

# LoPSiL: A Location-based Policy-specification Language

Jay Ligatti, Billy Rickey, and **Nalin Saigal**

Department of Computer Science and Engineering,  
University of South Florida

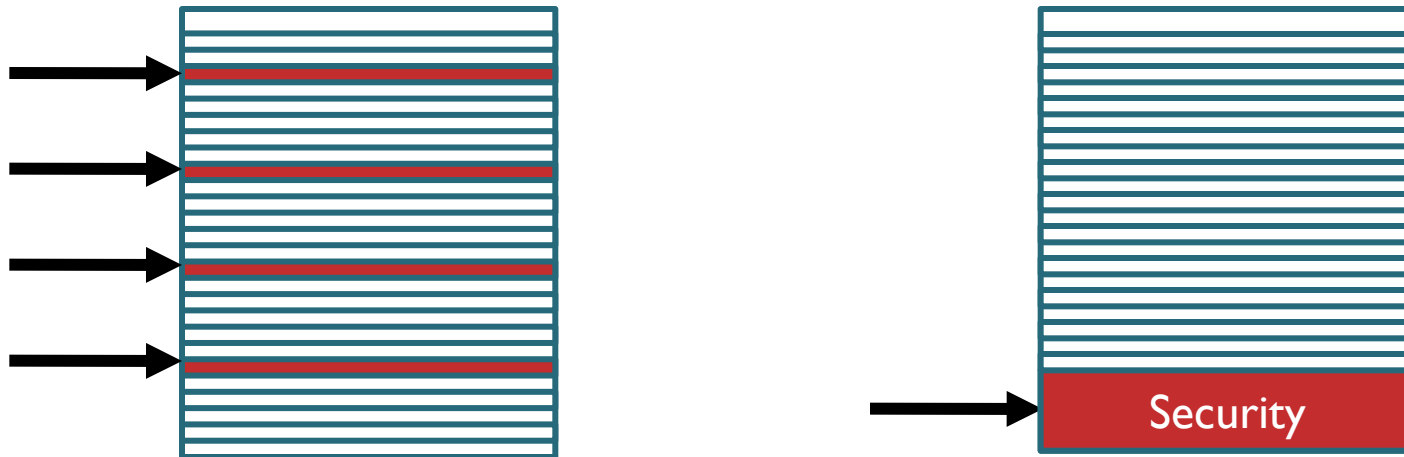
# Introduction

- **Policy-specification languages** are domain-specific programming languages designed to make it easier to specify and enforce sound security policies

```
policy allowOnlyHTTP(Socket s) {  
    if (s.getPort() == 80 ||  
        s.getPort() == 443)  
    then ALLOW  
    else DISALLOW  
}
```

# Motivations for Using Policy-specification Languages

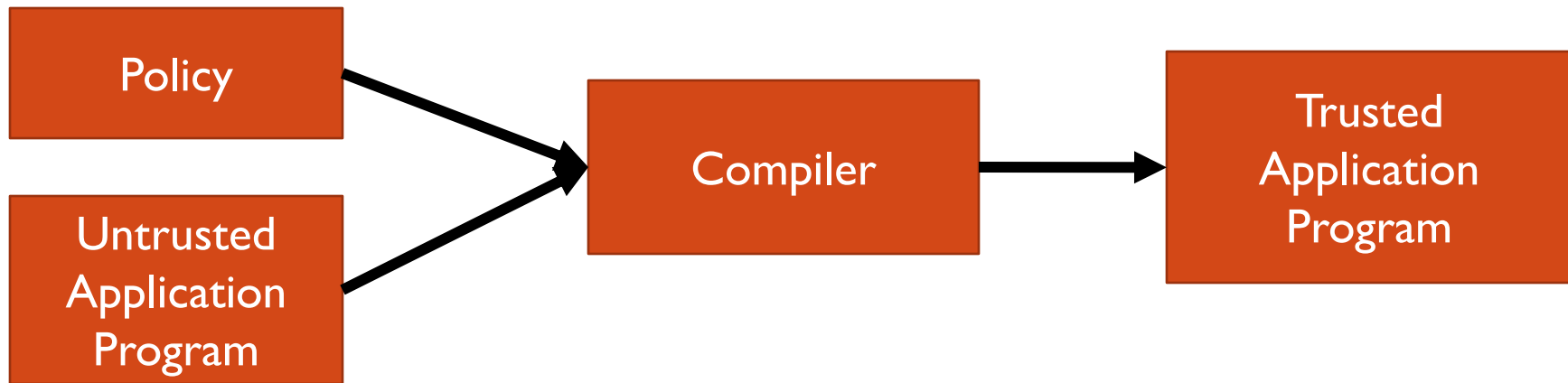
1. To specify and enforce custom constraints (i.e., a **policy**) on untrusted software
2. To enable programmers to organize their code with a centrally specified policy module



