Inferring Required Permissions for Statically Composed Programs

Tero Hasu Anya Helene Bagge Magne Haveraaen
{tero,anya,magne}@ii.uib.no

Bergen Language Design Laboratory
University of Bergen


travel
productivity
experience
NOKIA
BlackBerry
23:59
Contacts
Browser
Calendar
Text Messages
BlackBerry World
File Manager
Remember
Pictures
Settings
Search
Calculator
Clock
NOKIA
Duck Duck Go

smartphones—a security risk for users

- privacy and usage cost concerns
- natively third-party programmable
  - "app stores" have programs in large numbers
    - including malware and "grayware"

permission-based security models

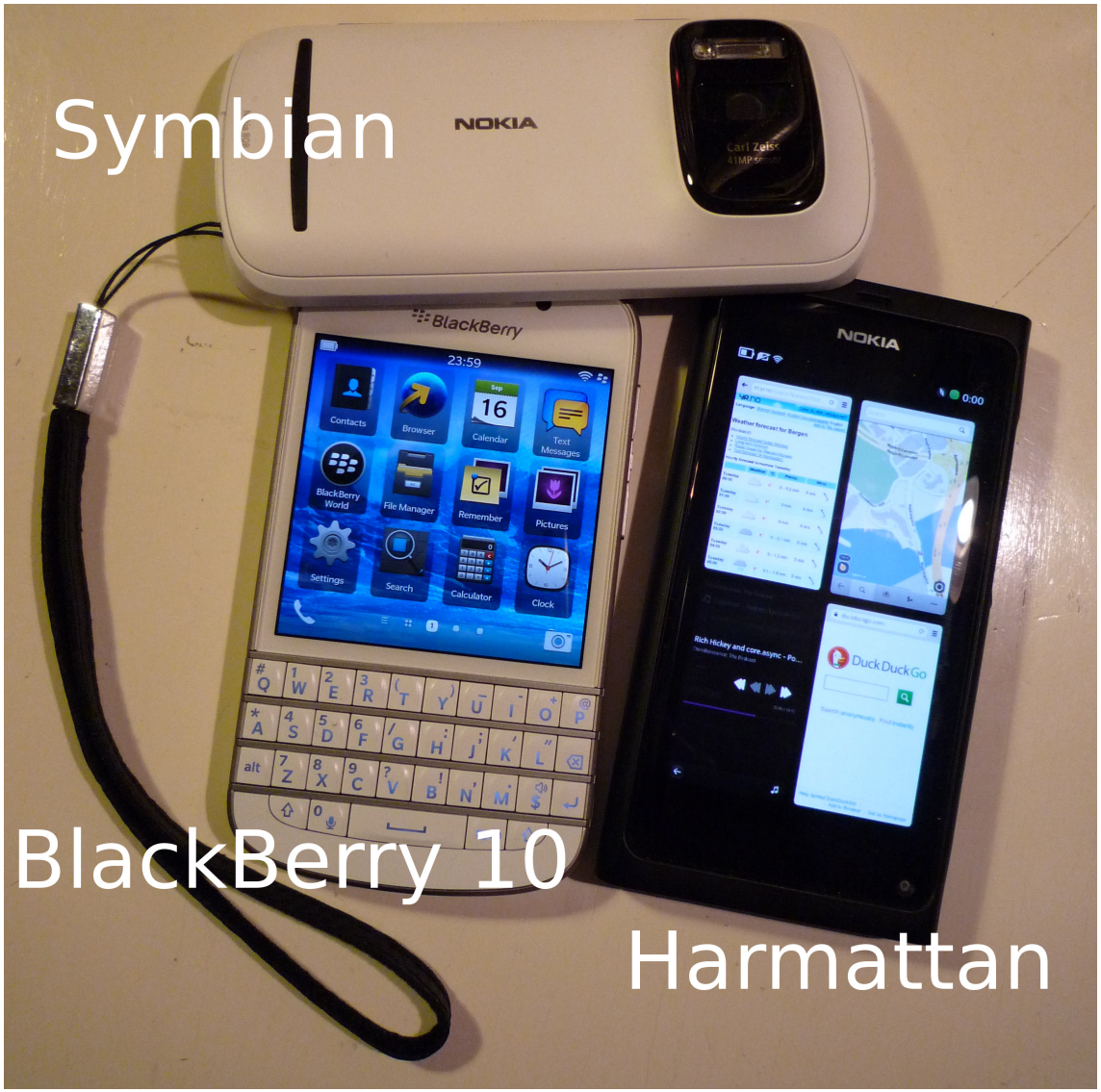
- similar to VAX/VMS "privileges" introduced in late 70's
- popularized by smartphone OSes
- primarily: access control for sensitive APIs
- user approval of permissions $\rightarrow$ security and usability implications?


