Bringing Object-orientation to Security Programming

Mark S. Miller and the Cajadores
Overview: Bottom up by Layers

Composing Networks of Games
Smart Contracts as Games
Dimensions & Taxonomy of Electronic Rights
Patterns of Safe Cooperation
Access Abstractions and Compositions
Object-capabilities (ocaps)
Objects, References, Messages
How do I designate thee?

by Introduction
ref to Carol
ref to Bob
decides to share
by Parenthood
by Endowment
by Initial Conditions

How might object Bob come to know of object Carol?
How do I designate thee?

Alice says: `bob.foo(carol)`

*by Introduction*
- ref to Carol
- ref to Bob
decides to share

by Parenthood
by Endowment
by Initial Conditions
How do I designate thee?

Alice says: `bob.foo(carol)`

by *Introduction*
- *Ref to Carol*
  - Ref to Bob
  - Decides to share

by *Parenthood*
- *Endowment*
- *Initial Conditions*
How do I designate thee?

Alice says: \texttt{bob.foo(carol)}

\textit{by Introduction}

\textit{ref to Carol}

\textit{ref to Bob}

decides to share

\textit{by Parenthood}

\textit{by Endowment}

\textit{by Initial Conditions}
How do I designate thee?

Alice says: \texttt{bob.foo(carol)}

\textit{by Introduction}
\textit{ref to Carol}
\textit{ref to Bob}
\textit{decides to share}

\textit{by Parenthood}
\textit{by Endowment}
\textit{by Initial Conditions}
How do I designate thee?

Alice says: bob.foo(carol)

by Introduction
ref to Carol
ref to Bob
decides to share

by Parenthood
by Endowment
by Initial Conditions
How do I designate thee?

Bob says: `var carol = { ... };`

by Introduction
ref to Carol
ref to Bob
decides to share

by Parenthood
by Endowment
by Initial Conditions
Alice says:  \texttt{var bob = \{ ... carol ... \};}

by Introduction
ref to Carol
ref to Bob
decides to share
by Parenthood
\textbf{by Endowment}
by Initial Conditions
How do I designate thee?

At $t_0$:

by Introduction
ref to Carol

ref to Bob
decides to share

by Parenthood

by Endowment

*by Initial Conditions*
OCaps: Small step from pure objects

- Memory safety and encapsulation
- Effects *only* by using held references
- No powerful references by default
OCaps: Small step from pure objects

- Memory safety and encapsulation
  - Effects *only* by using held references
  - No powerful references by default

Reference graph ≡ Access graph
- Only connectivity begets connectivity
- OO expressiveness for security patterns
Objects as Closures

```javascript
function makeCounter() {
    var count = 0;
    return {
        incr: function() { return ++count; },
        decr: function() { return --count; }
    };
}
```
Objects as Closures

function makeCounter() {
  var count = 0;
  return {
    incr: function() { return ++count; },
    decr: function() { return --count; }
  };
}

A record of closures hiding state is a fine representation of an object of methods hiding instance vars
"use strict";
function makeCounter() {
    var count = 0;
    return def({
        incr: function() { return ++count; },
        decr: function() { return --count; }
    });
}

A tamper-proof record of lexical closures encapsulating state is a defensive object.
Turning EcmaScript 5 into SES

<script src="initSES.js"></script>

- Monkey patch away bad non-std behaviors
- Remove non-whitelisted primordials
- Install leaky WeakMap emulation
- Make virtual global root
- Freeze whitelisted global variables
- Replace `eval` & `Function` with safe alternatives
- Freeze accessible primordials
function makeFnCaretaker(target) {
    return def({
        wrapper: function(…args) {
            return target(…args);
        },
        revoke: function() { target = null; }
    });
}
Unconditional Access

Alice says:

```
bob.foo(carol);
```

Grants Bob full access to Carol forever
Revocability ≡ Temporal attenuation

Alice says:
```
var ct = makeCaretaker(carol);
bob.foo(ct.wrapper);
```
Revocability ≡ Temporal attenuation

Alice

var ct = makeCaretaker(carol);
bob.foo(ct.wrapper);
//…

Bob

Carol

target

revoke

wrapper
Alice says:
var ct = makeCaretaker(carol);
bob.foo(ct.wrapper);
//…
ct.revoke();
Revocability ≡ Temporal attenuation

Alice says:
```javascript
var ct = makeCaretaker(carol);
bob.foo(ct.wrapper);
//…
ct.revoke();
```

Carol
Attenuators ≡ Access Abstractions

Alice says:
```javascript
var ct = makeCaretaker(carol);
bob.foo(ct.wrapper);
```

Express security policy by the behavior of the objects you provide.
Abstractions extend vocabulary

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Abstraction Forms</th>
<th>Extended Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, ., []</td>
<td><em>procedural abstraction</em></td>
<td>foo(bar, baz), ...</td>
</tr>
<tr>
<td>int, struct, array</td>
<td><em>data abstraction</em></td>
<td>Point, Window, ...</td>
</tr>
<tr>
<td>if, while, switch</td>
<td><em>control abstraction</em></td>
<td>addListener, visitor, ...</td>
</tr>
<tr>
<td>points-to</td>
<td><em>access abstraction</em></td>
<td>caretaker, membrane, ...</td>
</tr>
</tbody>
</table>
Membranes: Transitive Interposition

```javascript
function makeFnMembrane(target) {
  var enabled = true;
  function wrap(wrapped) {
    if (wrapped !== Object(wrapped)) {
      return wrapped;
    }
    return function(...args) {
      if (!enabled) { throw new Error("revoked"); }
      return wrap(wrapped(...args.map(wrap)));
    }
  }
  return def({
    wrapper: wrap(target),
    revoke: function() { enabled = false; }
  });
}
```
function makeROFile(file) {
  return def({
    read: file.read,
    getLength: file.getLength
  });
}

var rorFile = makeROFile(revocableFile);
No powerful references by default

Alice says:

```javascript
var bobSrc = //site B
var carolSrc = //site C
var bob = eval(bobSrc);
var carol = eval(carolSrc);
```
No powerful references by default

Bob and Carol are confined.
Only Alice controls how they can interact or get more connected.