- No need to scatter security checks throughout code
- Provides standard policy-modularization benefits
  - Makes centralized policy easier to create, locate, analyze, and maintain

# Related Work

- Many expressive policy-specification languages and systems have been implemented
- Implemented as compilers that convert untrusted into trusted applications



- The trusted application program is equivalent to the untrusted program except that it contains inlined code that enforces the policy at runtime

# Related Work

- Several languages and systems employ this architecture
  - E.g., *Naccio*, *PoET/PSLang*, *Polymer*
  - But it can be inconvenient to specify mobile-device policies in these languages because they lack primitives for managing location information
- On the other hand, some policy-specification languages do provide primitives for managing location information
  - E.g., *Policy Mapper*, *OpenAmbient*
  - But these languages are Turing-incomplete and limited to access-control policies

# LoPSiL (Location-based Policy-specification Language)

- This research designs and implements a language called **LoPSiL**
- As far as we are aware, LoPSiL is the first expressive (Turing-complete) policy-specification language that targets location-based policies
  - LoPSiL's novelty is that it provides abstractions for conveniently accessing and manipulating location information in policies
- We have posted the implementation of LoPSiL and several example policies online at [www.cse.usf.edu/~ligatti/projects/runtime/](http://www.cse.usf.edu/~ligatti/projects/runtime/)

# Outline

- Introduction
  - Motivation
  - Related work
- LoPSiL
  - Core linguistic constructs
  - Example policies
- A LoPSiL Compiler
  - Compiler architecture
  - Experiential observations
- Conclusions

# Language Overview

- Due to the popularity of Java (particularly Java ME) as an application programming language for mobile devices, we'll talk about (and we've implemented) LoPSiL in the context of Java
- However, LoPSiL is based on six core abstractions (each implemented as a Java class) that we expect to be portable to other languages and platforms

# Core Linguistic Construct I

- **Location**
  - May refer to a room, chair, floor, building, campus, GPS coordinates, region of a network topology, etc.
    - Definition of locations is not “baked in”
    - Users can define their own (possibly abstract) locations by extending the **Location** class
- **Locations** have an identity (e.g., name or GPS coordinates)
- LoPSiL provides many built-in utility methods for manipulating GPS locations (e.g., calculate distance between two locations)
  - Users are free to define others

# Core Linguistic Construct 2

- **LocationDevice**

- Is LoPSiL's interface to real-time location information

- **LocationDevices** must implement two methods:

1. To return the device's current location
2. To return the device's **location granularity**: with what precision/accuracy the current location is known (e.g., within 0.5m, 2km, 1 room etc.)

- Policies can require devices to provide location information with particular granularity thresholds

# Core Linguistic Construct 3

- **PolicyAssumptions**
  - Encapsulates two assumptions made by policies about a **LocationDevice**:
    - **LocationGranularityAssumption**: Policy may require location information with a particular accuracy
    - **FrequencyOfUpdatesAssumption**: Policy may require that location updates arrive with a particular frequency
  - A LoPSiL policy gets notified automatically whenever a **LocationDevice** violates that policy's granularity or frequency-of-updates assumptions

# Core Linguistic Construct 4

- **Action**

- Encapsulates information (method signature, run-time arguments, calling object, return value, etc.) about a **security-relevant method**
- An **Action** object gets passed to a policy before and after every security-relevant method executes
- The policy can analyze an **Action** object passed to it to determine which security-relevant method is being invoked or has just been invoked
- Policies get to decide whether and how the **Actions** passed to it will execute

# Core Linguistic Construct 5

- **Reaction**

- Conveys a policy's decision about whether and how an action is allowed to execute
- Policies can react to a given **Action  $a$**  by returning one of four **Reactions**
  1. **OK**:  $a$  is safe to execute
  2. **Exception**:  $a$  is unsafe; exception should be raised
  3. **Replace**:  $a$  is unsafe; a precomputed return value should be returned instead of executing  $a$
  4. **Halt**:  $a$  is unsafe; program should be halted

# Core Linguistic Construct 6

- **Policy**

- Specifies constraints on untrusted software

- Five parts to a LoPSiL **Policy**

1. **PolicyAssumptions**

2. **void handleGranularityViolation()**

3. **void handleFrequencyViolation()**

4. **void onLocationUpdate()**

- Invoked when any **LocationDevice** associated with the policy updates its **Location** information

