Smart Contract Security

Smart Contracts

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)

What can go wrong when programs handle billions of USD?

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)
- Executed on the Ethereum blockchain

What can go wrong when programs handle billions of USD?

Smart Contracts

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)
- Executed on the Ethereum blockchain
- Written in high-level languages (*e.g.*, Solidity)

What can go wrong when programs handle billions of USD?

Smart Contracts

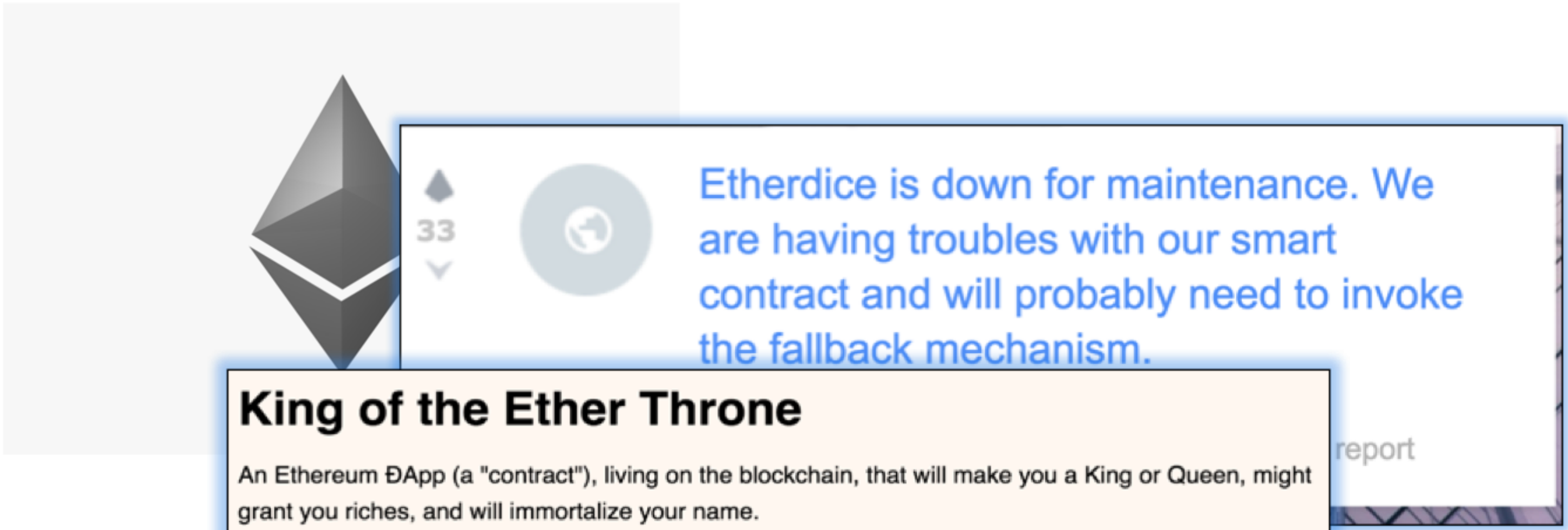
```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
            msg.sender.call.value(balance)();
    balance = 0;
} }
```

- Small programs that ***handle money*** (ether)
- Executed on the Ethereum blockchain
- Written in high-level languages (*e.g.*, Solidity)
- ***No patching*** after release

What can go wrong when programs handle billions of USD?

Funds Stolen From the DAO One Year Ago Would be Worth $1.35bn Today

JP Buntinx June 18, 2017 Crypto, News



Security Bug #1: Reentrancy

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

withdraw()

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

withdraw()

10 ether

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

withdraw()

10 ether

Later…

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

withdraw()

10 ether

Later…

withdraw()

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

withdraw()

10 ether

Later…

withdraw()

no transfer

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

withdraw()

10 ether

Later…

withdraw()

no transfer

Can the user contract withdraw more than 10 ether?

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
```

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
function () payable {
  // log payment
}
```

Wallet Contract

```
uint balance = 10

function withdr
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

calls the default "payable" function

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
function () payable {
  // log payment
}
```

Wallet Contract

```
uint balance = 10

function withdr
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

calls the default "payable" function

balance is zeroed ***after*** ether transfer

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
function () payable {
  wallet.withdraw();
}
```

Calls withdraw() before balance is set to 0

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

balance is zeroed ***after*** ether transfer

An adversary stole 3.6M Ether !