Alice says:

```javascript
var bobSrc = //site B
var carolSrc = //site C
var bob = eval(bobSrc);
var carol = eval(carolSrc);
```
No powerful references by default
Only connectivity begets connectivity

Alice says:

```javascript
var counter = makeCounter();
bob(counter.incr);
carol(counter.decr);

bob = carol = null;
```
Only connectivity begets connectivity

Bob can only count up and see result. Carol only down. Alice can only do both.

Alice says:

```javascript
var counter = makeCounter();
bob(counter.incr);
carol(counter.decr);
bob = carol = null;
```
Membrane eval → compartment

```javascript
var compartment = makeMembrane(eval);
var vbob = compartment.wrapper(bobSrc);
```
Membrane eval → compartment

```javascript
var compartment = makeMembrane(eval);
var vbob = compartment.wrapper(bobSrc);
// ...
```
Membrane eval → compartment

```javascript
var compartment = makeMembrane(eval);
var vbob = compartment.wrapper(bobSrc);
//...
compartment.revoke();
```
Composing Authority

Subset

Usually intersection

U?
Rights Amplification

Authority conditional on other possessions.

Enables more expressive power.
function makeBrand() {
    var amp = WeakMap();
    return def(
        seal: function(payload) {
            var box = def({});
            amp.set(box, payload);
            return box;
        },
        unseal: function(box) {
            return amp.get(box);
        }
    );
}

Rights Amplification
function makeBrand() {
  var amp = WeakMap();
  return def({
    seal: function(payload) {
      var box = def({});
      amp.set(box, payload);
      return box;
    },
    unseal: function(box) {
      return amp.get(box);
    }
  });
}
Distributed Secure Currency
Distributed Secure Currency

var paymentP = myPurse ! makePurse();
var paymentP = myPurse ! makePurse();
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
```
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
```
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
```
Distributed Secure Currency

\[
\text{var paymentP} = \text{myPurse} ! \text{makePurse}(); \\
\text{paymentP} ! \text{deposit}(10, \text{myPurse});
\]
Distributed Secure Currency

```plaintext
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
```
Distributed Secure Currency

```plaintext
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
```
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);

return Q(paymentP).when(function(p) {
```

[Diagram showing transactions between Alice and Bob with different currency amounts.]
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);

var goodP = bobP ! buy(desc, paymentP);

return Q(paymentP).when(function(p) {
  return Q(myPurse ! deposit(10, p)).when(function(_) {
```

Diagram:

- Alice
  - $90
- Bob
  - $10
  - $200
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);

return Q(paymentP).when(function(p) {
  return Q(myPurse ! deposit(10, p)).when(function(_) {
```

Diagram:
- Alice
  - $90
  - $10
- Bob
  - $200
-deposit
- buy(desc, paymentP)
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);

return Q(paymentP).when(function(p) {
    return Q(myPurse ! deposit(10, p)).when(function(_) {
```
Distributed Secure Currency

```javascript
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
return Q(paymentP).when(function(p) {
  return Q(myPurse ! deposit(10, p)).when(function(_) {
    return good;
  }, ...
```
Money as “factorial” of secure coding

No explicit crypto

```
function makeMint() {
  var amp = WeakMap();
  return function mint(balance) {
    var purse = def({
      getBalance: function() { return balance; },
      makePurse: function() { return mint(0); },
      deposit: function(amount, src) {
        Nat(balance + amount);
        amp.get(src)(Nat(amount));
        balance += amount;
      } });
    function decr(amount) {
      balance = Nat(balance – amount);
    }
    amp.set(purse, decr);
    return purse;
  }
}
```

Alice → Bob: buy

makeMint

mint

amp

purse

decr

balance

No explicit crypto

Alice: mint

Bob: mint

purse

decr

balance
## Dimensions of Electronic Rights

<table>
<thead>
<tr>
<th>Object reference</th>
<th>Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>Exclusive</td>
</tr>
<tr>
<td>Specific</td>
<td>Fungible</td>
</tr>
<tr>
<td>Opaque</td>
<td>Assayable</td>
</tr>
<tr>
<td>Exercisable</td>
<td>Symbolic</td>
</tr>
</tbody>
</table>
Smart Contracts as Board Games

**Negotiation**
- Design a game both expect to win

**Players make moves, but only “legal” ones**
- Move changes state of board
- Board-state determines move “legality”

**ERights are “pieces” placed on board**
- Game escrows pieces,
- Pieces/ERights released only by play
A Simple Exchange Game
The Five Players
A Covered Call Option
Composing Networks of Games
Questions?

Composing Networks of Games
Smart Contracts as Games
Dimensions & Taxonomy of Electronic Rights
Patterns of Safe Cooperation
Access Abstractions and Compositions
Object-capabilities (ocaps)
Objects, References, Messages