5. **Reaction react(Action a)**

- Return a reaction to any security-relevant action

# Example Policy: AllowAll

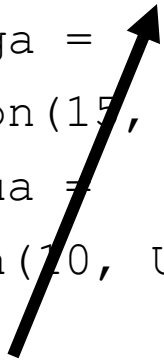
```
public class AllowAll extends Policy {  
    public LocationDevice[] devices = {new  
        LopsilGPS(LopsilGPS.GARMIN)};  
    public LocationGranularityAssumption lga =  
        new LocationGranularityAssumption(15, Units.METERS);  
    public FrequencyOfUpdatesAssumption fouda =  
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);  
    public PolicyAssumptions pa =  
        new PolicyAssumptions(this, devices, lga, fouda);  
    public void handleGranularityViolation() {System.exit(1);}  
    public void handleFrequencyViolation() {System.exit(1);}  
    public synchronized void onLocationUpdate() {  
        System.out.println("new location = " +  
            devices[0].getLocation()); }  
    public synchronized Reaction react(Action a) {  
        return new Reaction("ok"); } }  
}
```

# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
                                        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
                            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```


# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
                                        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
                            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```



# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fouda =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fouda);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```



# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```

# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```

# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
                                        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
                               devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```

# Example Policy: AllowAll

```
public class AllowAll extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public LocationGranularityAssumption lga =
        new LocationGranularityAssumption(15, Units.METERS);
    public FrequencyOfUpdatesAssumption fous =
        new FrequencyOfUpdatesAssumption(10, Units.SECONDS);
    public PolicyAssumptions pa =
        new PolicyAssumptions(this, devices, lga, fous);
    public void handleGranularityViolation() {System.exit(1);}
    public void handleFrequencyViolation() {System.exit(1);}
    public synchronized void onLocationUpdate() {
        System.out.println("new location = " +
            devices[0].getLocation()); }
    public synchronized Reaction react(Action a) {
        return new Reaction("ok"); } }
}
```

# Relationship with Existing Work

- Existing policy-specification languages such as Naccio, PSLang, and Polymer provide constructs similar to **Actions**, **Reactions**, and **Policy** with **react**-style methods
- LoPSiL's novelty is its addition of optional location-related policy components: **Locations**, **LocationDevices**, **PolicyAssumptions**, and **onLocationUpdate()**, **handleGranularityViolation()**, and **handleFrequencyViolation()** methods

# Outline

- Introduction
  - Motivation
  - Related work
- LoPSiL
  - Core linguistic constructs
  - **Example policies**
- A LoPSiL Compiler
  - Compiler architecture
  - Experiential observations
- Conclusions

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if(ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if(ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if (ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if (ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if(ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Access-control Policy

- Prevents an application from reading GPS data outside of work hours

```
public class NoGpsOutsideWorkTime extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa = ...
    public synchronized Reaction react(Action a) {
        if(ActionPatterns.matchesLocationRead(a) &&
            !TimeUtils.isWorkTime())
            //return a null location to the application
            return new Reaction("replace", null);
        else
            return new Reaction("ok");
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if (devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS) > 10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if(devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS)>10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if (devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS) > 10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if(devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS)>10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if (devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS) > 10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Deviation-from-path Policy

- Requires that navigational aid appear when the device's current location deviates from its expected path

