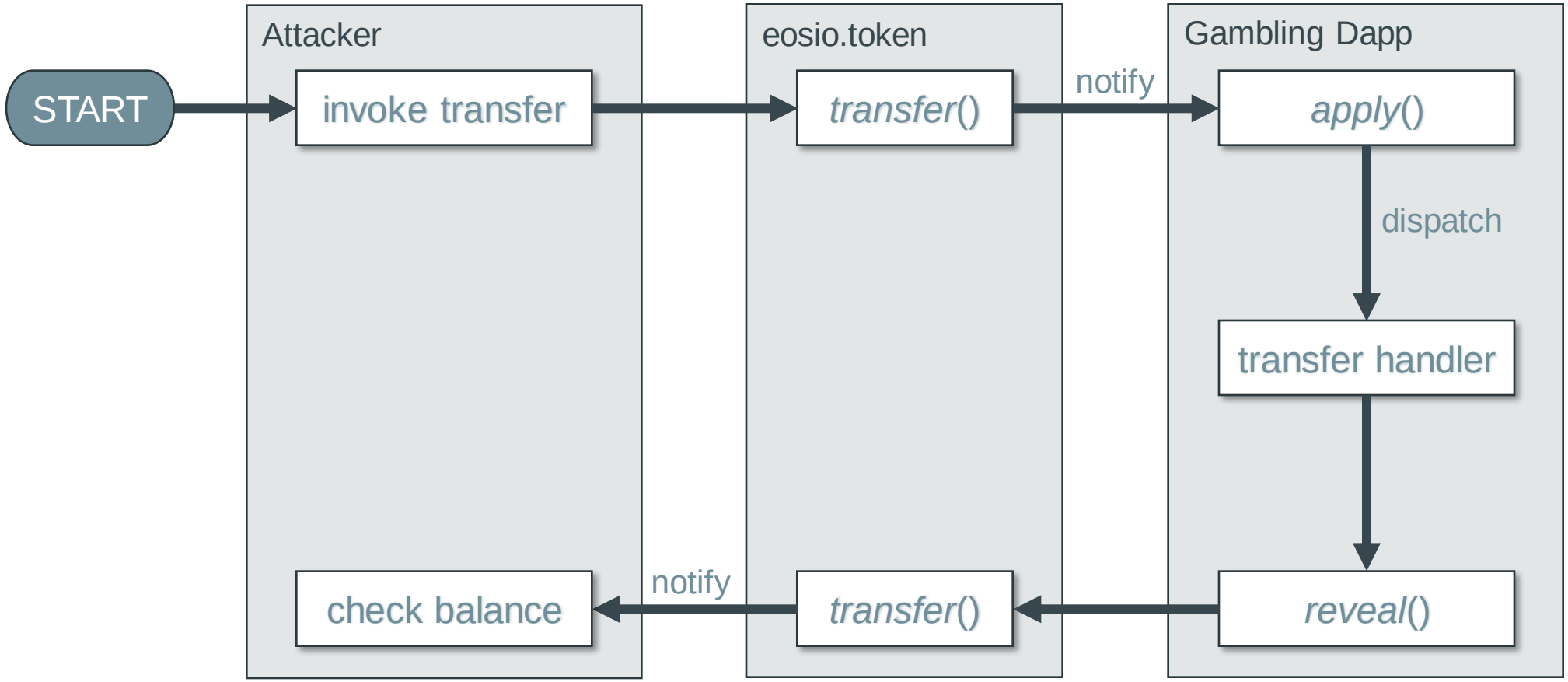
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



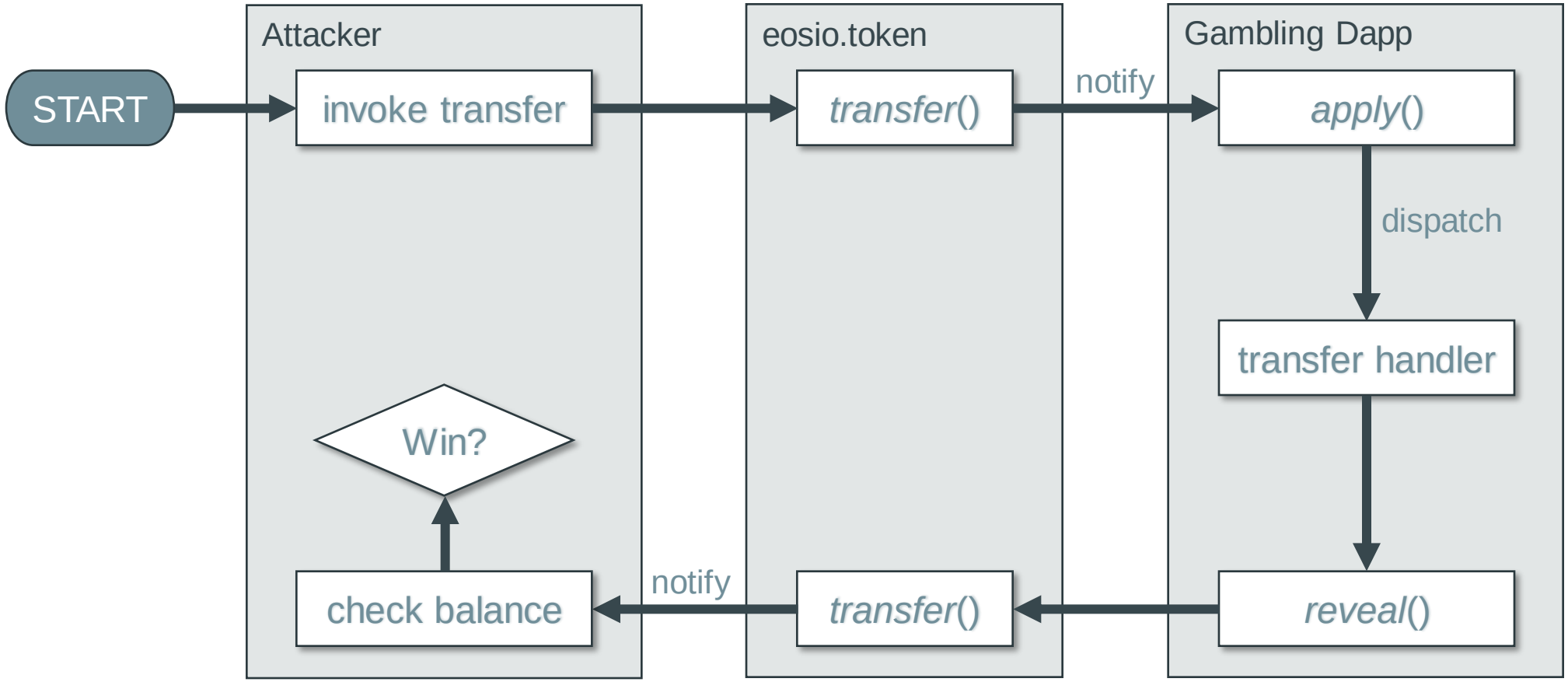
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



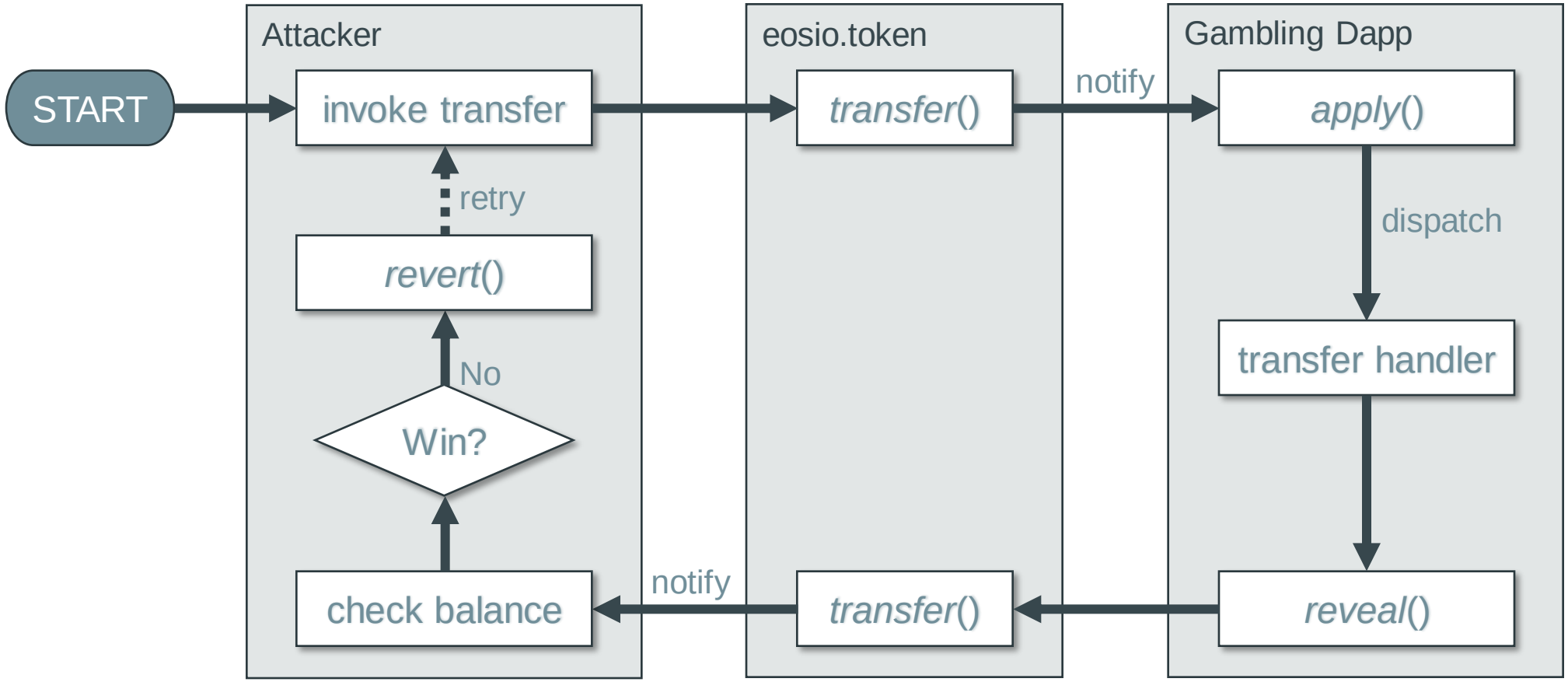
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



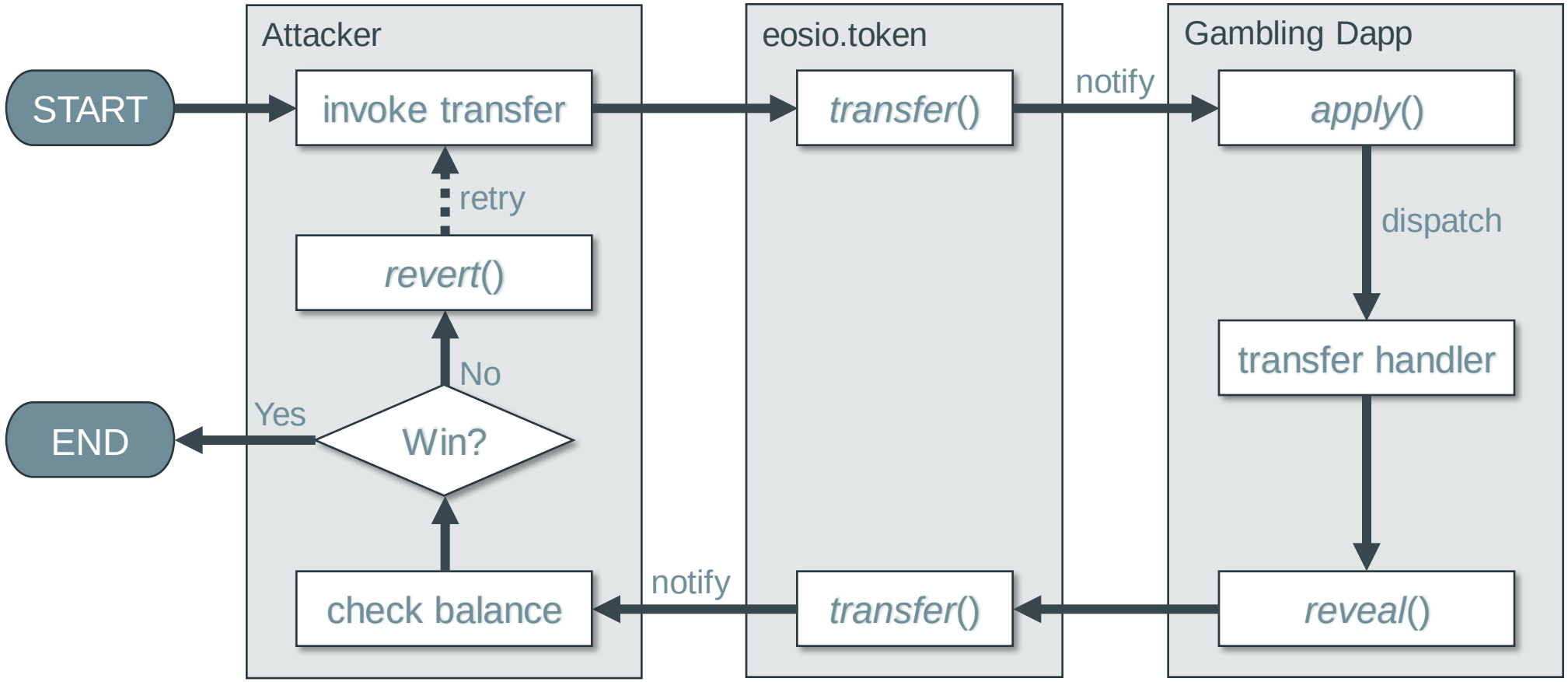
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



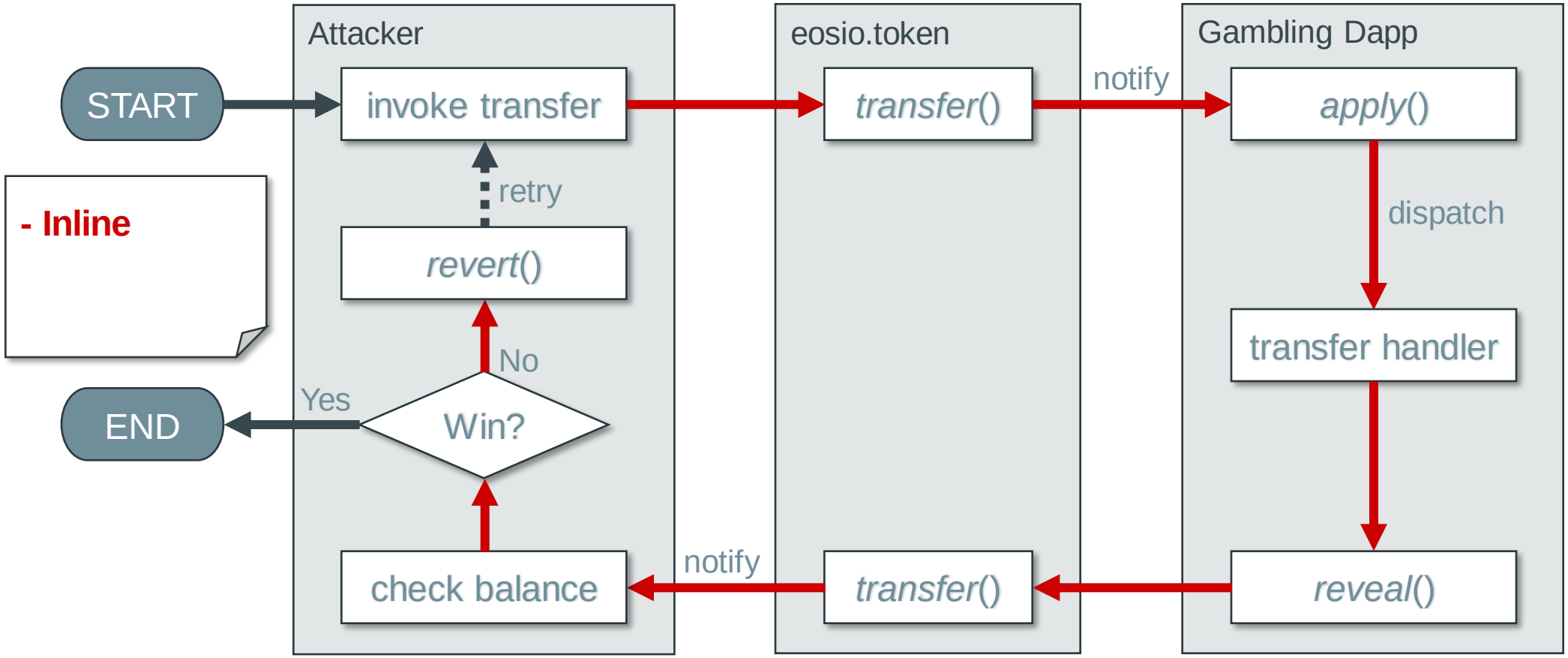
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