Security Bug #2: Unprivileged write to storage

Security Bug #2: Unprivileged write to storage

Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint amount) {
  if (msg.sender == owner) {
    owner.send(amount);
  }
}
```

Security Bug #2: Unprivileged write to storage

Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint amount) {
  if (msg.sender == owner) {
    owner.send(amount);
  }
}
```

Only owner can send ether

Security Bug #2: Unprivileged write to storage

Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint amount) {
  if (msg.sender == owner) {
    owner.send(amount);
  }
}
```

Any user may change the wallet's owner

Only owner can send ether

Security Bug #2: Unprivileged write to storage

Any user may change the wallet's owner

Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint amount) {
  if (msg.sender == owner) {
    owner.send(amount);
  }
}
```

Only owner can send ether

An attacker used a similar bug to ***steal $32M***

More security bugs

Unexpected ether flows

Insecure coding, such as unprivileged writes *(e.g., Multisig Parity bug)*

Use of unsafe inputs (e.g., reflection, hashing, …)

Reentrant method calls *(e.g., DAO bug)*

More security bugs

Unexpected ether flows

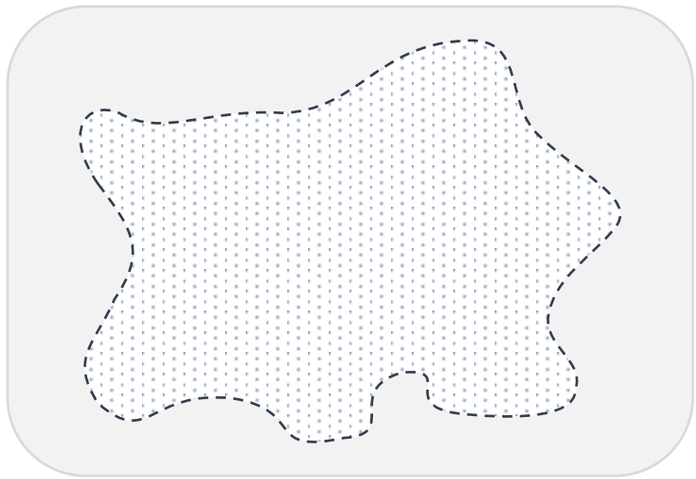
Insecure coding, such as unprivileged writes *(e.g., Multisig Parity bug)*

Use of unsafe inputs (e.g., reflection, hashing, …)

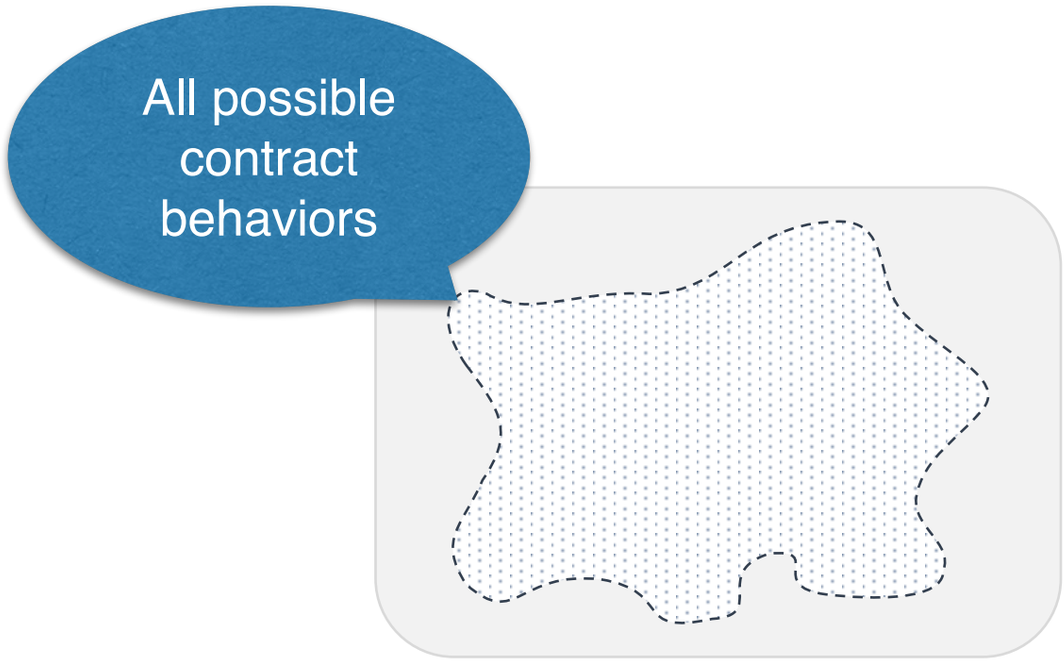
Reentrant method calls *(e.g., DAO bug)*