```
public class ShowNavigation extends Policy {
    public LocationDevice[] devices = {new
        LopsilGPS(LopsilGPS.GARMIN)};
    public PolicyAssumptions pa =
        ...
    public synchronized void onLocationUpdate() {
        if (devices[0].getLocation().
            distance(getExpectedCurrentLocation(),
                Units.METERS) > 10)
            AppGUI.displayNavigationalAid();
    }
}
```

# Other Policies

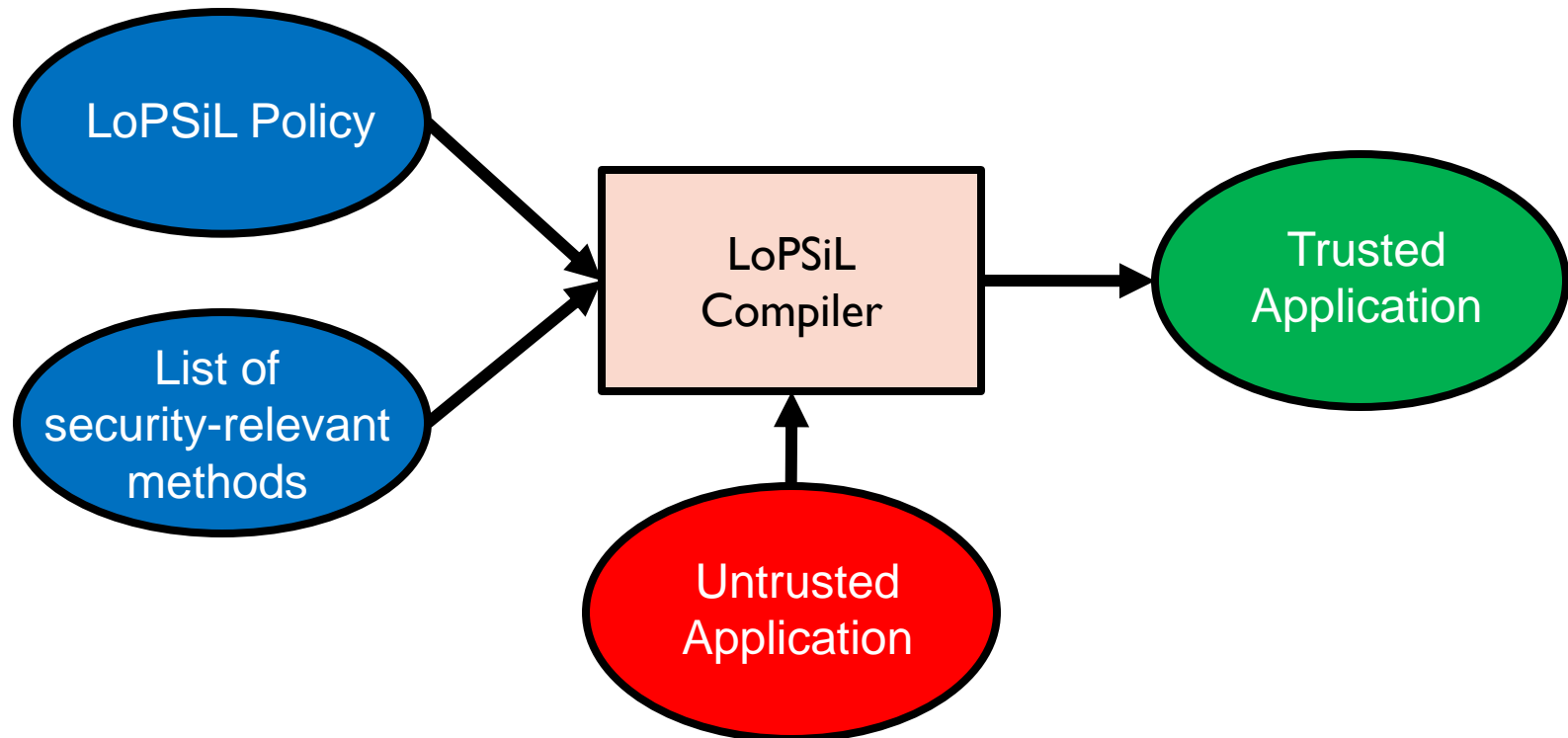
- **Safe-region Policy:**
  - Requires a robot to encrypt all outgoing communications when the robot's location is outside a secure-region perimeter
- **Social-networking Policy:**
  - If the policy infers that the device has completed a journey of at least 100km over the past 2 hours, then all friends in the user's address book who are located within 20km of the new location get invited to rendezvous

# Outline

- Introduction
  - Motivation
  - Related work
- LoPSiL
  - Core linguistic constructs
  - Example policies
- A LoPSiL Compiler
  - **Compiler architecture**
  - Experiential observations
- Conclusions

# Compiler Architecture

- A LoPSiL compiler needs to input a policy, a list of security-relevant methods, and an untrusted application