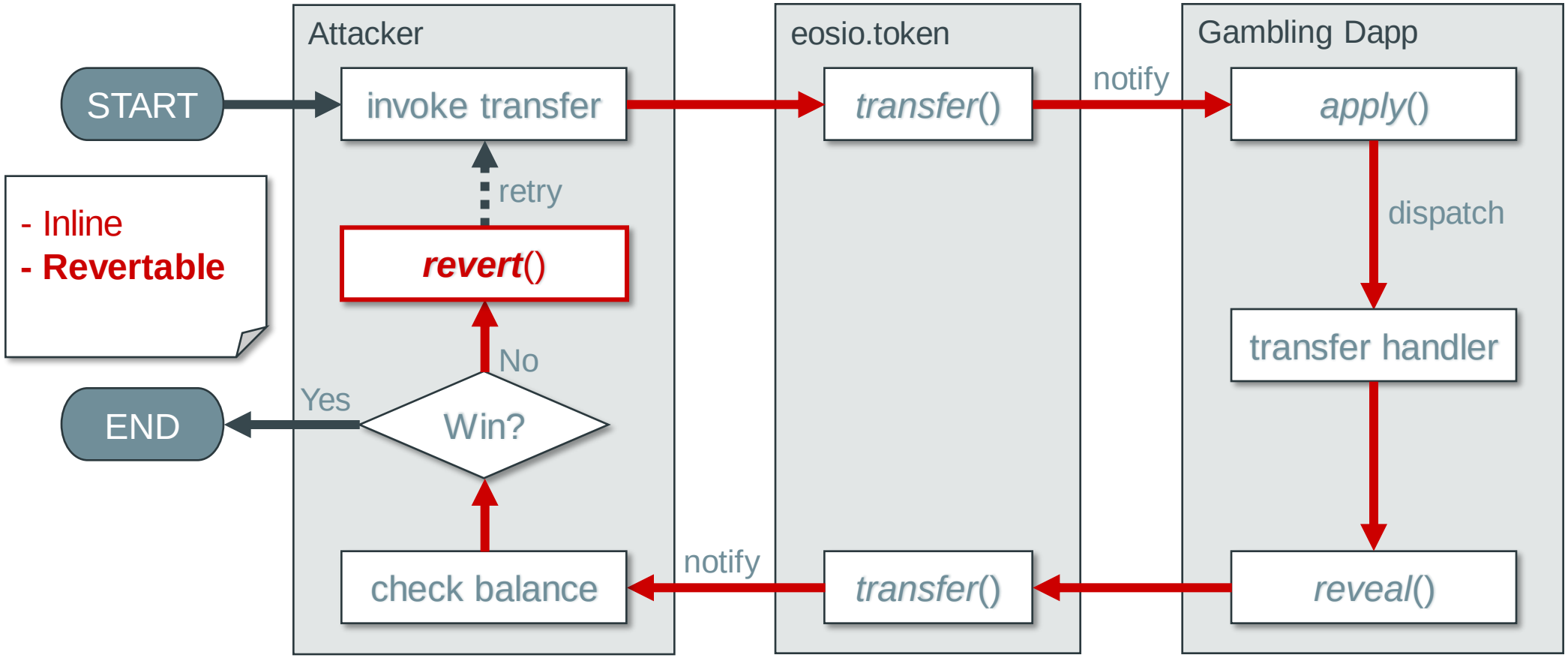
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



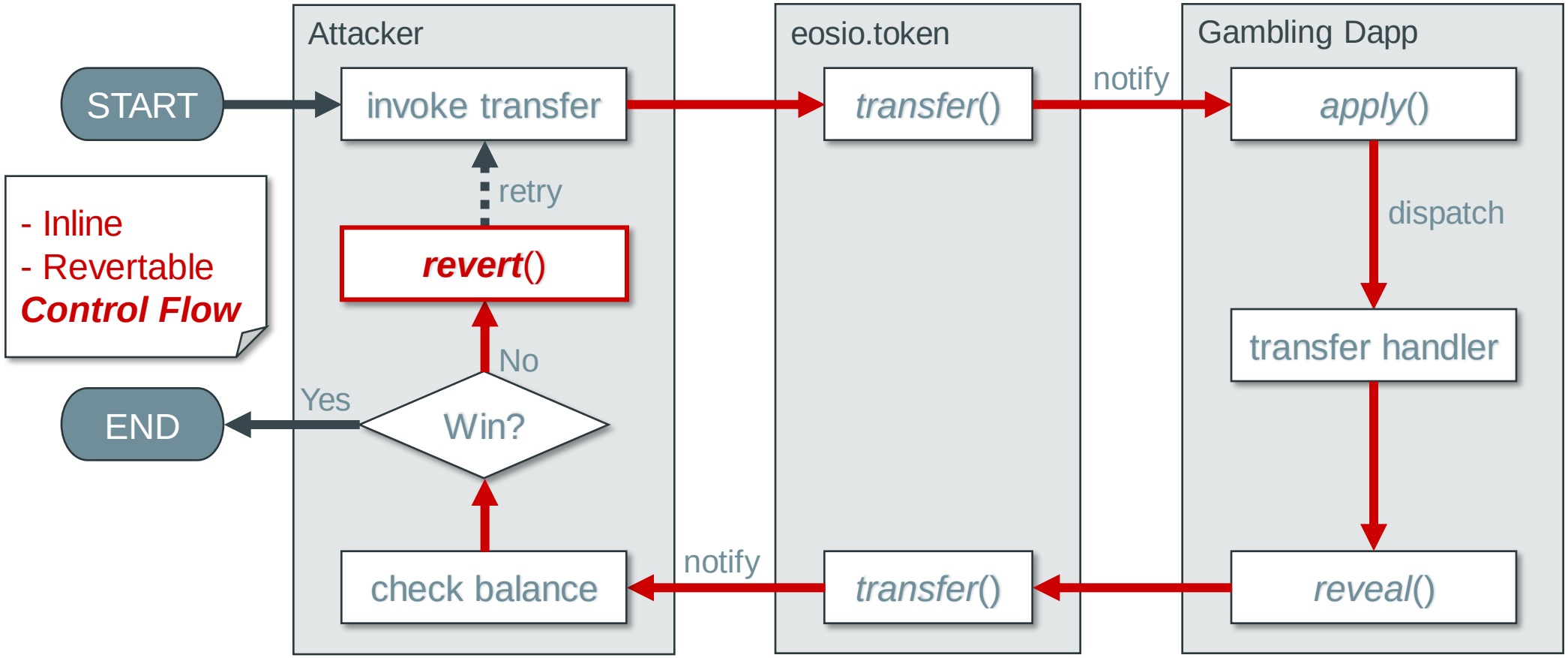
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability



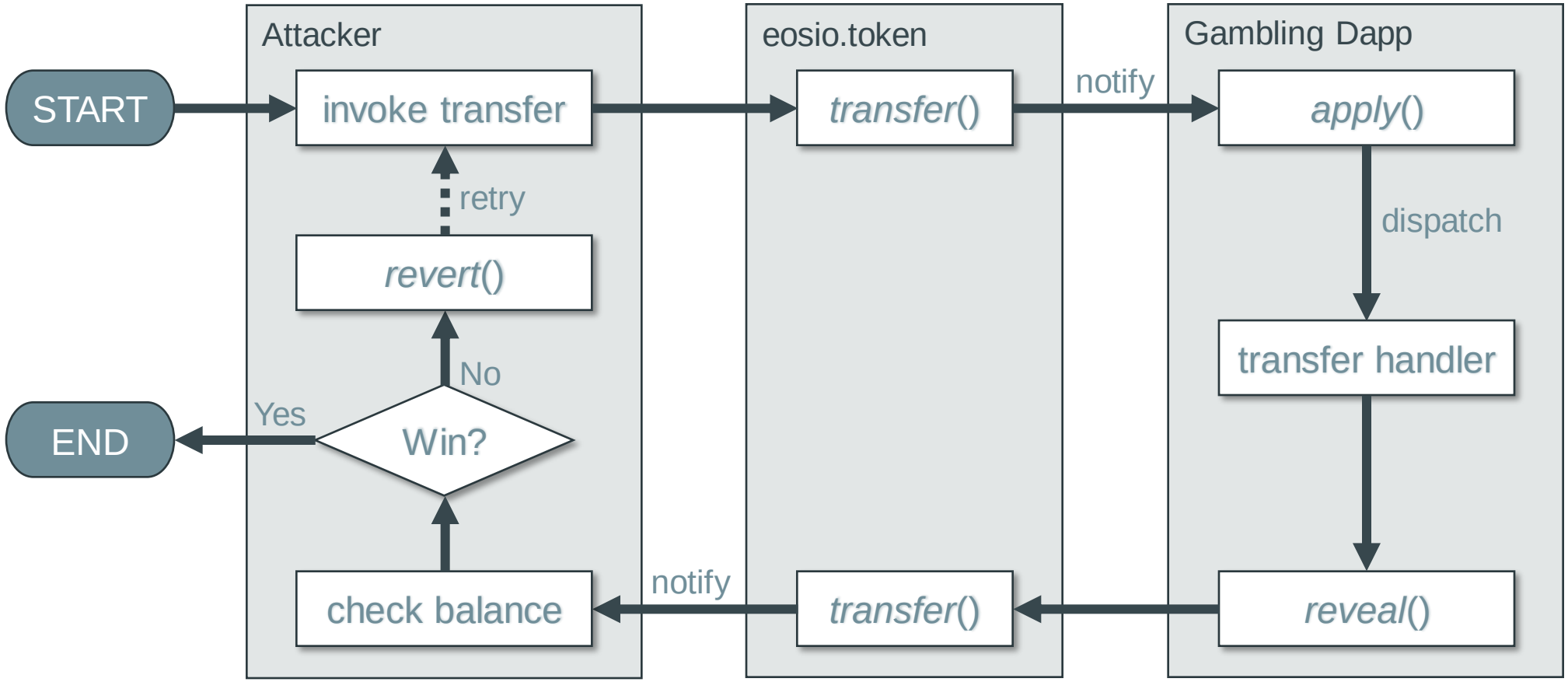
- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

Rollback Vulnerability

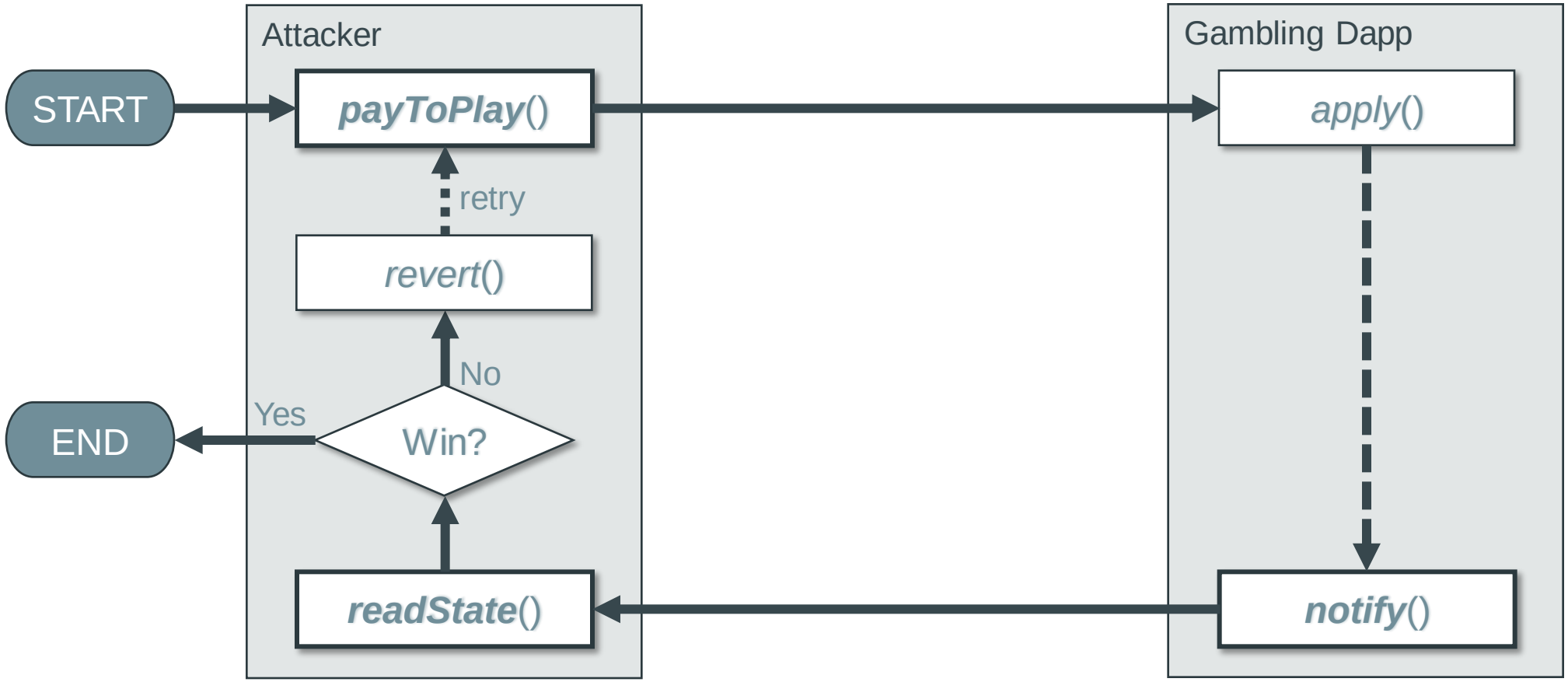


- EOSAFE (USENIX Security 21)
- WASAI (ISSTA 22)