Transaction reordering

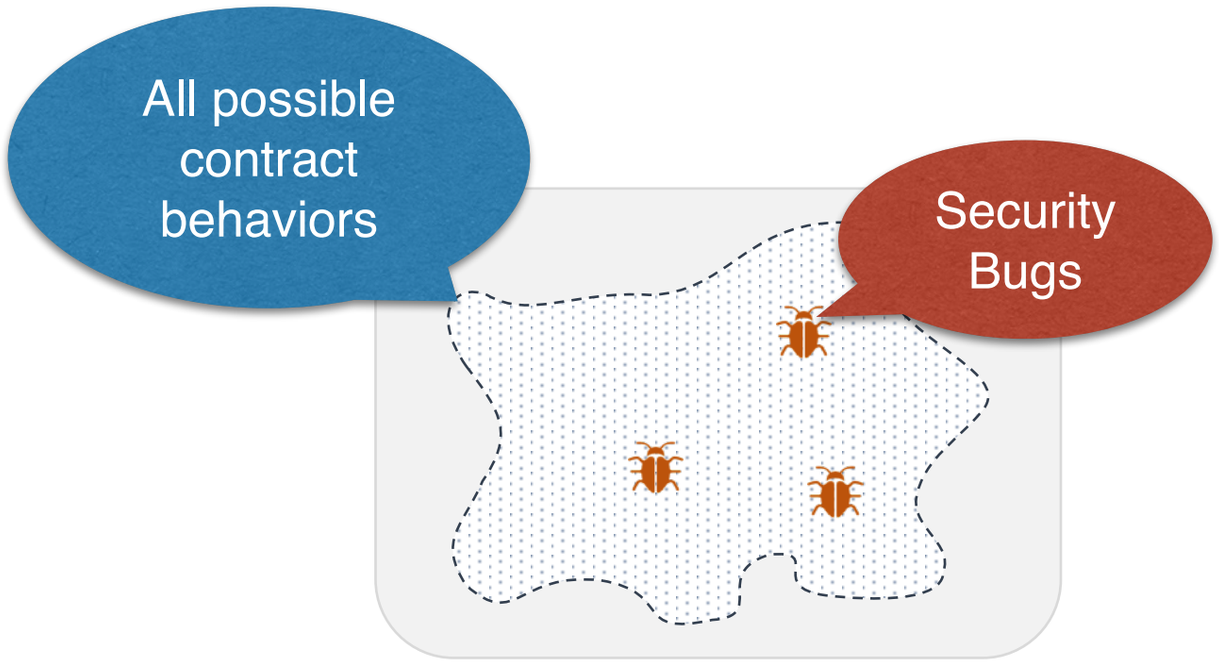
Automated Security Analysis



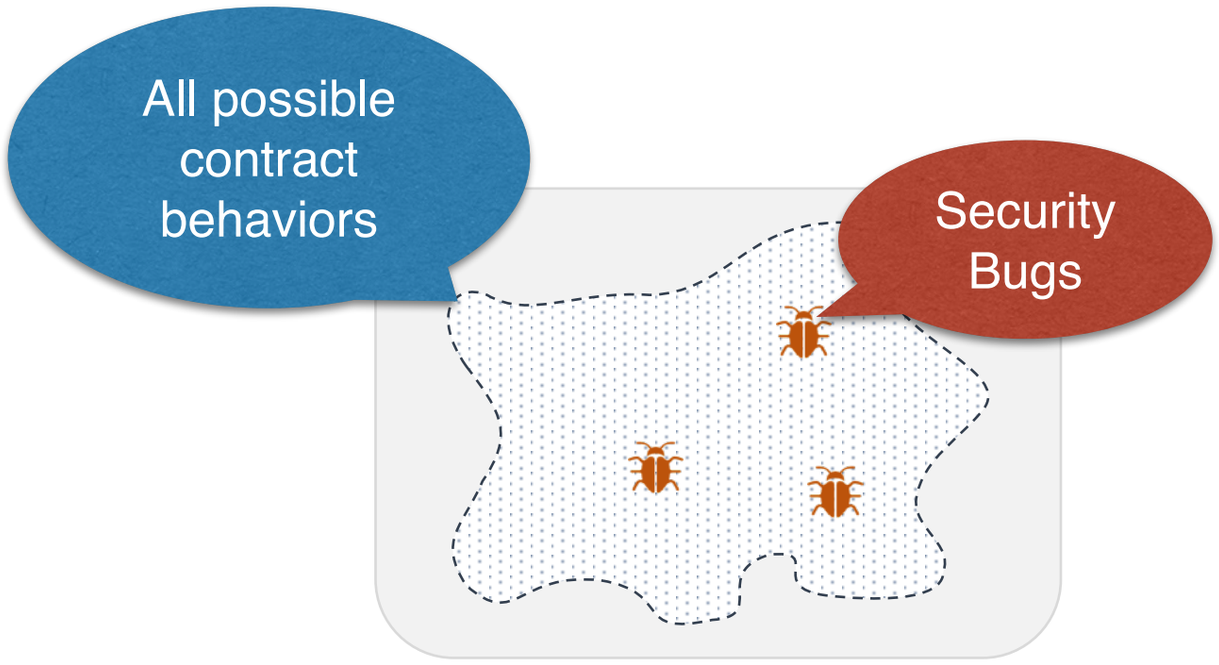
Automated Security Analysis



Automated Security Analysis



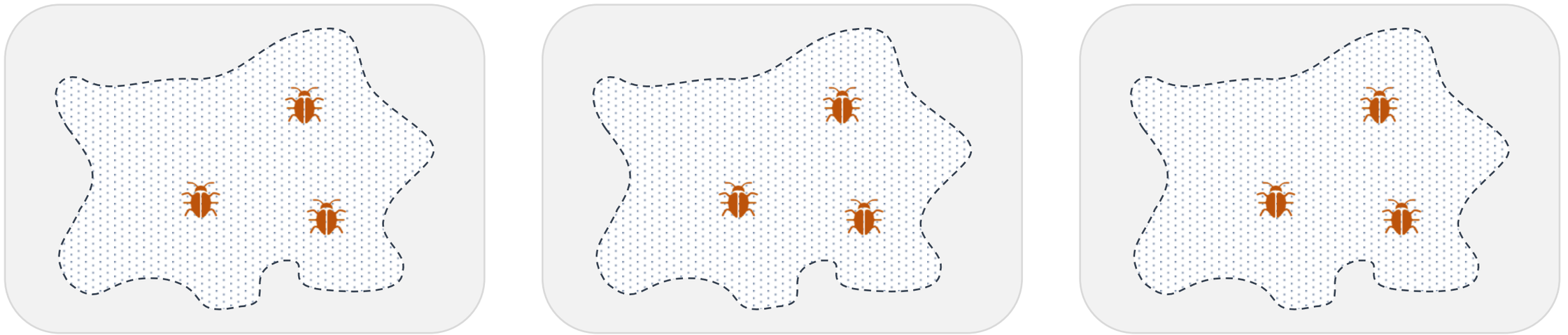