- A LoPSiL compiler needs to output a trusted application formed by inserting (inlining) policy code before and after every security-relevant method in the untrusted application



# Compiler Architecture

- We use AspectJ as a convenient tool for inlining policy code into the untrusted application
- LoPSiL inherits AspectJ's limitation that it can only inline code into application files (and not standard Java libraries)
  - So, LoPSiL monitors can't make decisions about the execution of security-relevant methods invoked by (non-application) library classes

# Compiler Architecture

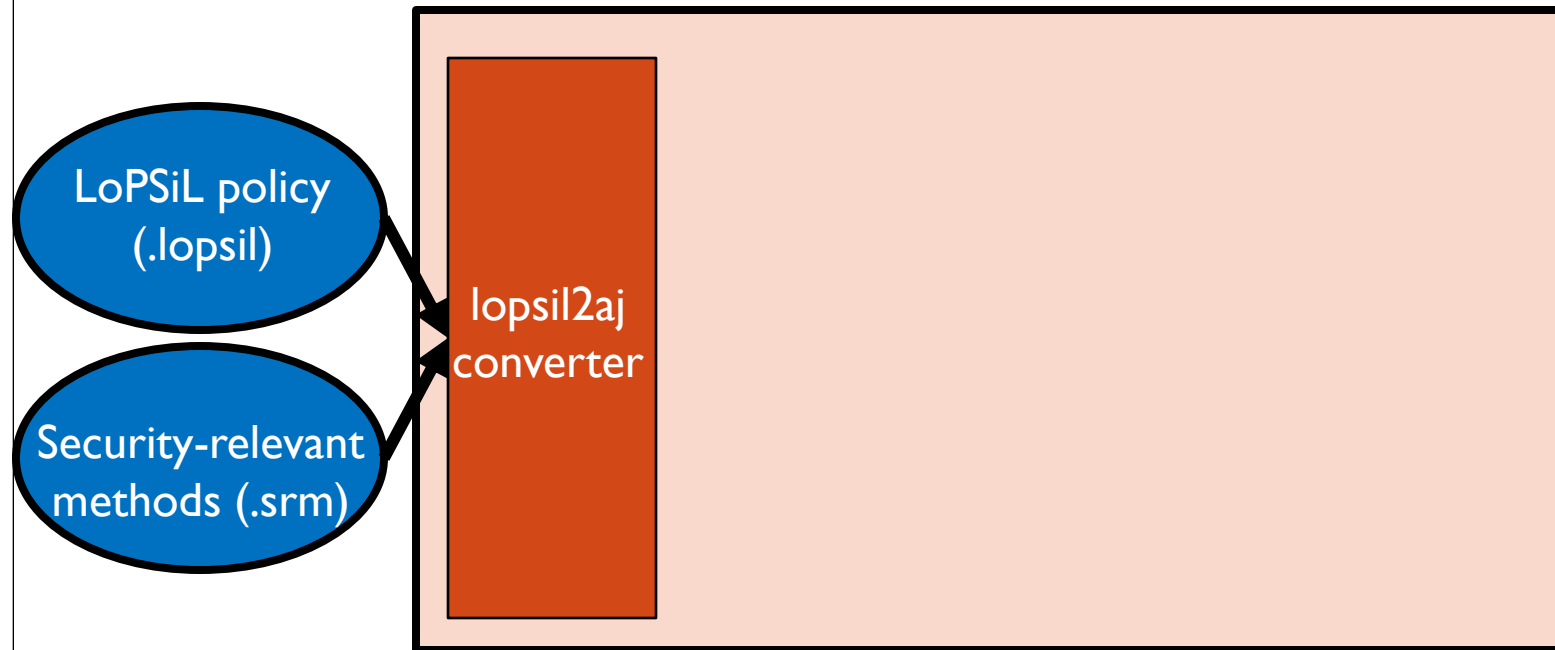
LoPSiL policy  
(.lopsil)

Security-relevant  
methods (.srm)

- User specifies desired policy in a `.lopsil` file
- User creates a listing of security-relevant methods in `.srm` file

# Compiler Architecture

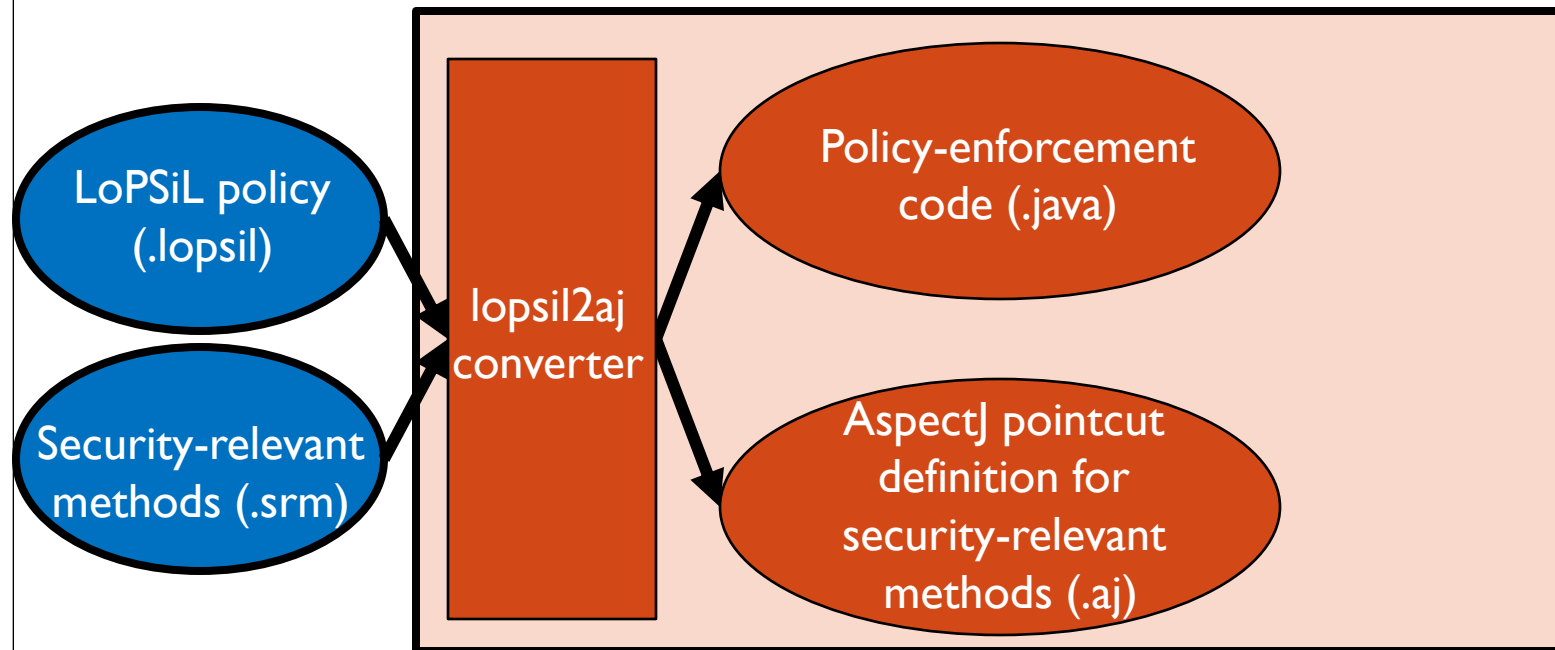
## LoPSiL Compiler



- User inputs the `.lopsil` and the `.srm` file into a `lopsil2aj` converter, because it converts LoPSiL files into files that can be input into an AspectJ compiler

# Compiler Architecture

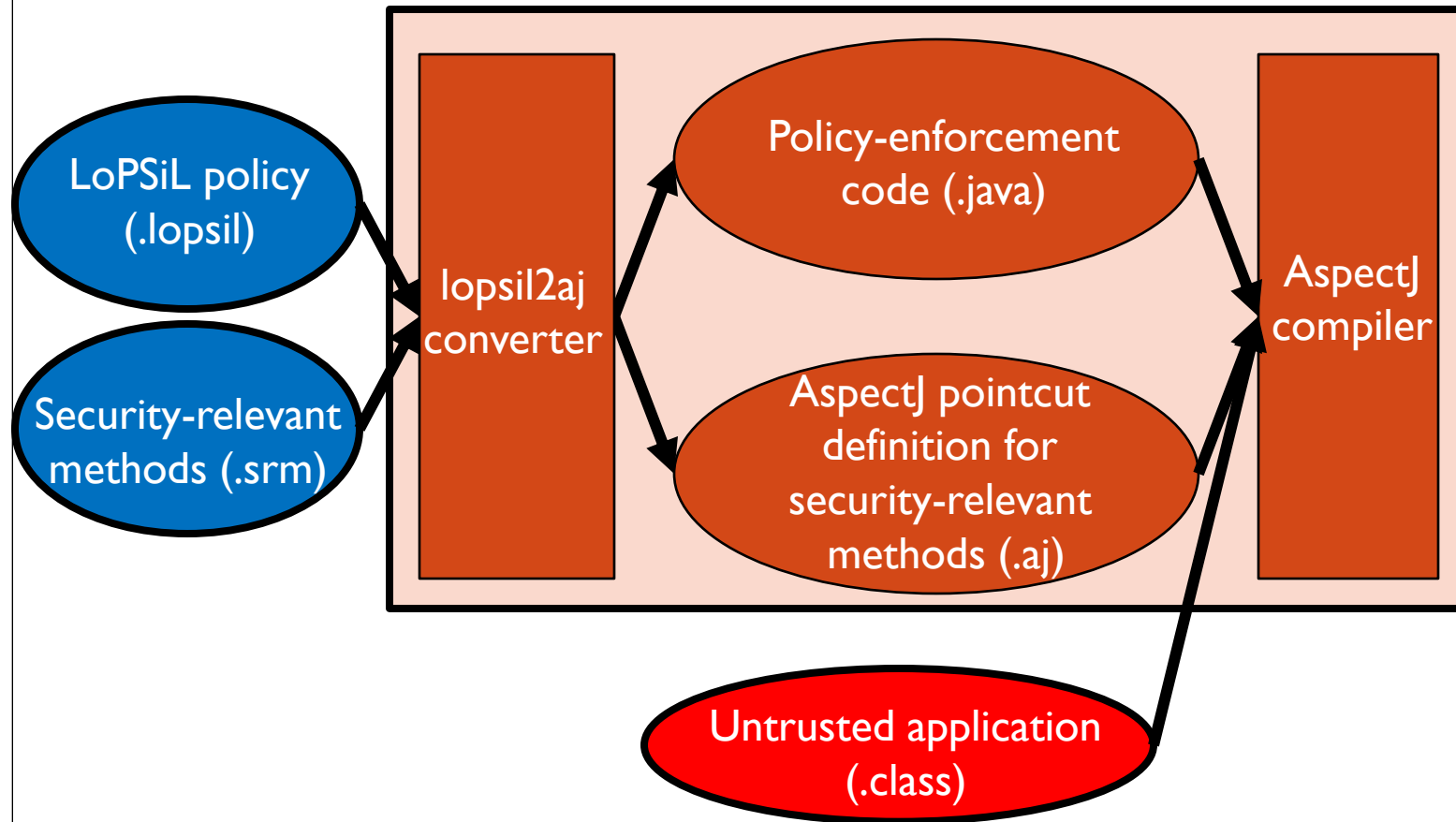
## LoPSiL Compiler



- The compiler converts the `.lopsil` file to Java version (`.java`) of the policy by inserting three lines of code to import LoPSiL-library classes
