Security Test Cases

Boosting confidence in code security

Stephan van Staden

Motivation

- Engineering secure software applications

Motivation

- Engineering secure software applications
- Capability-secure languages

Motivation

- Engineering secure software applications
- Capability-secure languages
- Security a property of an evolving project

Motivation

- Engineering secure software applications
- Capability-secure languages
- Security a property of an evolving project
- Security test cases can boost confidence in code

Revocable proxy/Caretaker (Version 1)

```
def makeCaretaker(var target) {
    def caretaker {
        match [verb, args] {
            E.call(target, verb, args)
        }
    }
    def revoker {
        to revoke() { target := null }
    }
    return [caretaker, revoker]
}
```

Usage scenario

```
def myfile := makeFile(filename)
def [caretaker, revoker] := makeCaretaker(myfile)
def alice := makeUser("alice")
alice.useResource(caretaker)
.
.  # alice can access myfile
.
revoker.revoke()
.
.  # alice should not be able to access myfile anymore
.
```

Test case (for Version 1)

```
def caretakerTestCase {
    to setUp() { ... }
    to tearDown() { ... }
    ...
    to testCaretaker() {
        def myfile := makeFile(filename)
        def [caretaker, revoker] := makeCaretaker(myfile)
        def alice := makeUser("alice")
        alice.useResource(caretaker)
        assertaccess(alice, myfile)
        revoker.revoke()
        assertnoaccess(alice, myfile)
    }
    ...
}
```

Revocable proxy/Caretaker (Version 2)

```
def makeCaretaker(target) {
    var enabled := true
    def caretaker {
        match [verb, args] {
            if (enabled) {
                E.call(target, verb, args)
            } else {
                throw("disabled")
            }
        }
    }
    def gate {
        to enable() { enabled := true }
        to disable() { enabled := false }
    }
    return [caretaker, gate]
}
```

Test case (for Version 2)

```
def caretakerTestCase {
    to setUp() { ... }
    to tearDown() { ... }
    ...
    to testCaretaker() {
        def myfile := makeFile(filename)
        def [caretaker, gate] := makeCaretaker(myfile)
        def alice := makeUser("alice")
        alice.useResource(caretaker)
        assertaccess(alice, myfile)
        gate.disable()
        assertnoaccess(alice, myfile)
        gate.enable()
        assertaccess(alice, myfile)
    }
    ...
}
```

Factors influencing security

1. Object connectivity graph

Factors influencing security

1. Object connectivity graph

2. Control flow

Factors influencing security

1. Object connectivity graph
2. Control flow, influenced by
   - Code

Factors influencing security

1. Object connectivity graph

2. Control flow, influenced by
   - Code
   - Data

Problems

- Undecidability of the access problem

Problems

- Undecidability of the access problem
- Forced to use safe approximations

Problems

- Undecidability of the access problem
- Forced to use safe approximations
- Potential intractablility of solutions

Problems

- Undecidability of the access problem
- Forced to use safe approximations
- Potential intractablility of solutions
- Tradeoff: closeness of approximation vs. efficiency

Tools

- Object connectivity graph

Tools

- Object connectivity graph
- Control flow analysis

Tools

- Object connectivity graph
- Control flow analysis
- Pre- & postconditions

Tools

- Object connectivity graph
- Control flow analysis
- Pre- & postconditions
- Abstract interpretation

Tools

- Object connectivity graph
- Control flow analysis
- Pre- & postconditions
- Abstract interpretation
- Situation calculus

Tools

- Object connectivity graph
- Control flow analysis
- Pre- & postconditions
- Abstract interpretation
- Situation calculus
- Constraint programming