Automated Security Analysis



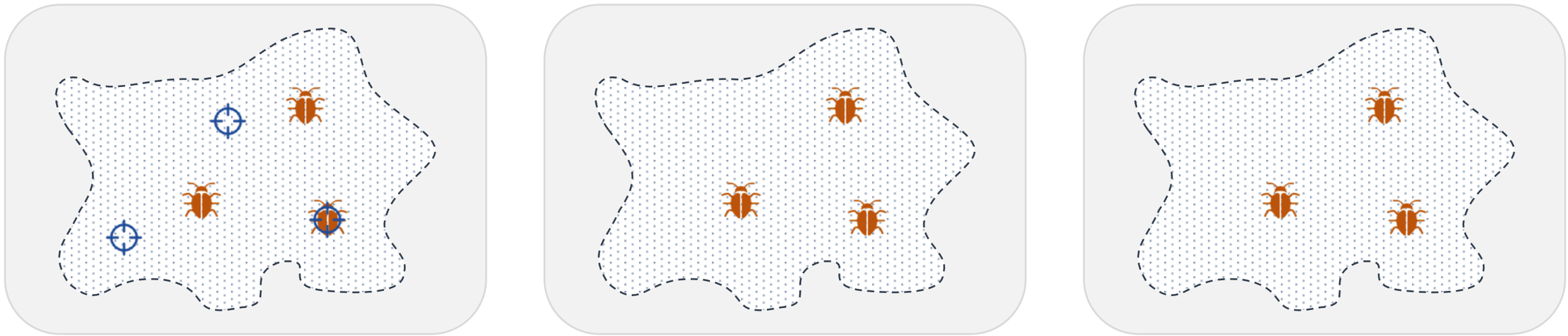
Problem: Cannot enumerate all possible contract behaviors…

Automated Security Analysis: Existing Solutions