- The compiler converts the `.srm` file to an AspectJ-code file (`.aj`) that has definitions indicating
  1. which policy code is to be executed and
  2. when should that policy code be executed

# Compiler Architecture

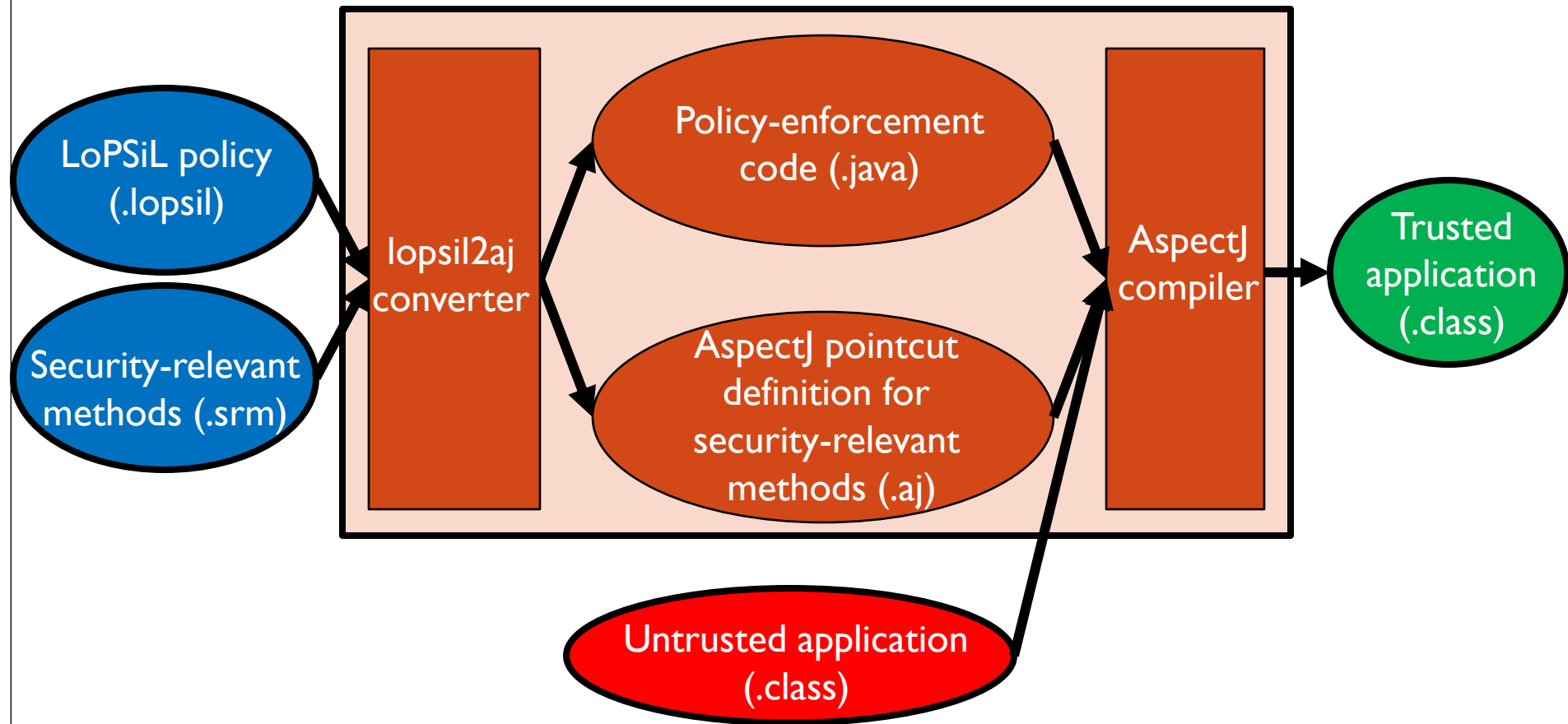
## LoPSiL Compiler



- The compiler inputs the `.java` file, the `.aj` file, and the untrusted application (`.class`) into a standard AspectJ compiler

# Compiler Architecture

## LoPSiL Compiler



- AspectJ compiler inlines policy code before and after all security-relevant methods and produces an application that is secure w.r.t. the original LoPSiL policy

# Outline

- Introduction
  - Motivation
  - Related work
- LoPSiL
  - Core linguistic constructs
  - Example policies
- A LoPSiL Compiler
  - Compiler architecture
  - **Experiential observations**
- Conclusions

# Experiential Observations

- Our location-dependent policies consistently based policy decisions on:
  - Absolute location of the device
  - Geographic relationship of device's location with another location, or with a region of locations
  - Velocity or acceleration of the device
- Therefore, LoPSiL provides several utility methods for calculating distances, boundaries, velocities, and acceleration