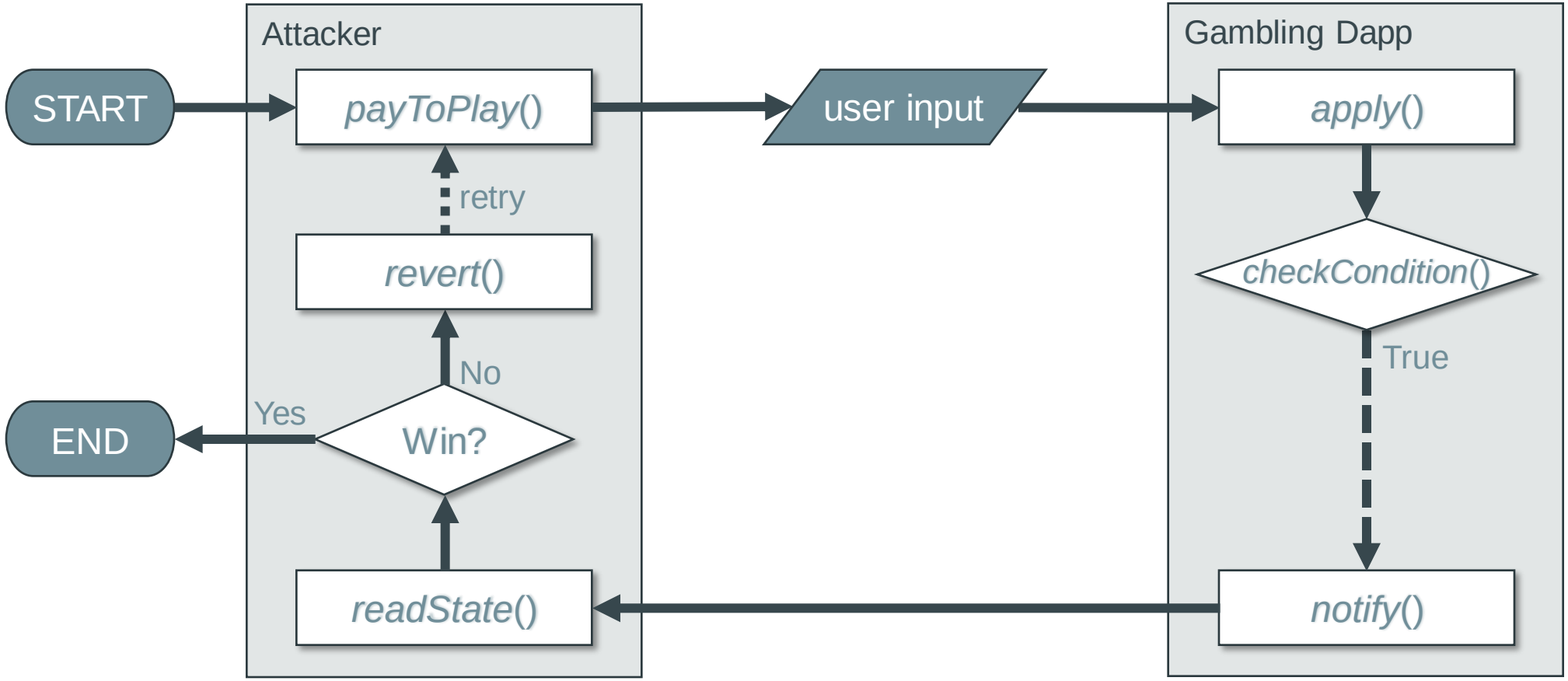
More Details…



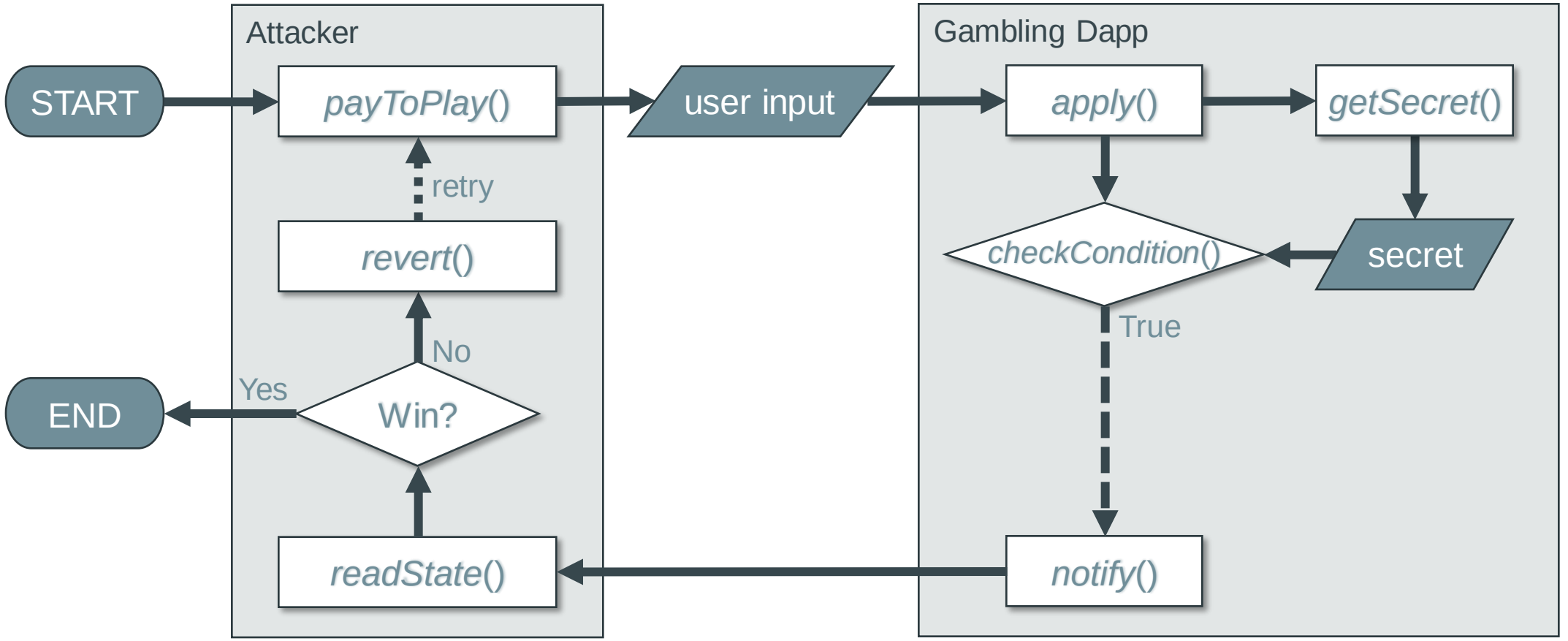
More Details…



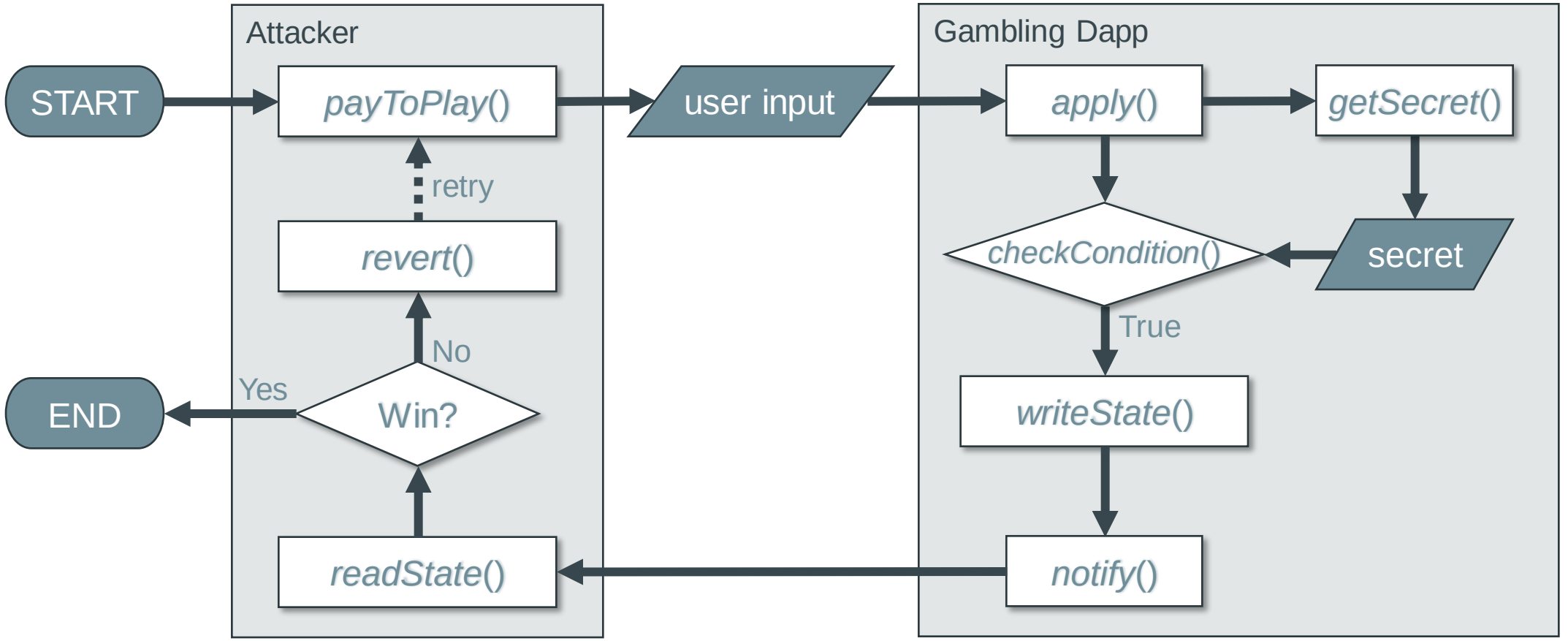
More Details…



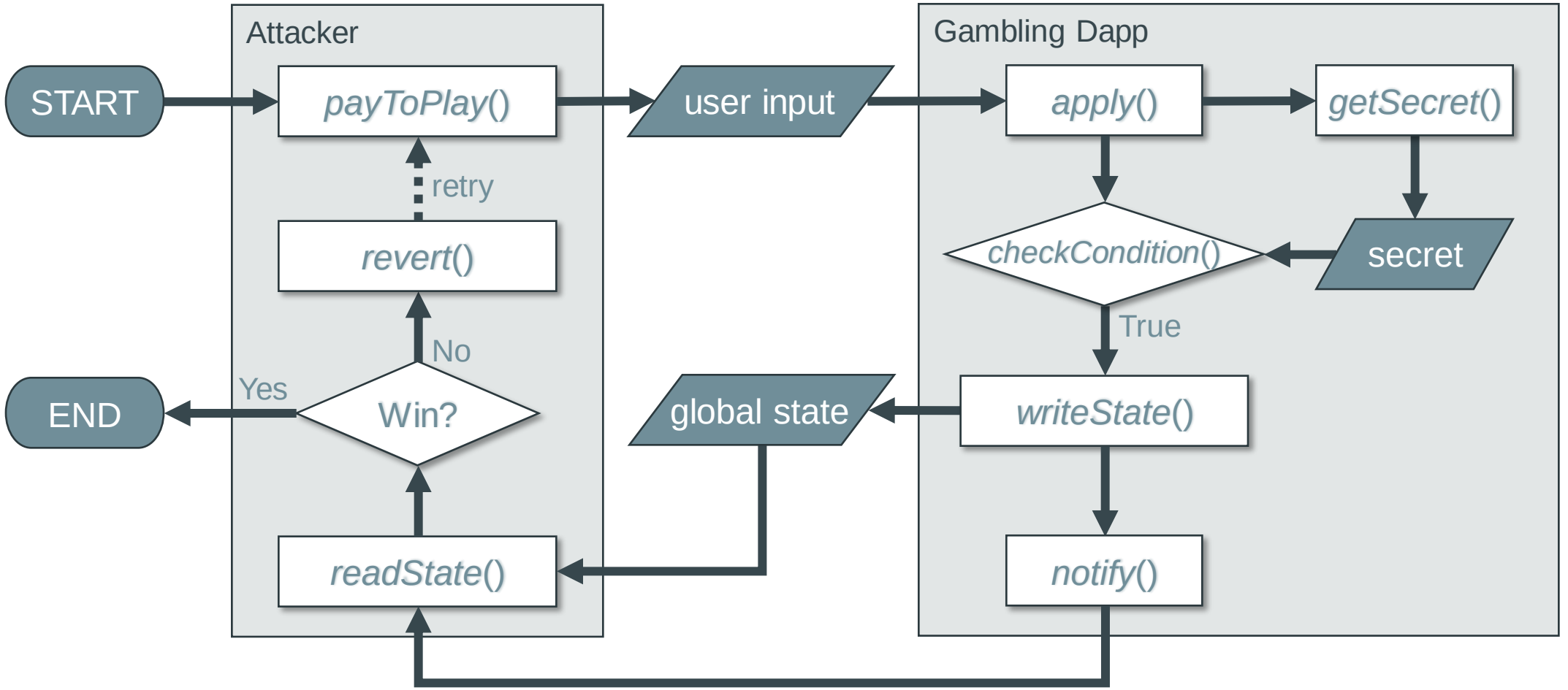
More Details…



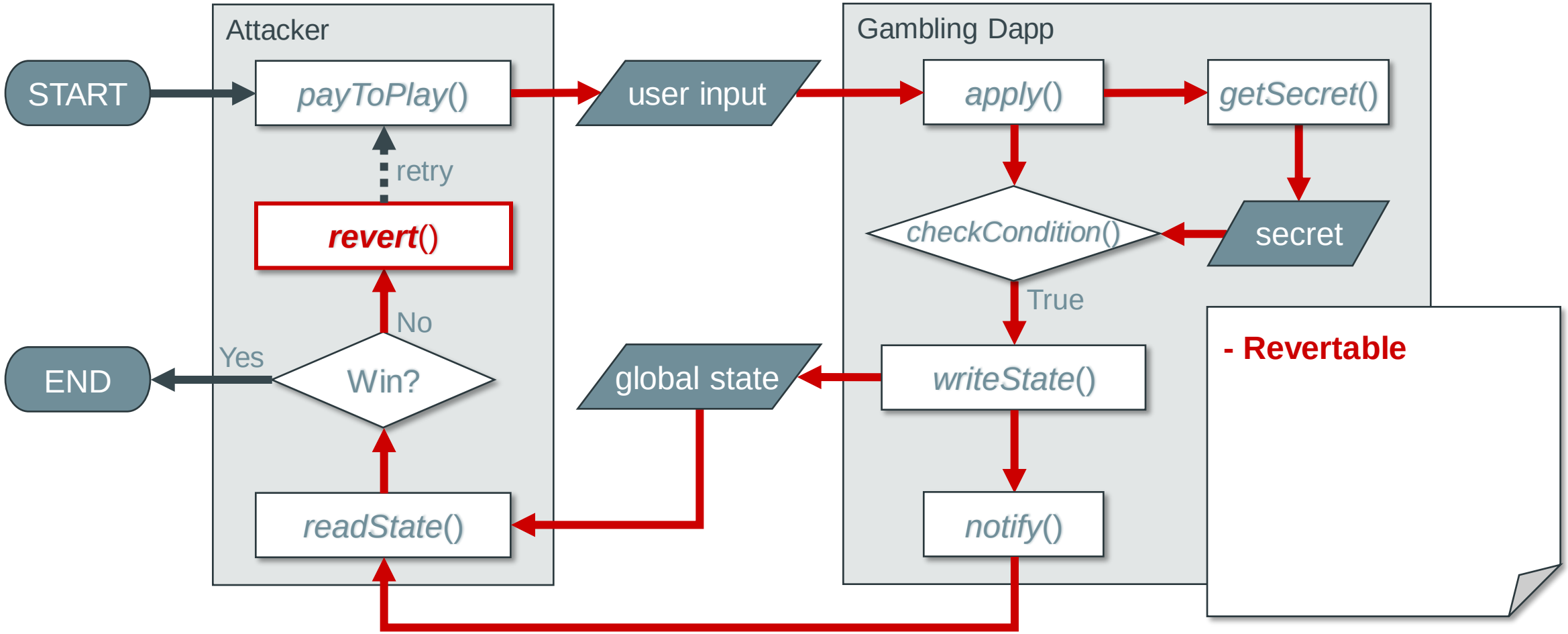
More Details…



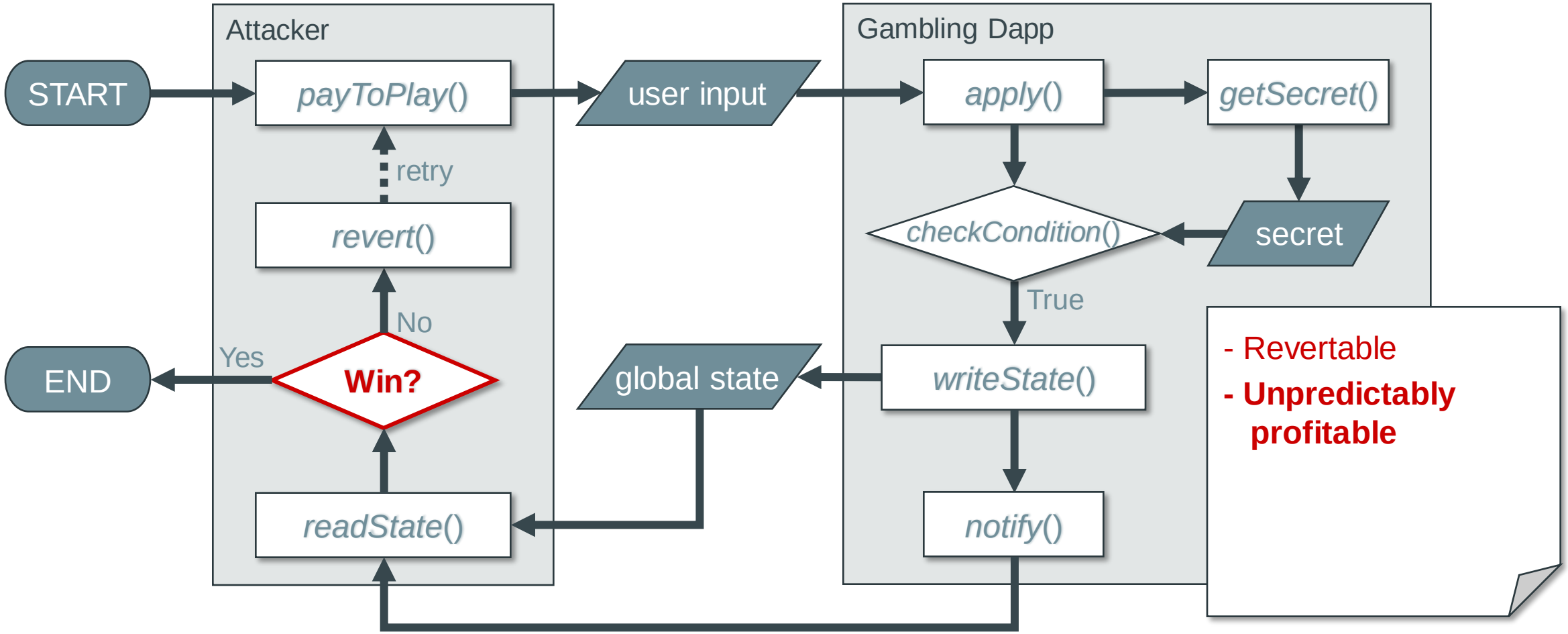
Groundhog Day Vulnerability



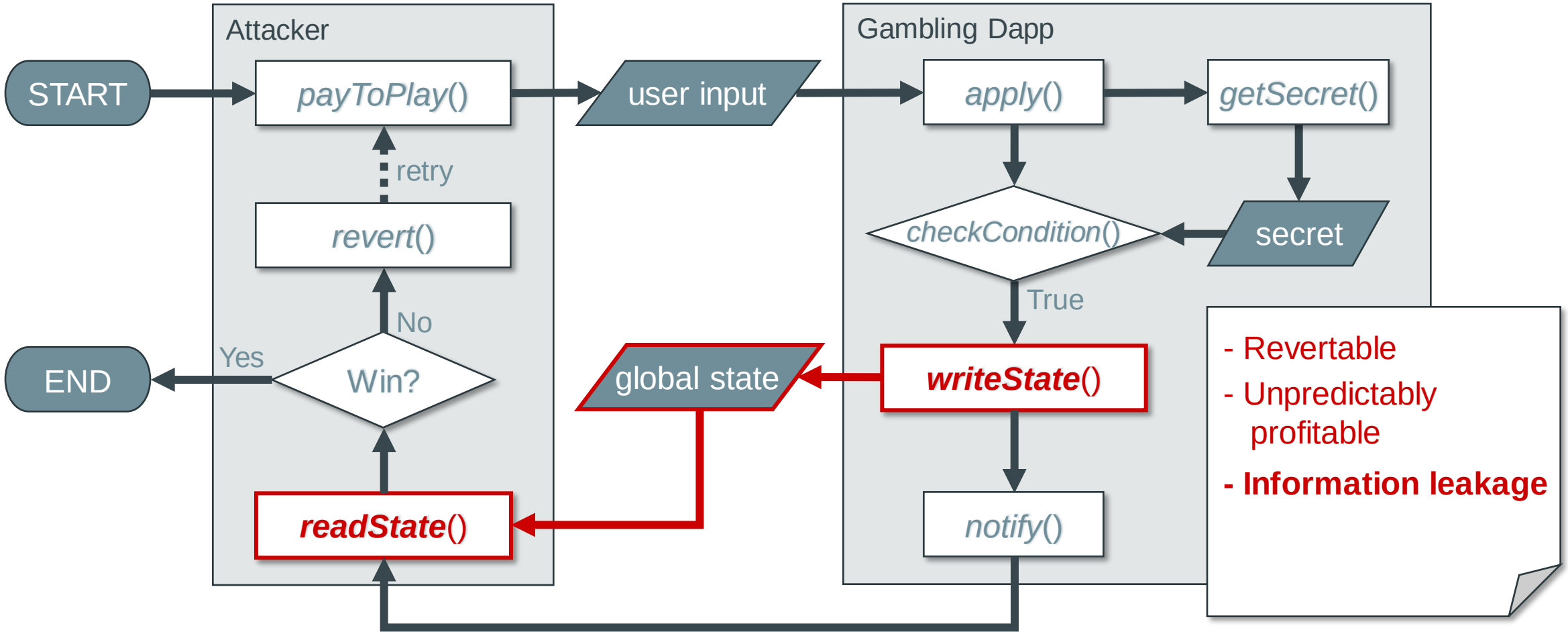
Groundhog Day Vulnerability



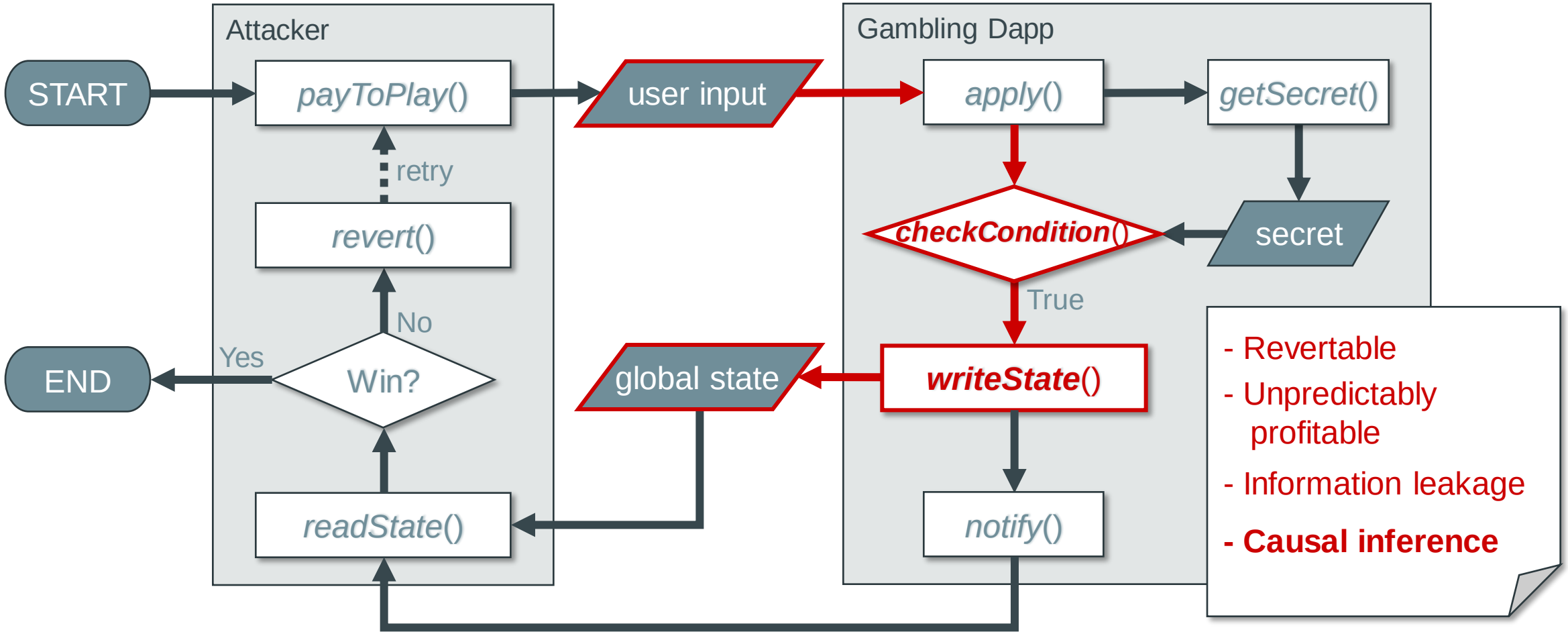
Groundhog Day Vulnerability



Groundhog Day Vulnerability