Automated Security Analysis: Existing Solutions

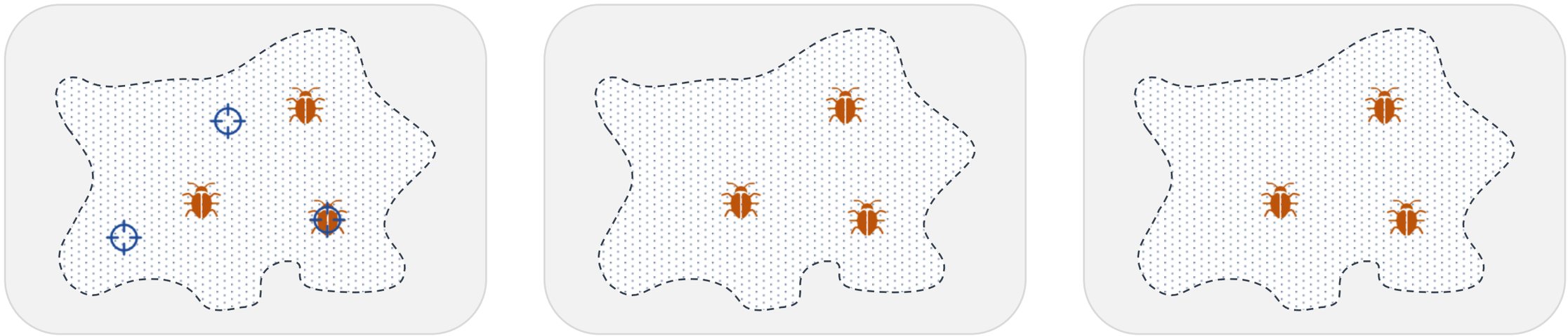


Automated Security Analysis: Existing Solutions



- Testing

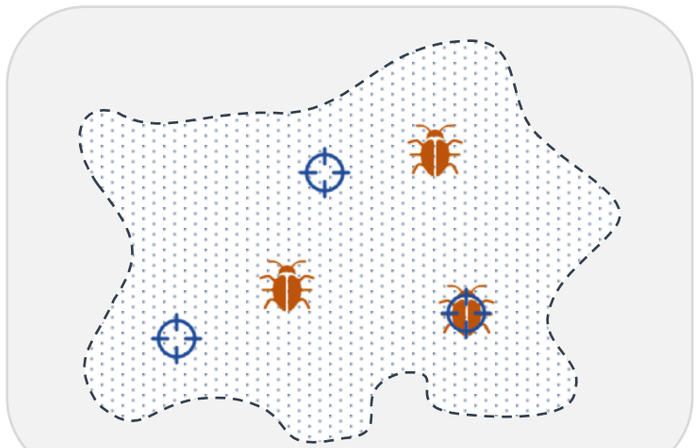
Automated Security Analysis: Existing Solutions