- Users can access all these methods and can implement other custom operators when built-in methods are insufficient

# Experiential Observations

- We found that LoPSiL was sufficiently expressive to specify all the location-dependent policies we considered enforcing
- None of the example policies mentioned earlier took much time (more than a few hours) to design, specify, and test
- Therefore, we believe that the six core constructs underlying LoPSiL serve as good abstractions for specifying location-dependent policies

# Outline

- Introduction
  - Motivation
  - Related work
- LoPSiL
  - Core linguistic constructs
  - Example policies
- A LoPSiL Compiler
  - Compiler architecture
  - Experiential observations
- **Conclusions**

# Conclusions

- LoPSiL is a language for specifying location-dependent runtime security policies
- LoPSiL's novelty lies in its expressive abstractions for accessing and manipulating location information in policies
- Given the increasing ubiquity of mobile devices, we believe the area of location-based policy-specification languages will be an important topic of research
  - Will need involvement/collaboration of researchers in the security, programming languages, and mobile-device-applications communities

Thanks/Questions?

# References

- [1] M. Anisetti, C. Ardagna, V. Bellandi, and E. Damiani. Openambient: a pervasive access control architecture. In A. Schmidt, M. Kreutzer, and R. Accorsi, editors, *Long-Term and Dynamical Aspects of Information Security: Emerging Trends in Information and Communication Security*. Nova Science Publisher, Inc., 2007.
- [2] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Supporting location-based conditions in access control policies. In *Symposium on Information, computer and communications security*, 2006.
- [3] R. Bhatti, M. Damiani, D. Bettis, and E. Bertino. Policy mapper: Administering location-based access-control policies. *IEEE Internet Computing*, 12(2):38–45, 2008.
- [4] U. Erlingsson and F. B. Schneider. IRM enforcement of Java stack inspection. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.
- [5] D. Evans and A. Twyman. Flexible policy-directed code safety. In *IEEE Security and Privacy*, Oakland, CA, May 1999.
- [6] J. Ligatti, L. Bauer, and D. Walker. Enforcing non-safety security policies with program monitors. In *10th European Symposium on Research in Computer Security (ESORICS)*, Sept. 2005.