Groundhog Day Vulnerability



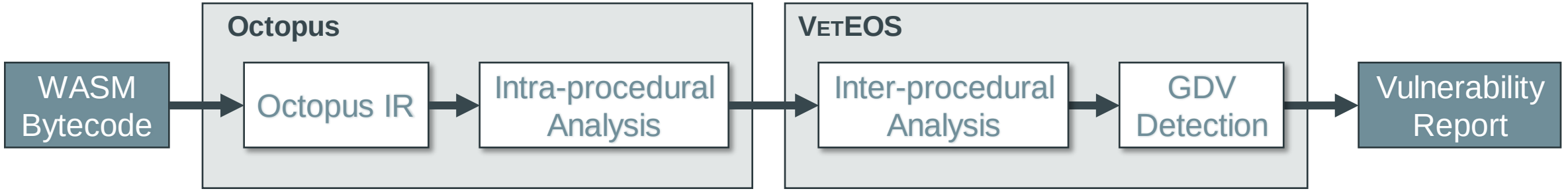
GDV Definition – 4 Factors

- F1: Revertable
- F2: Unpredictably profitable
- F3: Information leakage
- F4: Causal inference

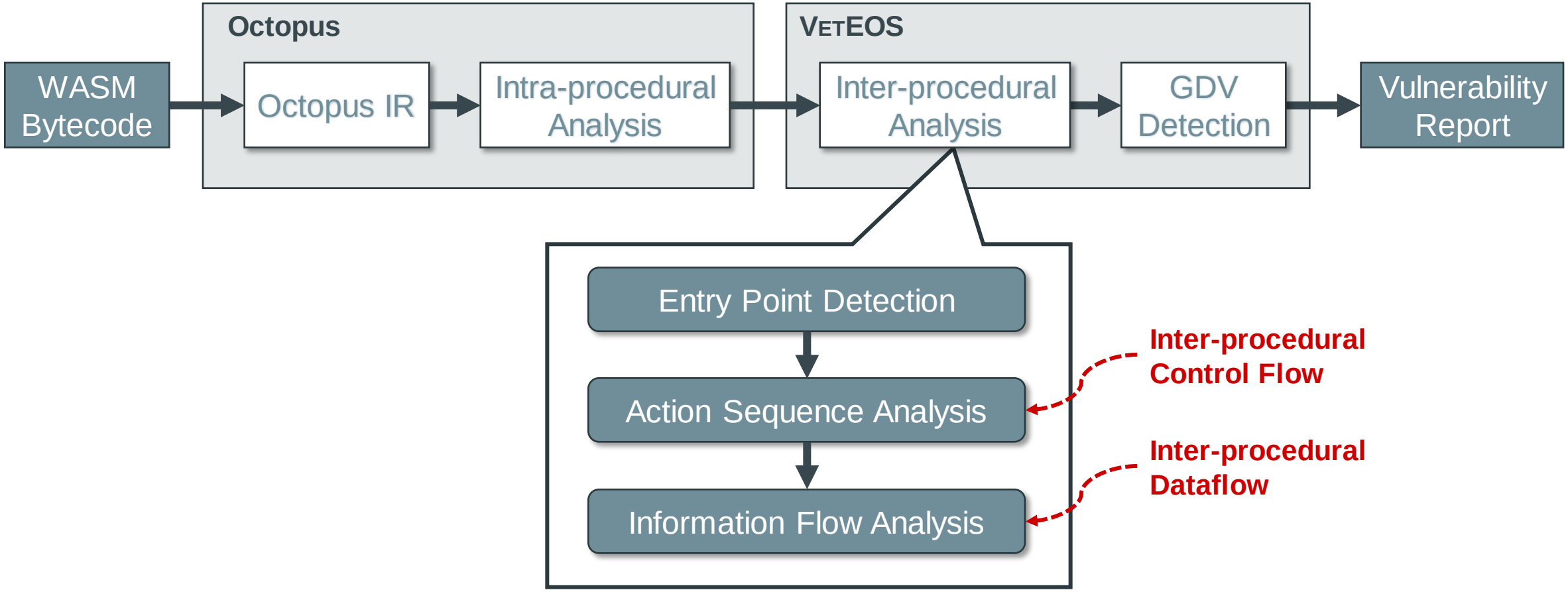
Control Flow

Dataflow

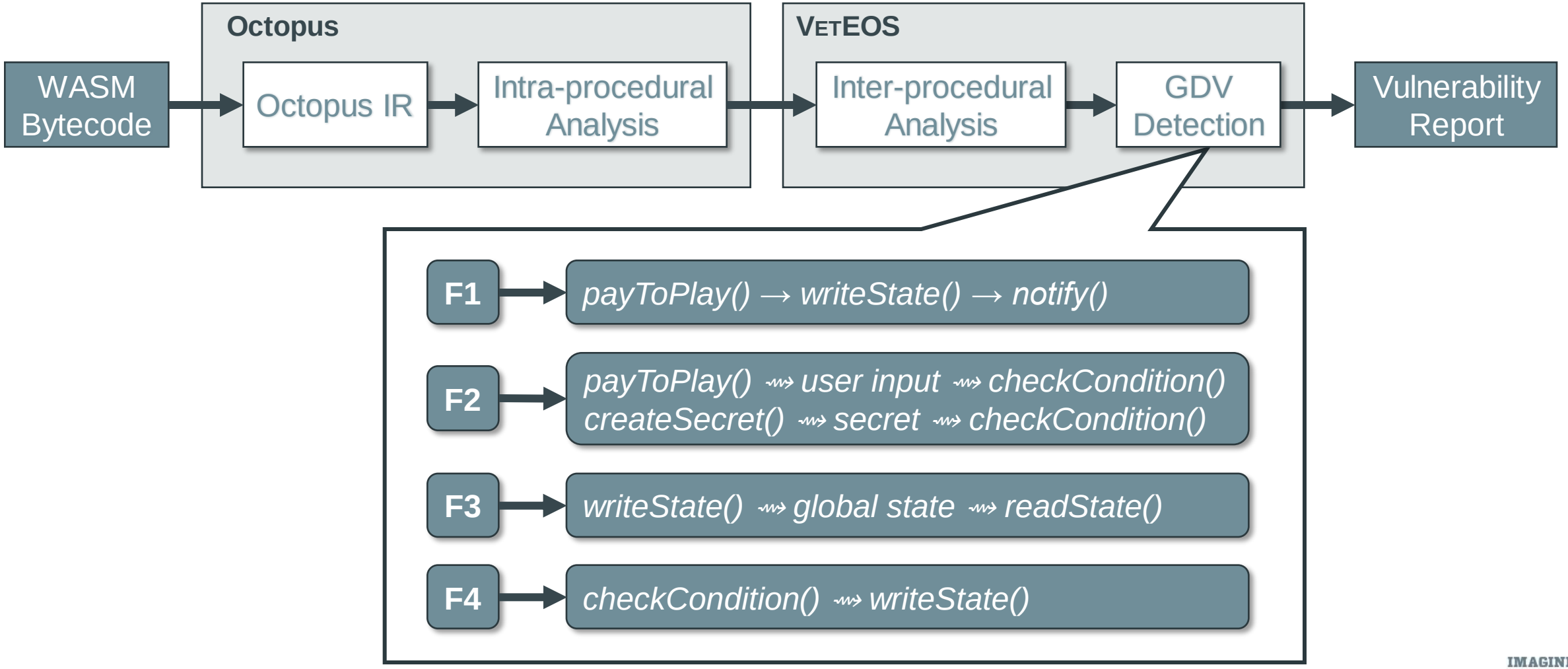
VETEOS – Design



VETEOS – Design



VetEOS – Design



VETEOS – Technical Challenges

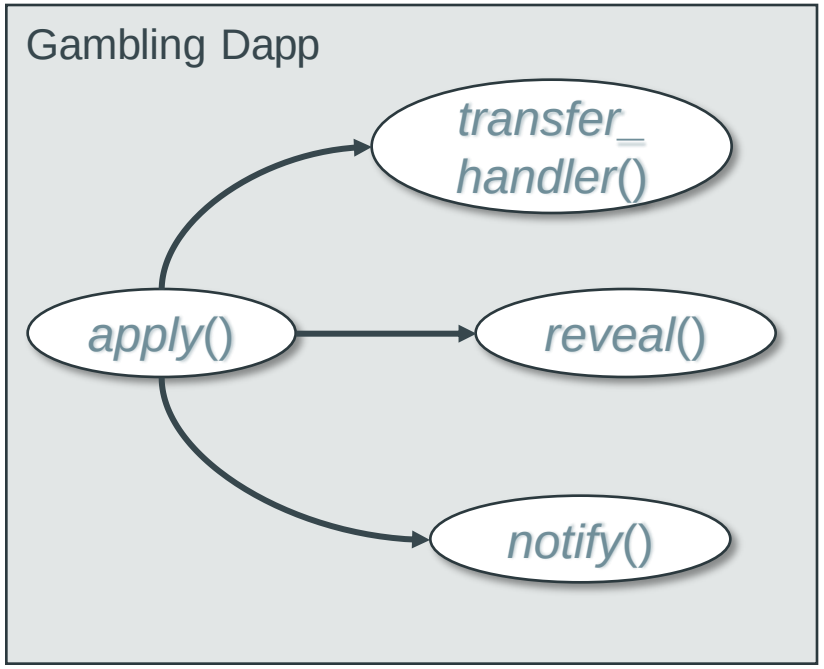
- **C1**: Entry point detection.
- **C2**: Control flow of implicit and indirect calls.