- Testing

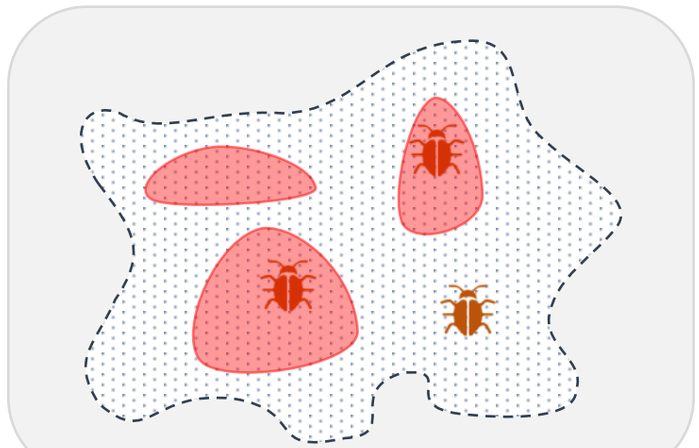
  Very limited guarantees

Automated Security Analysis: Existing Solutions

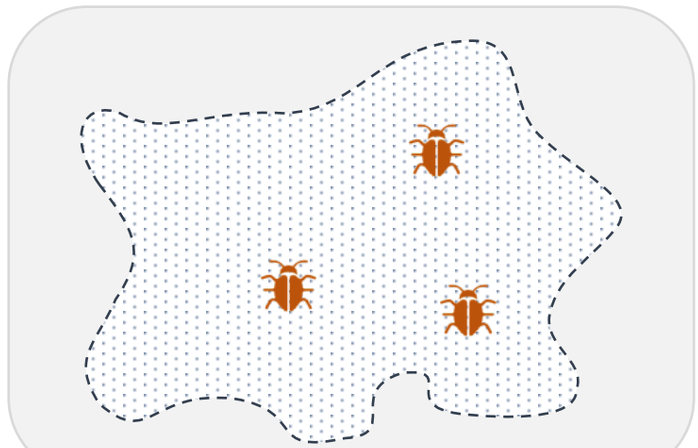


- Testing

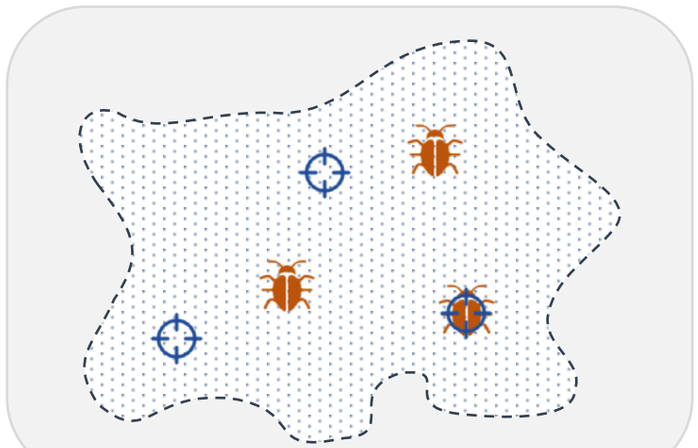
  Very limited guarantees



- Dynamic analysis
- Symbolic execution

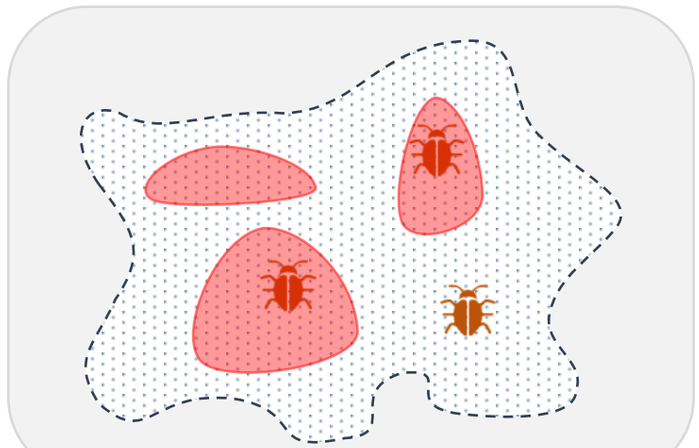


Automated Security Analysis: Existing Solutions



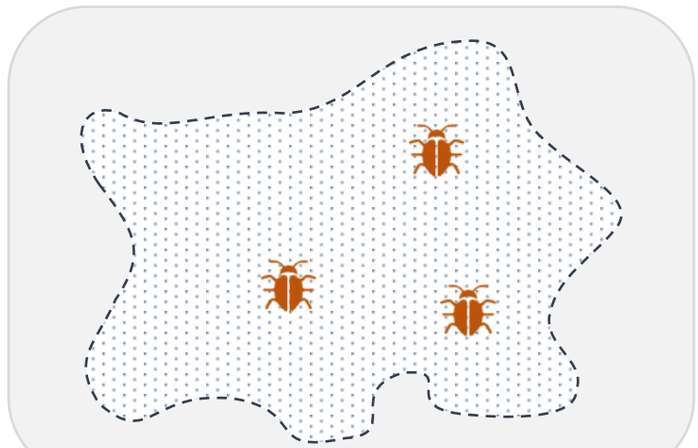