Control Flow Analysis

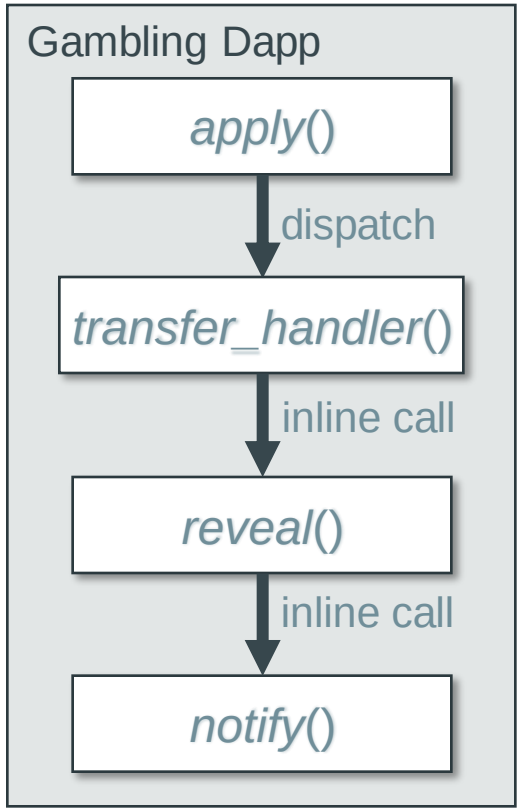
- **C3**: Data dependency of inline actions.
- **C4**: Dataflow of global variables in tables.

Dataflow Analysis

C1 – Entry Point Detection

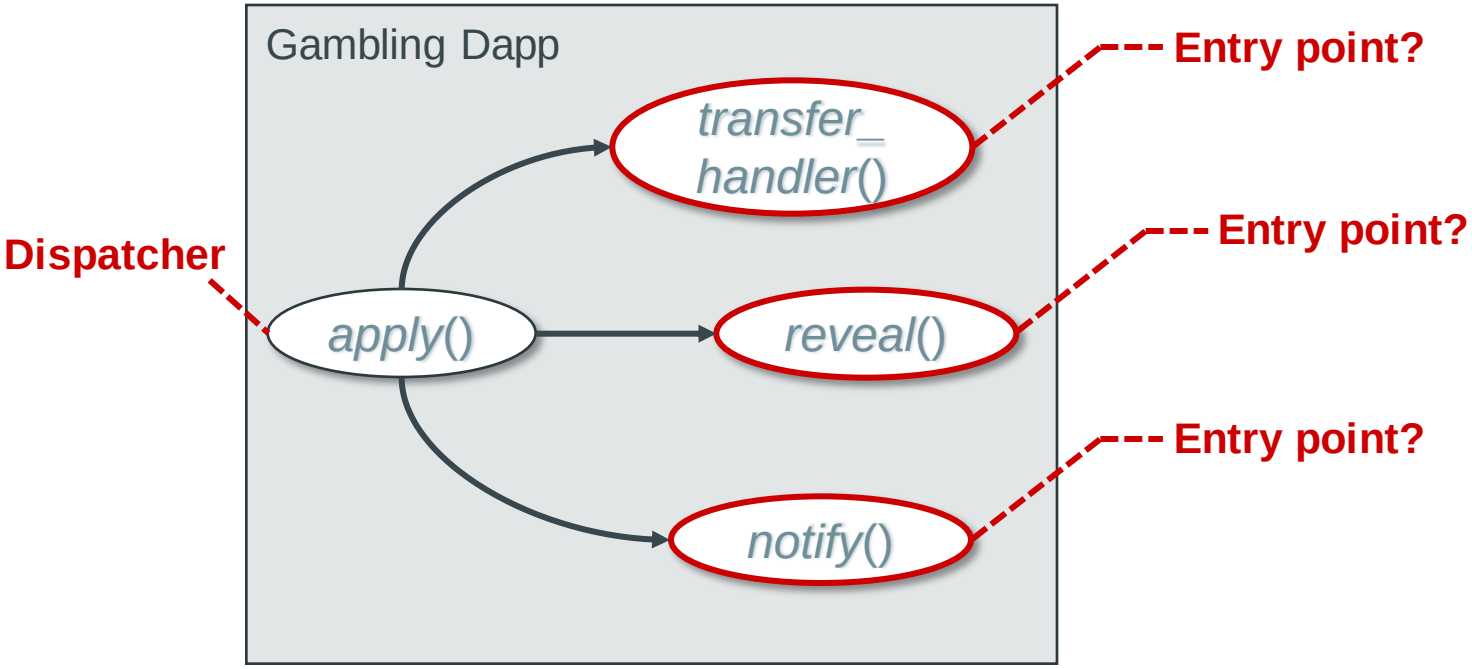


Function Call Graph

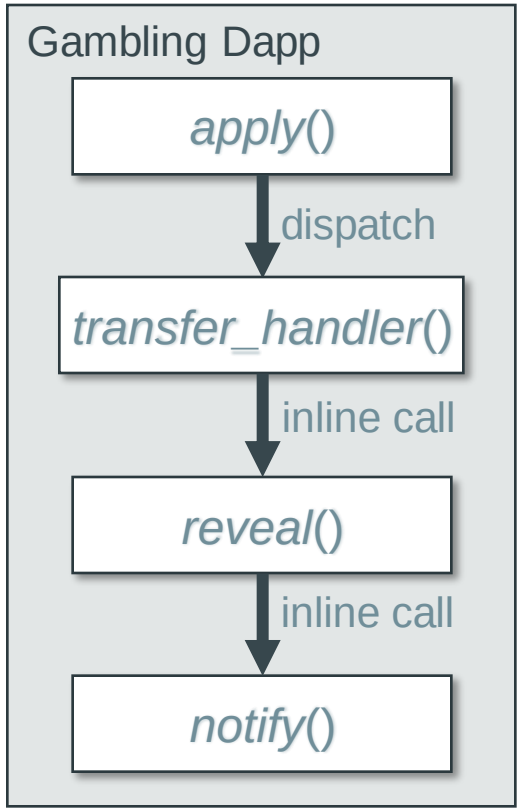


Action Sequence in Business Logic

C1 – Entry Point Detection

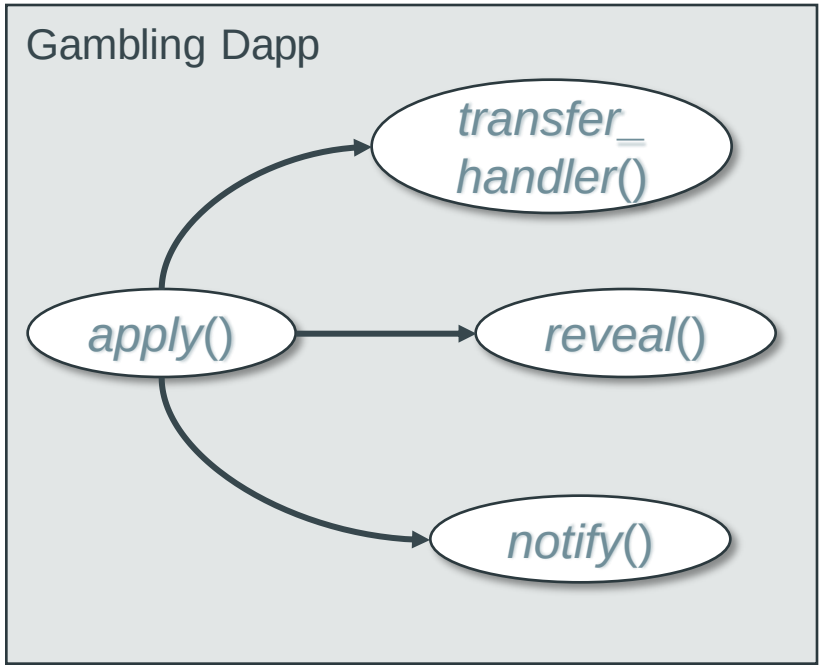


Function Call Graph

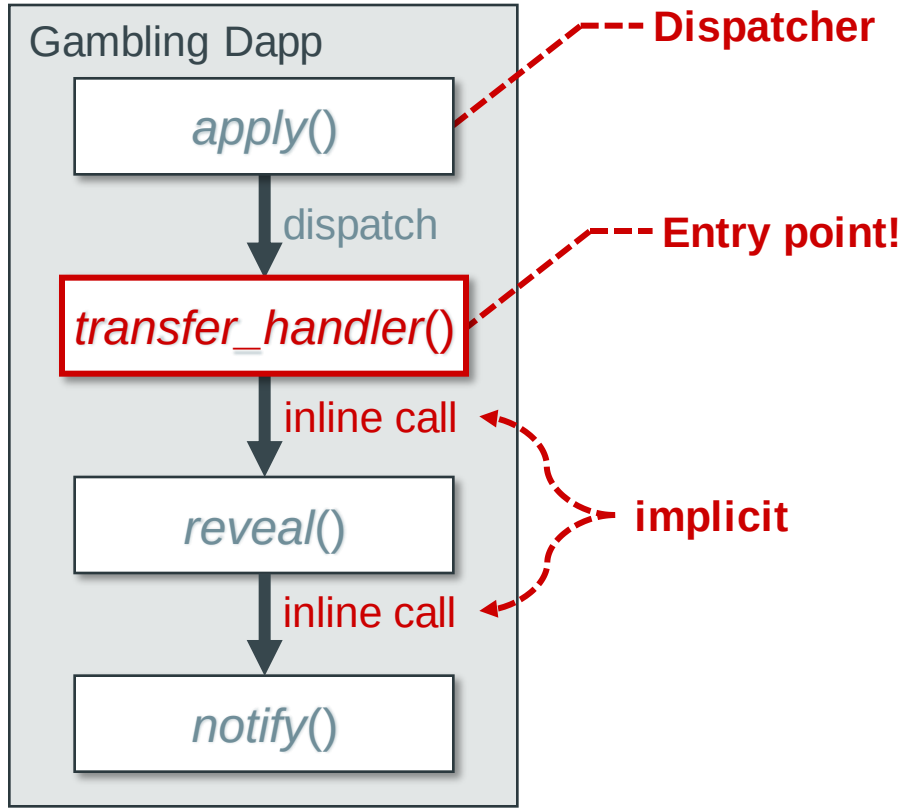


Action Sequence in Business Logic

C1 – Entry Point Detection



Function Call Graph



Action Sequence in Business Logic

C2 – Control Flow of Action Calls

```
1  void refund(eosio::name payee) {
2    eosio::name code = name("eosio.token");
3    eosio::name action = name("transfer");
4    action(
5      // permission level
6      permission_level{get_self(), "active"_n},
7      code,                        // target contract
8      action,                      // target action
9      std::make_tuple(             // transaction data
10       get_self(),                // token sender
11       payee,                     // token receiver
12       asset(10000,               // token amount
13         symbol("SYS", 4)),       // token symbol
14       std::string("refund"))     // transaction memo
15   ).send();
16   ...
17 }
```

Fig. 2: Example of Inline Action Calls

Implicit Call

```
1  void apply(uint64_t receiver, uint64_t code, uint64_t
       action) {
2    // action redirection
3    if (action == name("func1").value) {
4      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func3);
5    }
6    else if (action == name("func2").value) {
7      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func4);
8    }
9    ...
10 }
```

Fig. 1: Example of `apply()` Function

Indirect Call

C2 – Control Flow of Action Calls

```
1  void refund(eosio::name payee) {
2    eosio::name code = name("eosio.token");
3    eosio::name action = name("transfer");
4    action(
5      // permission level
6      permission_level{get_self(), "active"_n},
7      code,                          // target contract
8      action,                        // target action
9      std::make_tuple(               // transaction data
10       get_self(),                  // token sender
11       payee,                       // token receiver
12       asset(10000,                 // token amount
13         symbol("SYS", 4)),         // token symbol
14       std::string("refund"))       // transaction memo
15   ).send();
16   ...
17 }
```

Fig. 2: Example of Inline Action Calls

Implicit Call

```
1  void apply(uint64_t receiver, uint64_t code, uint64_t
       action) {
2    // action redirection
3    if (action == name("func1").value) {
4      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func3);
5    }
6    else if (action == name("func2").value) {
7      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func4);
8    }
9    ...
10 }
```

Fig. 1: Example of `apply()` Function

Indirect Call

C2 – Control Flow of Action Calls

```
1  void refund(eosio::name payee) {
2    eosio::name code = name("eosio.token");
3    eosio::name action = name("transfer");
4    action(
5      // permission level
6      permission_level{get_self(), "active"_n},
7      code,                          // target contract
8      action,                        // target action
9      std::make_tuple(               // transaction data
10       get_self(),                  // token sender
11       payee,                       // token receiver
12       asset(10000,                 // token amount
13         symbol("SYS", 4)),         // token symbol
14       std::string("refund"))       // transaction memo
15   ).send();
16   ...
17 }
```

String Analysis

Fig. 2: Example of Inline Action Calls

Implicit Call

```
1  void apply(uint64_t receiver, uint64_t code, uint64_t
       action) {
2    // action redirection
3    if (action == name("func1").value) {
4      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func3);
5    }
6    else if (action == name("func2").value) {
7      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func4);
8    }
9    ...
10 }
```

String Analysis

Pointer Analysis

Fig. 1: Example of `apply()` Function

Indirect Call

C2 – Control Flow of Action Calls

```
1  void refund(eosio::name payee) {
2    eosio::name code = name("eosio.token");
3    eosio::name action = name("transfer");
4    action(
5      // permission level
6      permission_level{get_self(), "active"_n},
7      code,                          // target contract
8      action,                        // target action
9      std::make_tuple(               // transaction data
10       get_self(),                  // token sender
11       payee,                       // token receiver
12       asset(10000,                 // token amount
13         symbol("SYS", 4)),         // token symbol
14       std::string("refund"))       // transaction memo
15   ).send();
16   ...
17 }
```

String Analysis

Fig. 2: Example of Inline Action Calls

Implicit Call

```
1  void apply(uint64_t receiver, uint64_t code, uint64_t
       action) {
2    // action redirection
3    if (action == name("func1").value) {
4      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func3);
5    }
6    else if (action == name("func2").value) {
7      eosio::execute_action(eosio::name(receiver), eosio::
         name(code), &mycontract::func4);
8    }
9    ...
10 }
```

String Analysis

Pointer Analysis

Fig. 1: Example of `apply()` Function

Indirect Call

```
a9: %C0 = get_local 6()
ab: call_indirect(%C0)
```

function index

WASM Instructions

C3 – Dataflow of Inline Actions

Read state

Write state

```
4   [[eosio::action]] void reveal(eosio::name username, std
      ::string user_input) {
5     action(permission_level{"gambling"_n, "active"_n},
6       "gambling"_n, "getbalance"_n,
7       std::make_tuple(username)).send();
8     ...
9     uint64_t secret = getSecret();
10    if (checkCondition(user_input, secret)) {
11      writeState(username, state);
12      createSecret(); ...
13    }
14    notify(username, message);
15  }
```

Inline call

```
16  [[eosio::action]] void getbalance(eosio::name username){
17    require_auth(get_self());
18    uint64_t balance = readState(username);
19    ...
20    notify(username, message);
21  }
```

IMAGINE

C3 – Dataflow of Inline Actions

Read state

Write state

```
4   [[eosio::action]] void reveal(eosio::name username, std
      ::string user_input) {
5     action(permission_level{"gambling"_n, "active"_n},
6       "gambling"_n, "getbalance"_n,
7       std::make_tuple(username)).send();
8     ...
9     uint64_t secret = getSecret();
10    if (checkCondition(user_input, secret)) {
11      writeState(username, state);
12      createSecret(); ...
13    }
14    notify(username, message);
15  }
```

Source Code Order

Action Order Inferred from Source Code

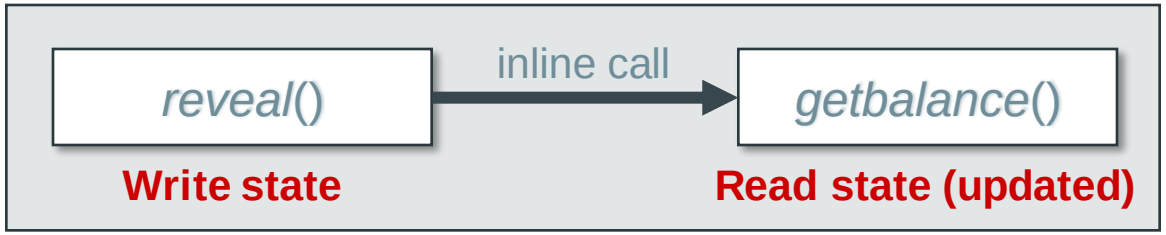
C3 – Dataflow of Inline Actions

Read state

Write state

```
4   [[eosio::action]] void reveal(eosio::name username, std
      ::string user_input) {
5     action(permission_level{"gambling"_n, "active"_n},
6       "gambling"_n, "getbalance"_n,
7       std::make_tuple(username)).send();
8     ...
9     uint64_t secret = getSecret();
10    if (checkCondition(user_input, secret)) {
11      writeState(username, state);
12      createSecret(); ...
13    }
14    notify(username, message);
15  }
```

Source Code Order



Real Action Order

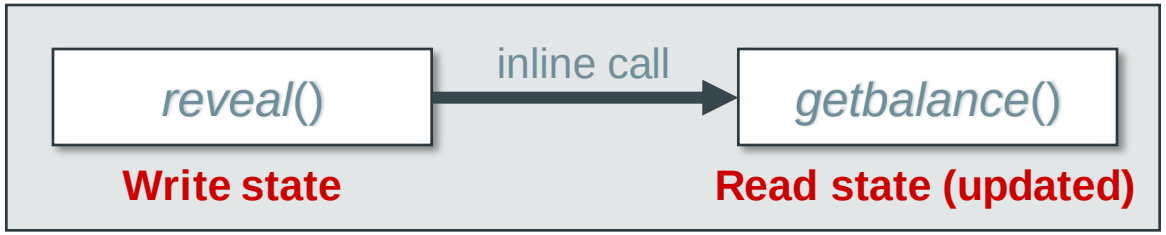
C3 – Dataflow of Inline Actions

Read state

Write state

```
4   [[eosio::action]] void reveal(eosio::name username, std
      ::string user_input) {
5     action(permission_level{"gambling"_n, "active"_n},
6       "gambling"_n, "getbalance"_n,
7       std::make_tuple(username)).send();
8     ...
9     uint64_t secret = getSecret();
10    if (checkCondition(user_input, secret)) {
11      writeState(username, state);
12      createSecret(); ...
13    }
14    notify(username, message);
15  }
```

Source Code Order



Real Action Order

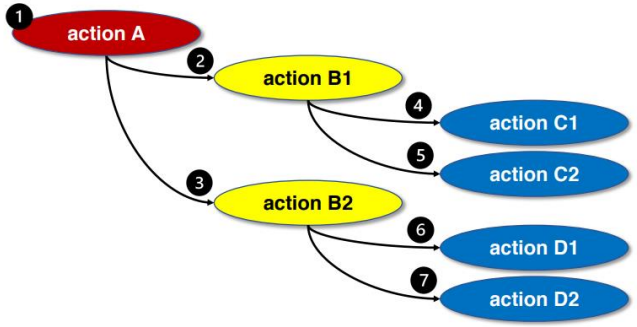


Fig. 8: Nested Inline Action Calls

C4 – Dataflow through Table

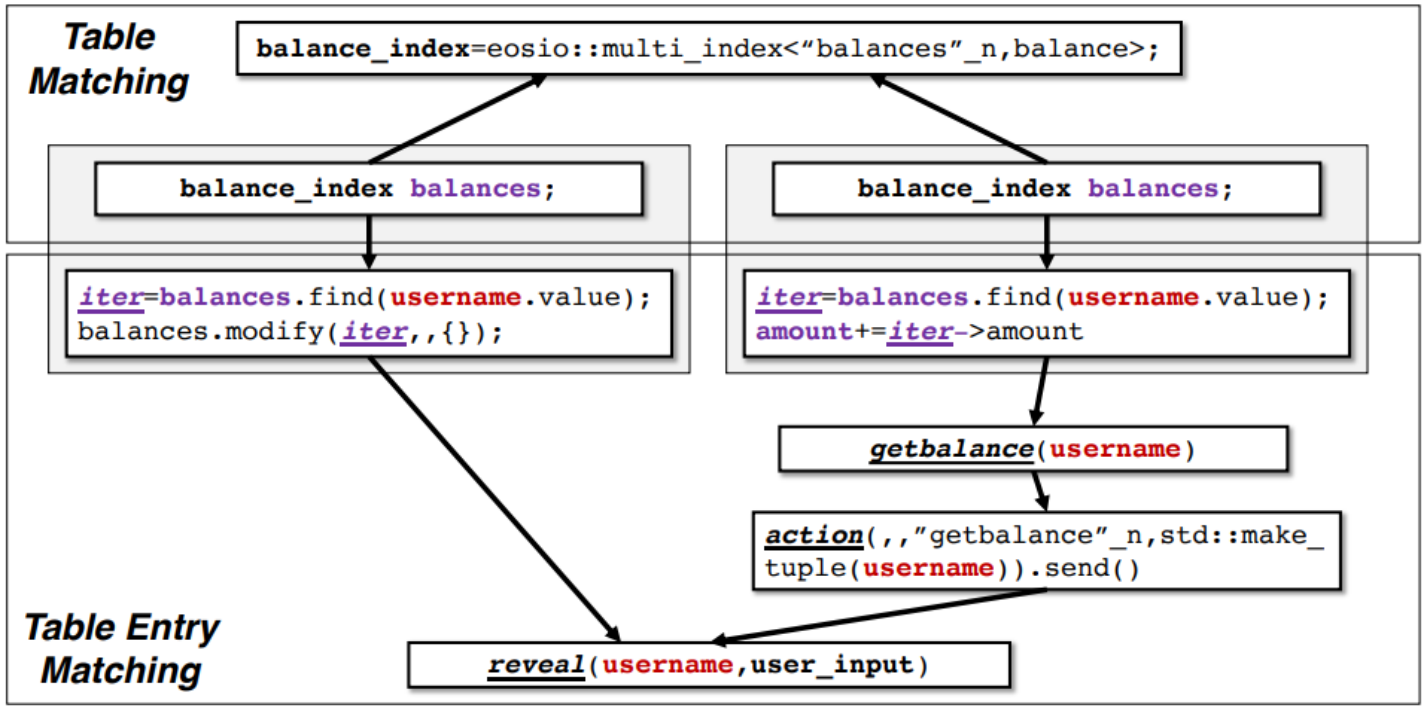


Fig. 9: Two-Level Matching for Table Accesses

Evaluation

Dataset	Sample Amount
Real-world binary EOSIO smart contracts	60,577
Rollback-vulnerable contracts from EOSAFE	715
Open-source EOSIO smart contracts	98

Evaluation

Dataset	Sample Amount	Experiment
Real-world binary EOSIO smart contracts	60,577	**E1**
Rollback-vulnerable contracts from EOSAFE	715	**E2**
Open-source EOSIO smart contracts	98	**E3**

- **E1**: Detect Groundhog Day Vulnerabilities in real world.
- **E2**: Compare VETEOS with EOSAFE as ground truth.
- **E3**: Evaluate static analysis accuracy.

E1 – Vulnerability Detection

TABLE II: Detecting Vulnerabilities by Steps

Factor	Semantics	# of Contracts
F1	`payToPlay` → `writeState` → `notify`	3,702/60,577
F2 & F1	`payToPlay` ⇝ `(user_input)` ⇝ `checkCondition` `createSecret` ⇝ `(secret)` ⇝ `checkCondition`	3,394/3,702
F3 & F2 & F1	`writeState` ⇝ `(global_state)` ⇝ `readState`	2,086/3,394
F4 & F3 & F2 & F1	`checkCondition` → `writeState`	735/2,086

E1 – Vulnerability Detection

TABLE II: Detecting Vulnerabilities by Steps

Factor	Semantics	# of Contracts
F1	payToPlay → writeState → notify	3,702/60,577
F2 & F1	payToPlay ⇝ (user_input) ⇝ checkCondition createSecret ⇝ (secret) ⇝ checkCondition	3,394/3,702
F3 & F2 & F1	writeState ⇝ (global_state) ⇝ readState	2,086/3,394
F4 & F3 & F2 & F1	checkCondition → writeState	735/2,086

10% generate over **2K** transactions daily.

1.3K average transactions for the top 2K contracts.

899K USD total balance can be directly affected.

Case Study

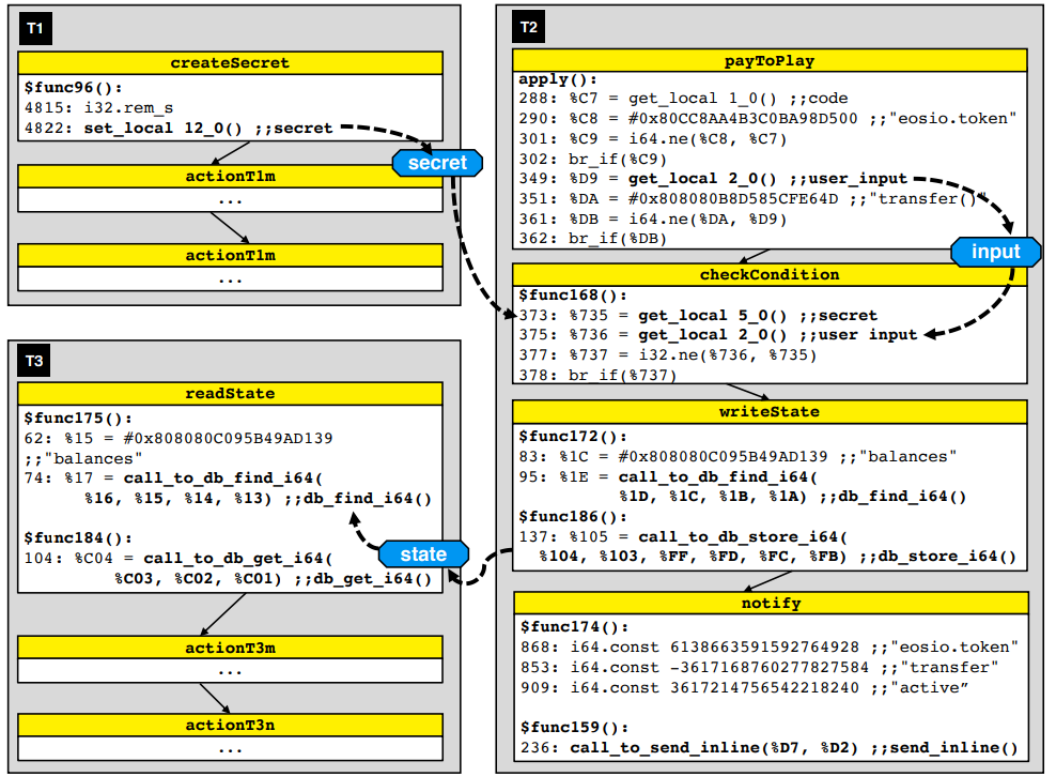


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Case Study

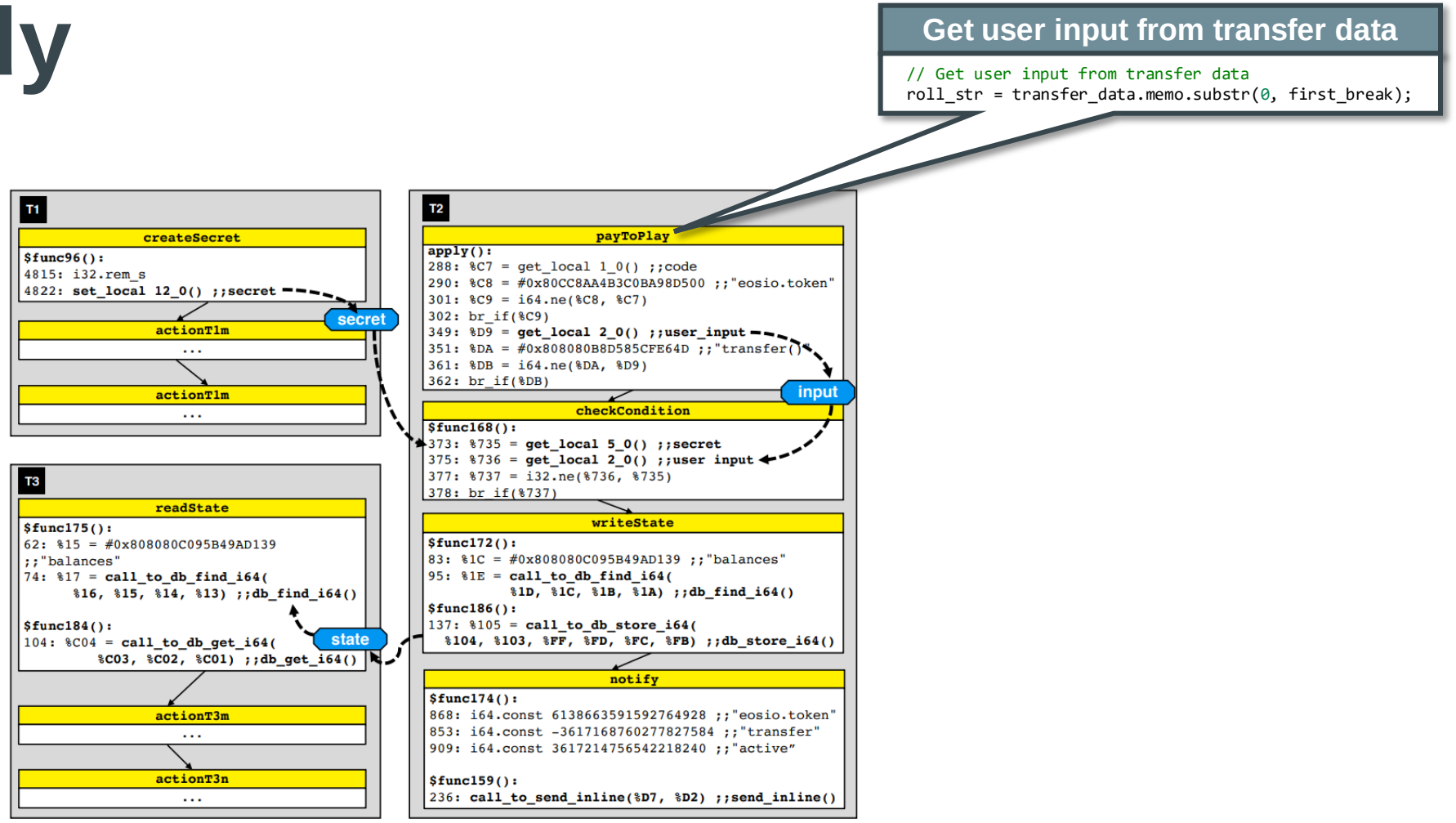


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Case Study

Create secret number

```
// Create secret number in [1,100]
const uint64_t random_roll = ((random_num_hash.hash[0]
+ random_num_hash.hash[1] + random_num_hash.hash[2] +
random_num_hash.hash[3] + random_num_hash.hash[4] +
random_num_hash.hash[5] + random_num_hash.hash[6] +
random_num_hash.hash[7]) % 100) + 1;
```

Get user input from transfer data

```
// Get user input from transfer data
roll_str = transfer_data.memo.substr(0, first_break);
```

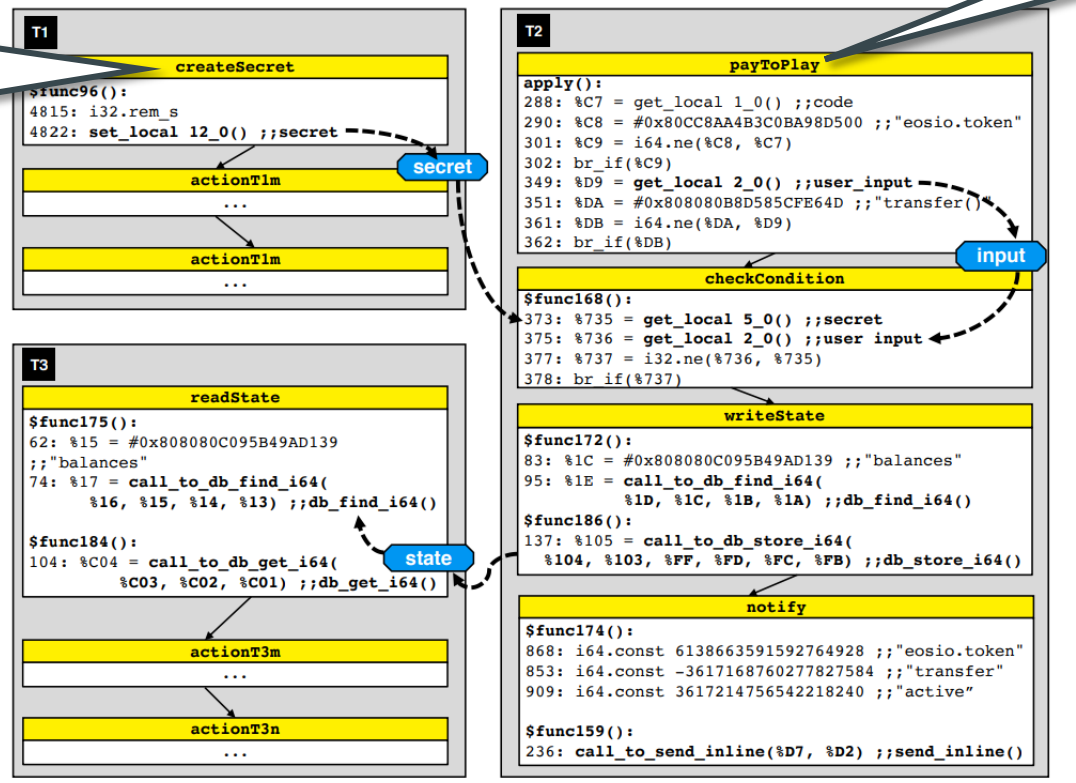


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Case Study

Create secret number

```
// Create secret number in [1,100]
const uint64_t random_roll = ((random_num_hash.hash[0]
+ random_num_hash.hash[1] + random_num_hash.hash[2] +
random_num_hash.hash[3] + random_num_hash.hash[4] +
random_num_hash.hash[5] + random_num_hash.hash[6] +
random_num_hash.hash[7]) % 100) + 1;
```