- Testing

Very limited guarantees

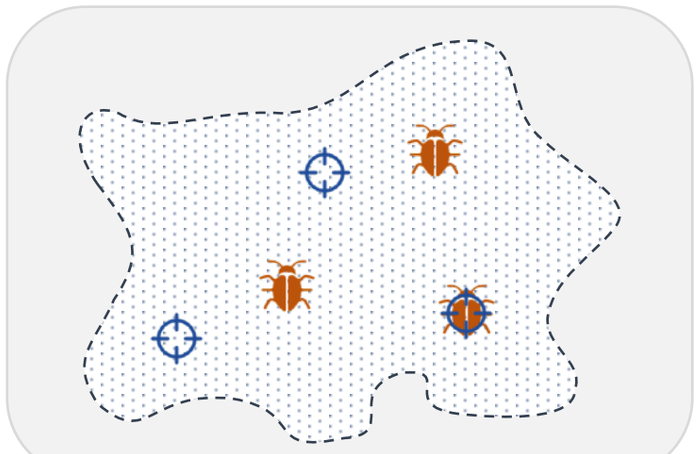


- Dynamic analysis
- Symbolic execution

Better than testing, but
can still miss vulnerabilities

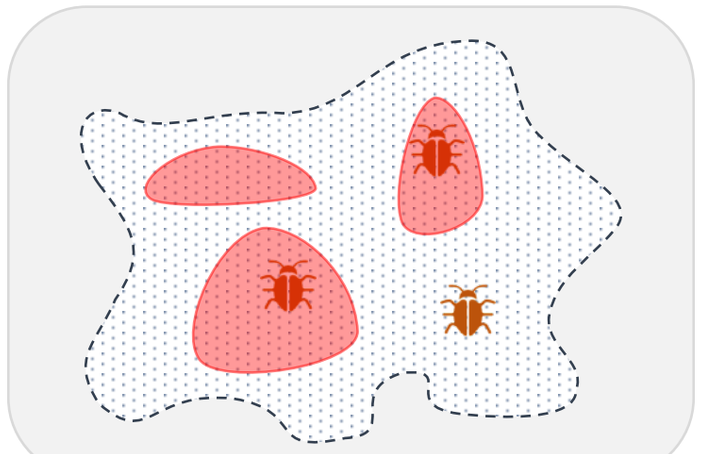


Automated Security Analysis: Existing Solutions



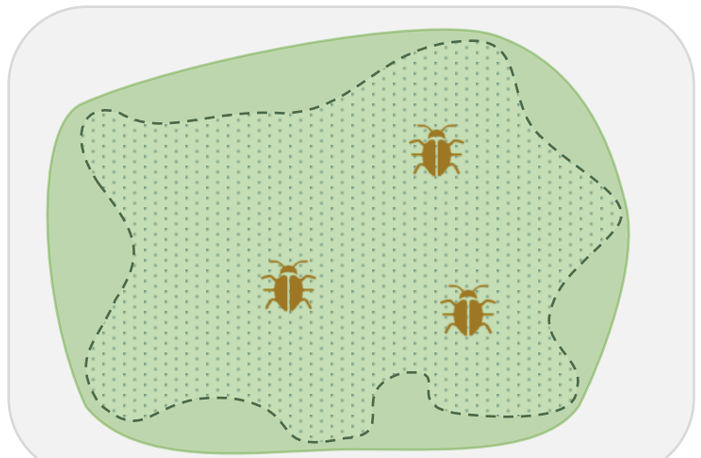
- Testing

Very limited guarantees



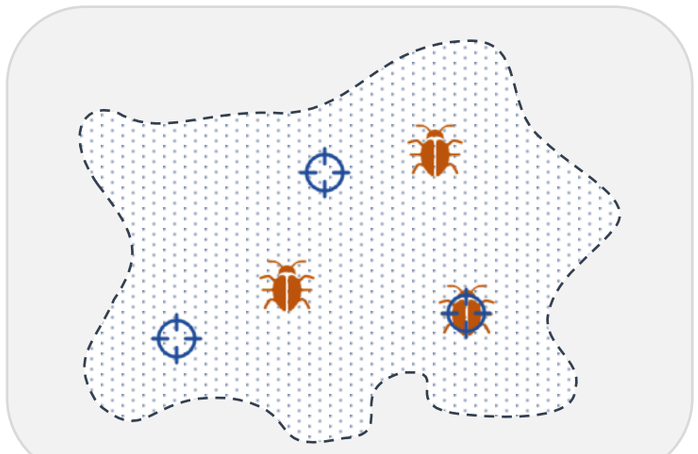
- Dynamic analysis
- Symbolic execution