Get user input from transfer data

```
// Get user input from transfer data
roll_str = transfer_data.memo.substr(0, first_break);
```

Compare secret with user input

```
// Compare secret with user input
if(random_roll < activebets_itr->roll_under){
    payout = (activebets_itr->bet_amt *
    get_payout_mult_times10000(activebets_itr->
    roll_under, edge)) / 10000;
}
```

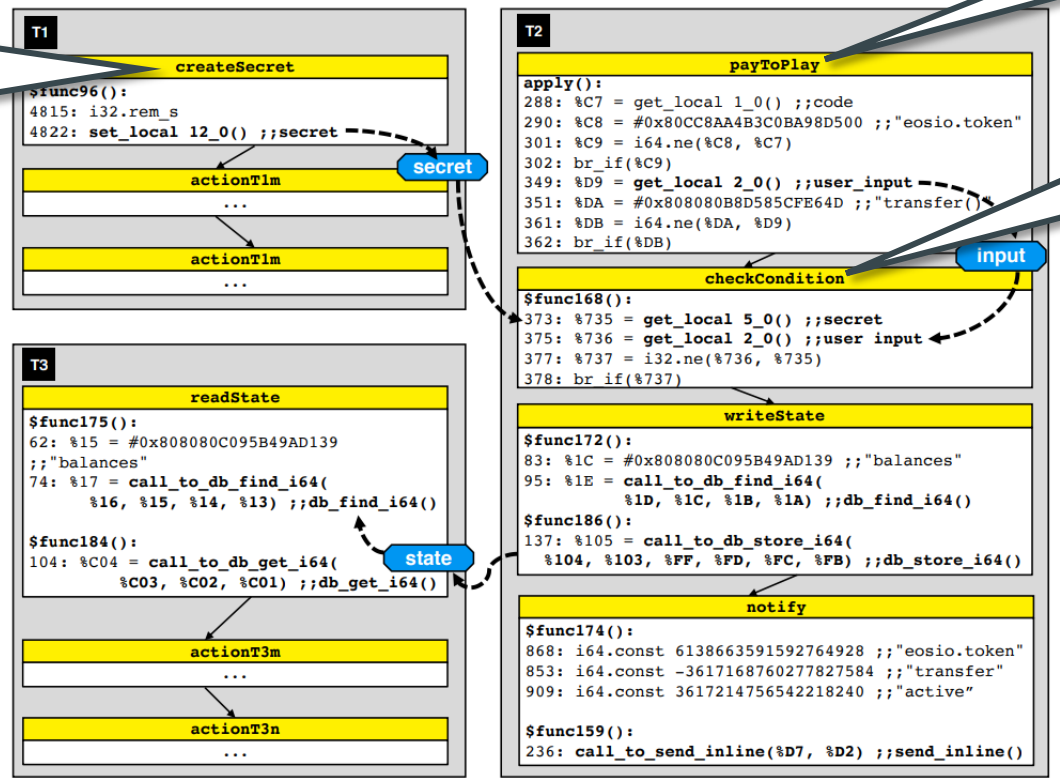


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Case Study

Create secret number

```
// Create secret number in [1,100]
const uint64_t random_roll = ((random_num_hash.hash[0]
+ random_num_hash.hash[1] + random_num_hash.hash[2] +
random_num_hash.hash[3] + random_num_hash.hash[4] +
random_num_hash.hash[5] + random_num_hash.hash[6] +
random_num_hash.hash[7]) % 100) + 1;
```

Get user input from transfer data

```
// Get user input from transfer data
roll_str = transfer_data.memo.substr(0, first_break);
```

Compare secret with user input

```
// Compare secret with user input
if(random_roll < activebets_itr->roll_under){
    payout = (activebets_itr->bet_amt *
    get_payout_mult_times10000(activebets_itr->
    roll_under, edge)) / 10000;
}
```

Update the global state

```
// Update the global state
increment_game_stats(activebets_itr->
bet_amt, payout);
```

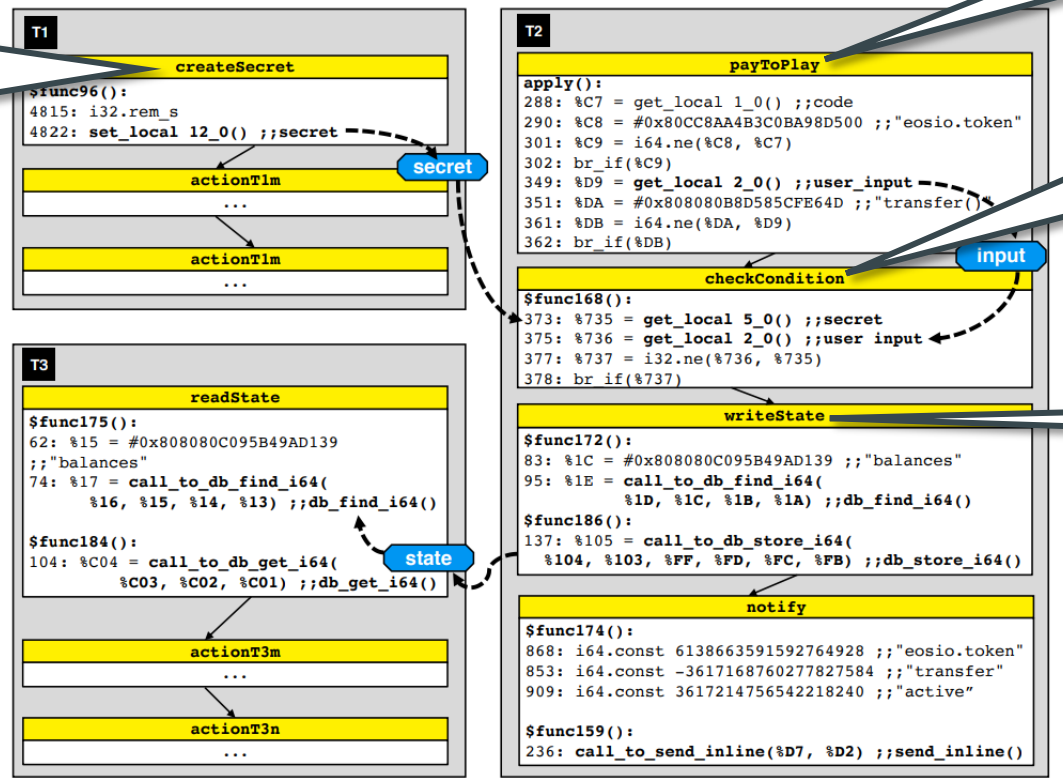


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Case Study

Create secret number

```
// Create secret number in [1,100]
const uint64_t random_roll = ((random_num_hash.hash[0]
+ random_num_hash.hash[1] + random_num_hash.hash[2] +
random_num_hash.hash[3] + random_num_hash.hash[4] +
random_num_hash.hash[5] + random_num_hash.hash[6] +
random_num_hash.hash[7]) % 100) + 1;
```

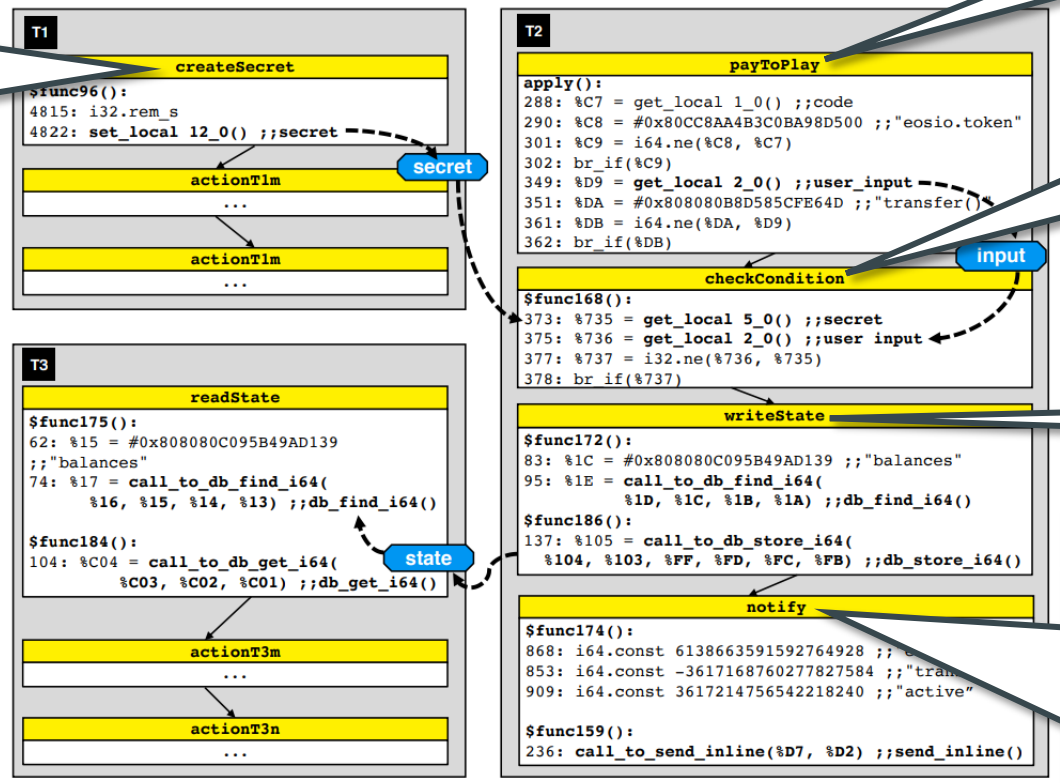


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Get user input from transfer data

```
// Get user input from transfer data
roll_str = transfer_data.memo.substr(0, first_break);
```

Compare secret with user input

```
// Compare secret with user input
if(random_roll < activebets_itr->roll_under){
    payout = (activebets_itr->bet_amt *
    get_payout_mult_times10000(activebets_itr->
    roll_under, edge)) / 10000;
}
```

Update the global state

```
// Update the global state
increment_game_stats(activebets_itr->
bet_amt, payout);
```

Send the rewards to the player

```
// Send the rewards to the player
action(
    permission_level{_self, N(active)},
    N(eosio.token),
    N(transfer),
    std::make_tuple(
        _self,
        activebets_itr->bettor,
        asset(payout, symbol_type(S(4, EOS))),
        std::string("Bet id: ") +
        std::to_string(bet_id) +
        std::string(" -- Winner! Play: dice.eosbet.io")
    )
).send();
```

IMAGINE

Case Study

Create secret number

```
// Create secret number in [1,100]
const uint64_t random_roll = ((random_num_hash.hash[0]
+ random_num_hash.hash[1] + random_num_hash.hash[2] +
random_num_hash.hash[3] + random_num_hash.hash[4] +
random_num_hash.hash[5] + random_num_hash.hash[6] +
random_num_hash.hash[7]) % 100) + 1;
```

Get user input from transfer data

```
// Get user input from transfer data
roll_str = transfer_data.memo.substr(0, first_break);
```

Compare secret with user input

```
// Compare secret with user input
if(random_roll < activebets_itr->roll_under){
    payout = (activebets_itr->bet_amt *
    get_payout_mult_times10000(activebets_itr->
    roll_under, edge)) / 10000;
}
```

Read state from the table

```
// Read global information from the table
auto activebets_itr = activebets.find( bet_id );
```

Update the global state

```
// Update the global state
increment_game_stats(activebets_itr->
bet_amt, payout);
```

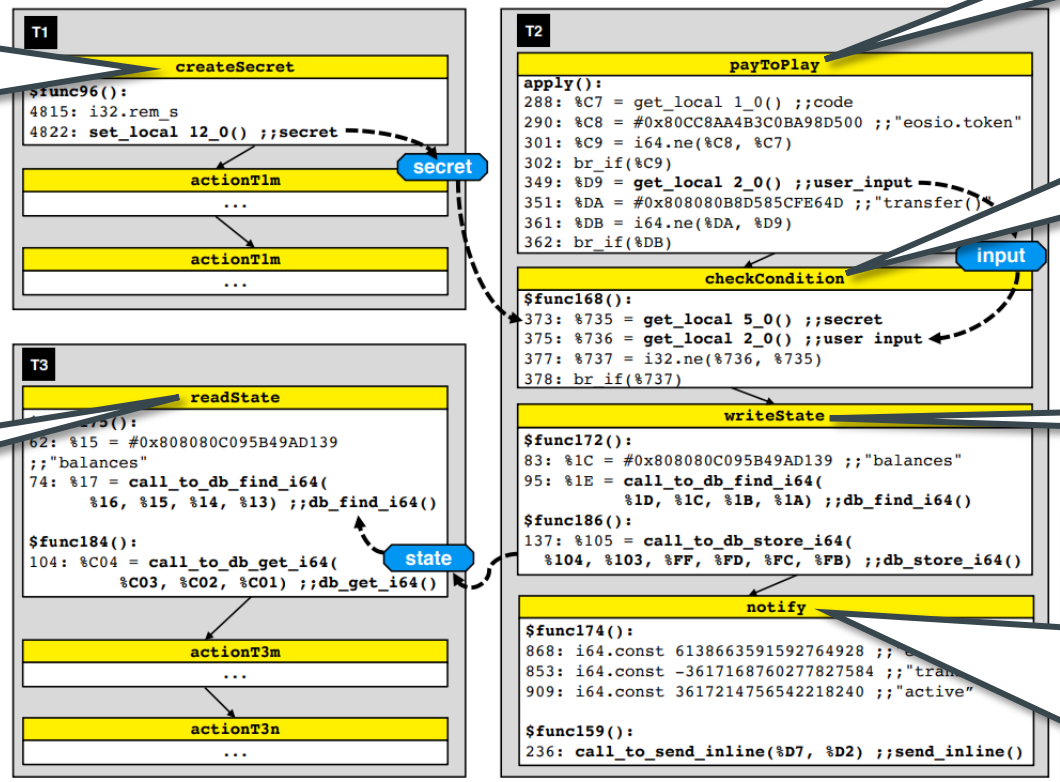


Fig. 15: Groundhog Day Vulnerability in *EOSBet Casino*

Send the rewards to the player

```
// Send the rewards to the player
action(
    permission_level{_self, N(active)},
    N(eosio.token),
    N(transfer),
    std::make_tuple(
        _self,
        activebets_itr->bettor,
        asset(payout, symbol_type(S(4, EOS))),
        std::string("Bet id: ") +
        std::to_string(bet_id) +
        std::string(" -- Winner! Play: dice.eosbet.io")
    )
).send();
```