Better than testing, but
can still miss vulnerabilities



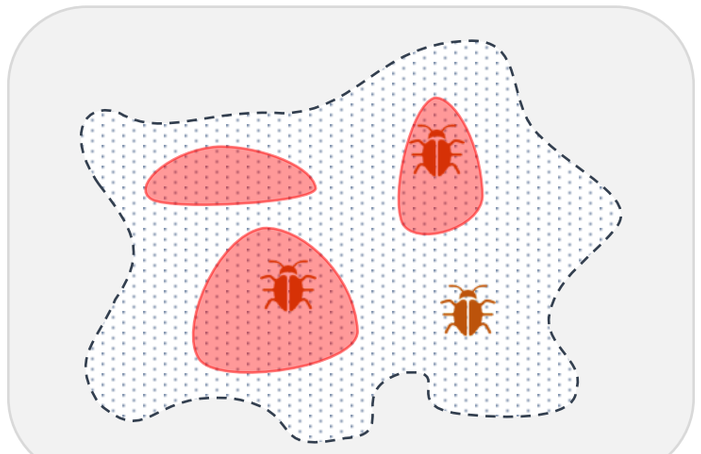
- Static analysis
- Formal verification

Automated Security Analysis: Existing Solutions



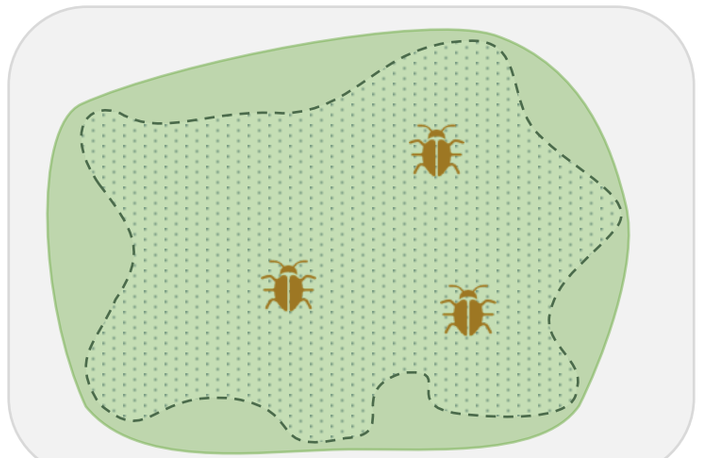
- Testing

Very limited guarantees



- Dynamic analysis
- Symbolic execution

Better than testing, but
can still miss vulnerabilities



- Static analysis
- Formal verification

Strong guarantees