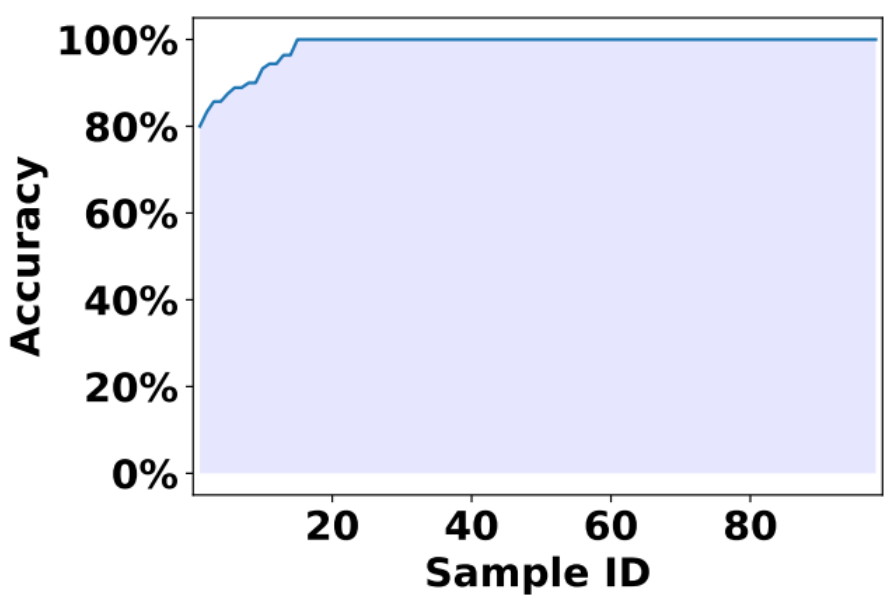
IMAGINE

E2 – Compared with EOSAFE

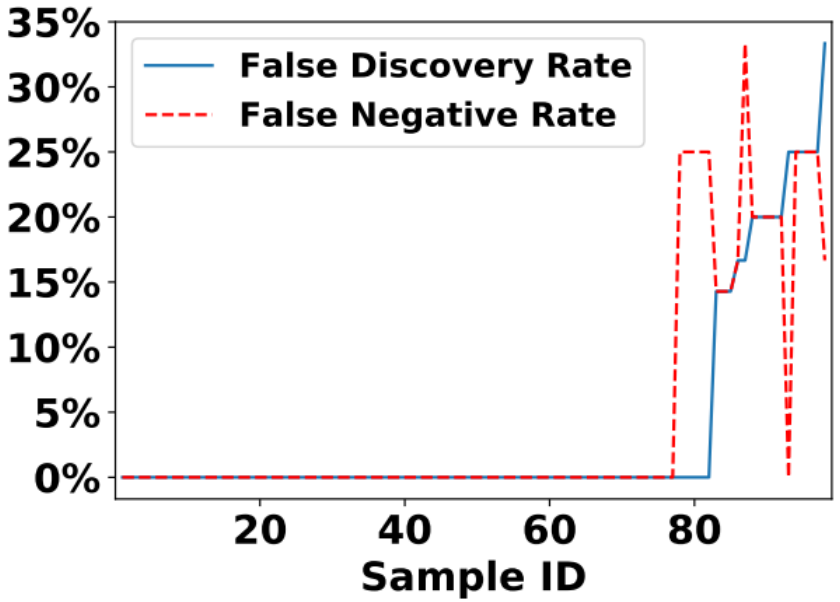
TABLE III: Identified Vulnerability Factors

Factor	EOSAFE	VETEOS
F1	715/715	715/715
F2 & F1	715/715	563/715
F3 & F2 & F1	NA	195/715
F4 & F3 & F2 & F1	NA	144/715

E3 – Static Analysis Accuracy



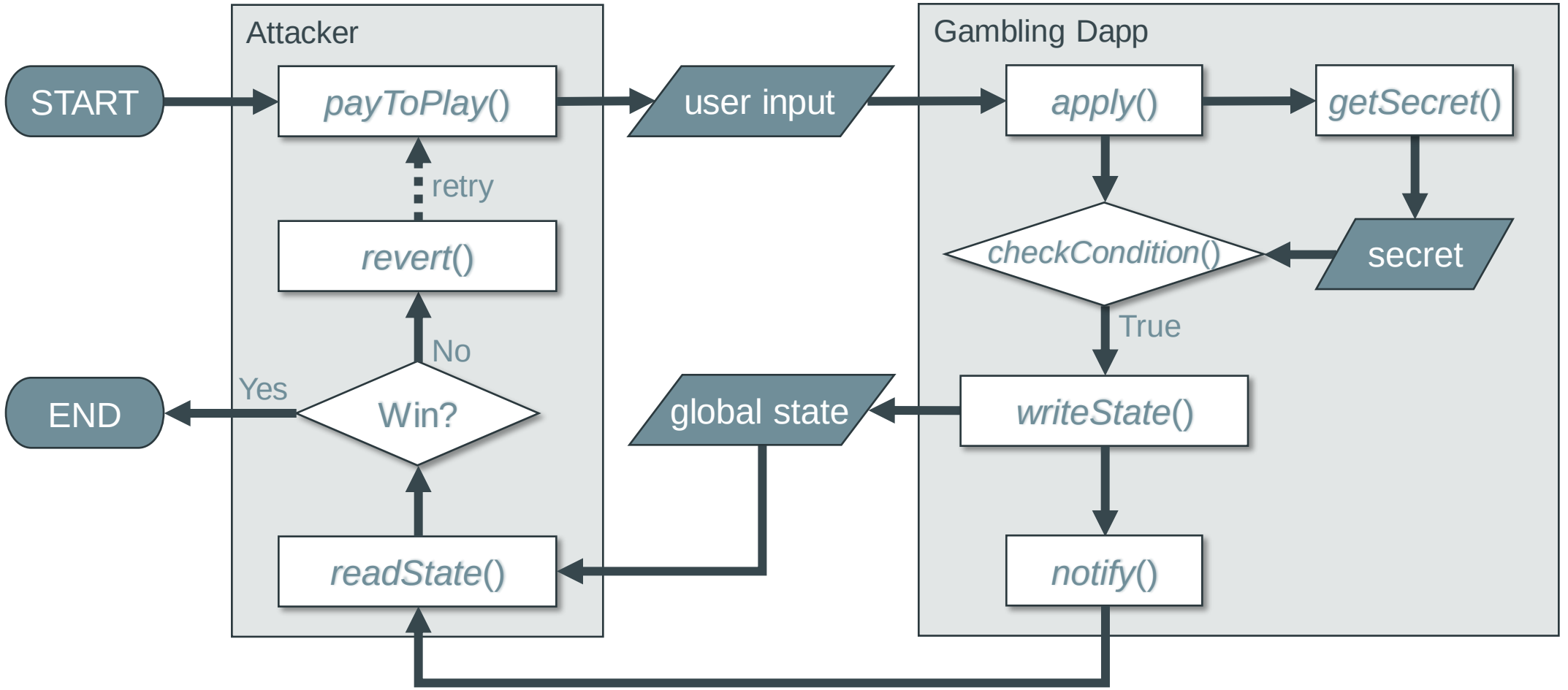
(a) Entry Point Discovery



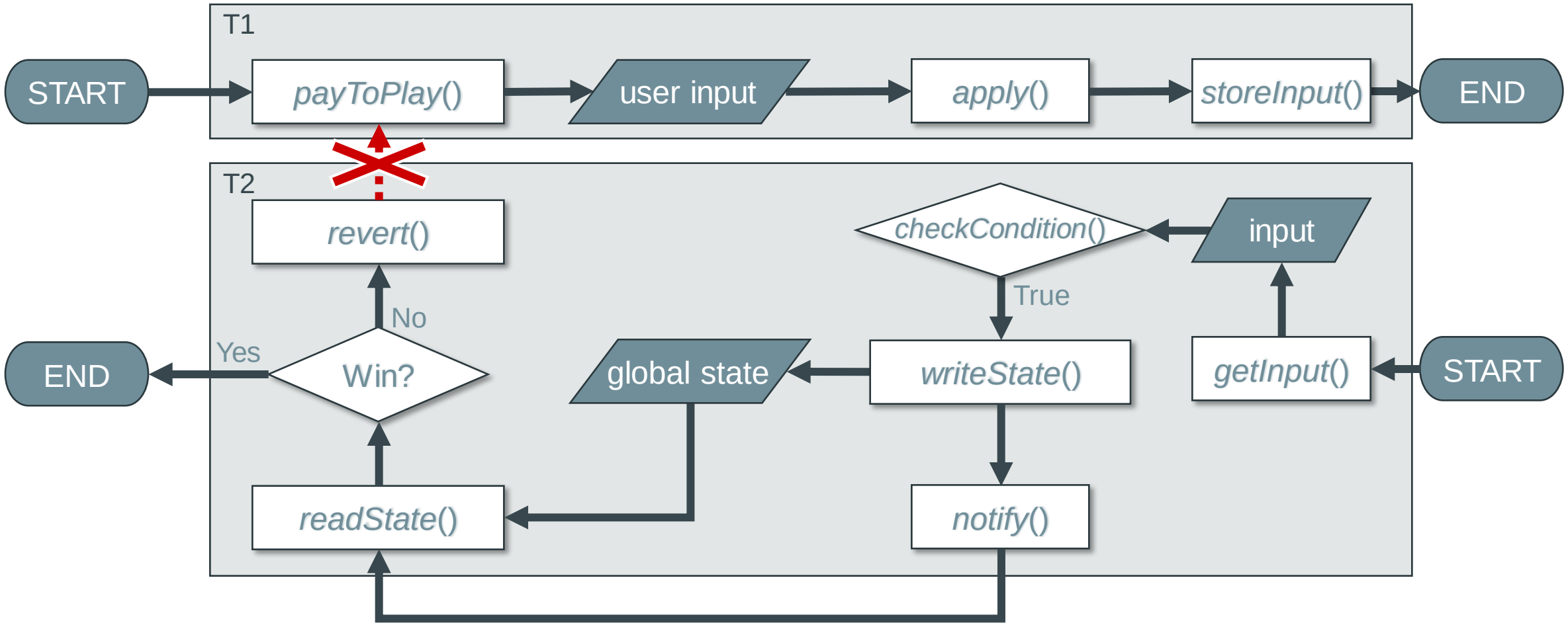
(b) Dataflow

Fig. 12: Accuracy of Static Analysis

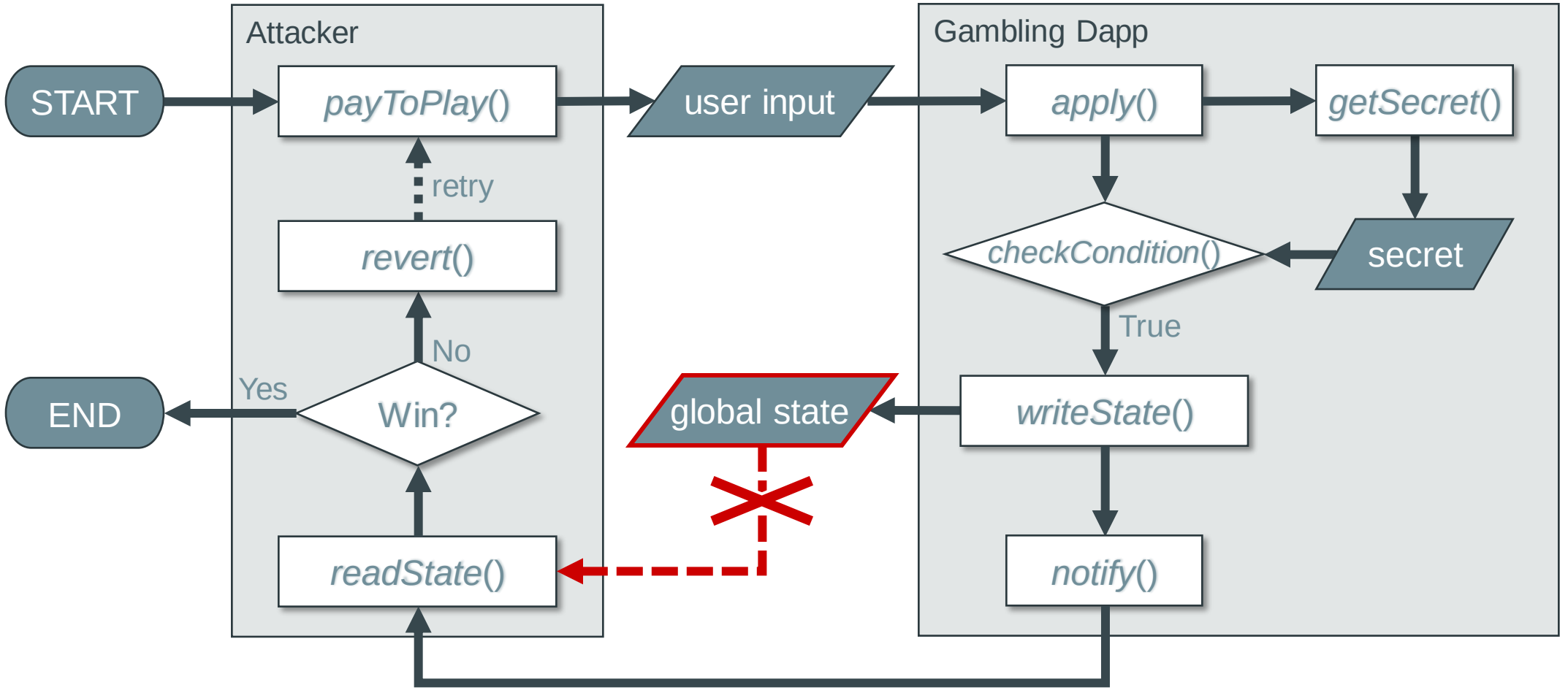
Mitigation


Attacker
Gambling Dapp
START
payToPlay()
user input
apply()
getSecret()
retry
revert()
checkCondition()
secret
True
No
Yes
END
Win?
global state
writeState()
readState()
notify()

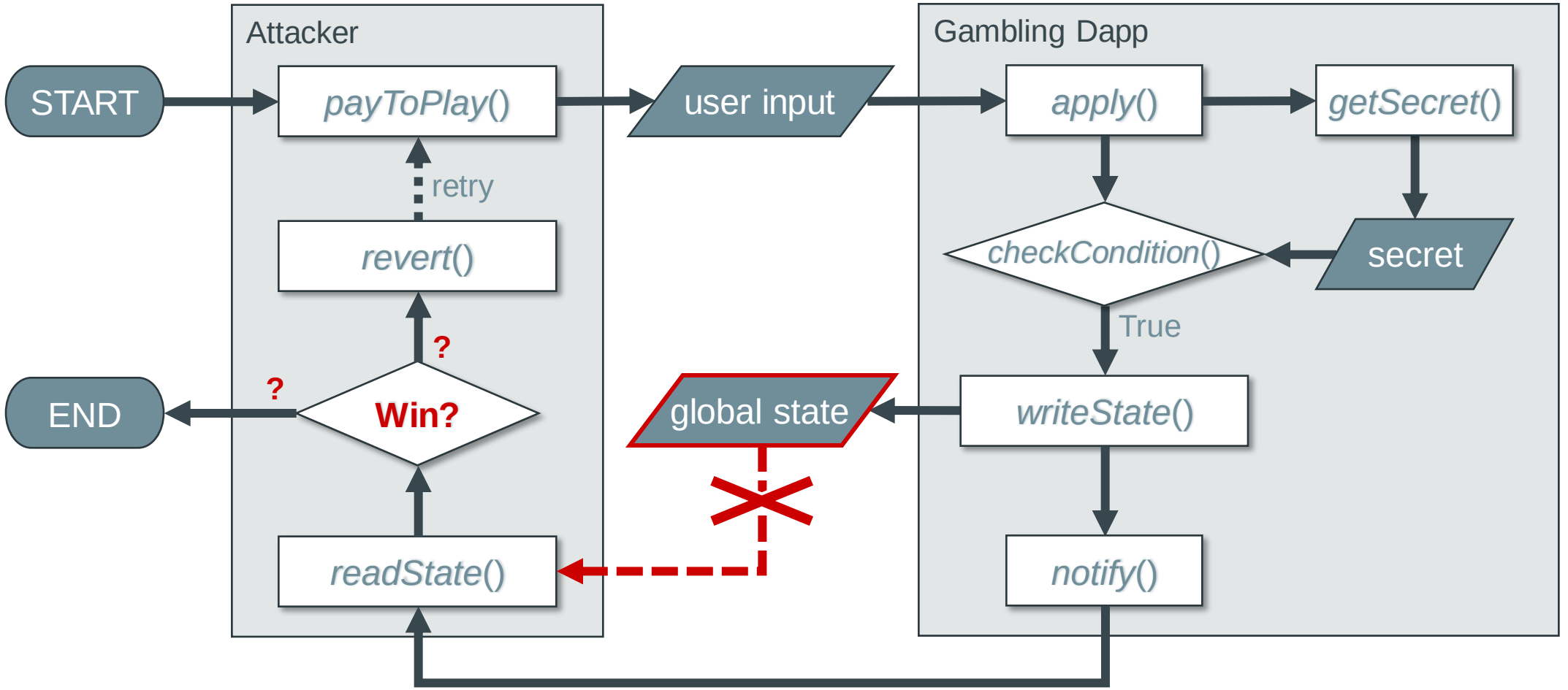
Mitigation – Control Flow



Mitigation – Data Flow



Mitigation – Data Flow



Conclusion

- We propose **VETEOS**, a **static vetting tool** for the Groundhog Day Vulnerabilities in EOSIO contracts.
- We formally define the unique **Groundhog Day Vulnerability** as a **control and data dependency problem**.
- We addresses multiple distinct challenges for **analyzing EOSIO WASM programs**.
- We have detected **735 new vulnerabilities** in the wild.

THANK YOU