permissions—a concern for app developers

declaring permissions

too small a set $\rightsquigarrow$ runtime errors
too large a set $\rightsquigarrow$ worried users
optimal set $\rightsquigarrow$ maintenance hassle

hassle compounds in a cross-platform setting



- permission requirements vary between platform releases
  - often inadequately documented
- an app may come in multiple variants
  - sometimes because of permission restrictions
    - can differ per distribution channel

prevalent smartphone vendor supported approach

- infer required permissions from a program's platform API use

inverse inference: API use from permissions?

- ContextLogger2—a maximally intrusive app (unusual case)
  - configuration script
    - target & certificate $\xrightarrow{compute}$ available permissions
    - target & SDK & permissions $\xrightarrow{compute}$ available/accessible APIs
  - ☹ lots of conditional compilation at API use sites

```
;; Music Player Remote Control API accessibility (Symbian)
(define/public (have-mplayerremotecontrol.attr)
  (and
   (and (>= (s60-vernum.attr) 31)
        (<= (s60-vernum.attr) 32))
   (= (kit-vernum.attr) 31)
   (sublist?
    '(ReadDeviceData ReadUserData
      WriteUserData WriteDeviceData)
    (capabilities))))
```

prevalent smartphone vendor supported approach

- infer required permissions from a program's platform API use
- some tools are available
  - examine either binaries or source code
  - current tools for scanning *native* programs rely on heuristics
    - dynamic loading and invocation (when allowed) make accurate analysis difficult/impossible

permission analysis tools availability

Android	Stowaway, Permission Check Tool (both 3rd party)
bada	API and Privilege Checker
BB10	none
Harmattan	`aegis-manifest` (automatically generates a declaration)
Symbian	Capability Scanner
Tizen	API and Privilege Checker
WP7	Store Test Kit (managed code only in WP7 apps)
WP8	none


cross-platform permission inference

- infer required permissions from a program's platform-agnostic API use
  - implementations encapsulate platform API use
- and: declare permissions for each implementation of said APIs
- and: program against said APIs in a language you can analyze to determine API use

Can reuse the same API:

- for multiple platforms (if can implement)
- in multiple apps (if suitably general)

domain engineering

favorable language characteristics

interface-based abstraction

- to support organizing cross-platform codebases

static analysis friendliness

- to allow for accurate inference

adopting the approach

- adopt a favorable language, preferably
  - (coding conventions may help)

in-source permission annotations

- as an extra-language feature (probably within comments)
- using any language-provided annotation support
- by extending the language

our proof of concept: based on Magnolia

- general-purpose research programming language Magnolia
  - `http://magnolia-lang.org/` 
- its implementation provides the required language infrastructure
- permission management is just one application for Magnolia
  - perhaps: address error handling in general (not just permission errors)
    - separate idea of *partiality* from concrete details of error reporting—Bagge: Separating exceptional concerns (2012)
    - abstract over different mechanisms—Hasu: Concrete error handling mechanisms should be configurable (2012)


Magnolia's interface-based abstraction

- a Magnolia interface is declared as a `concept`
  - each `concept` may have multiple `implementation`s
  - one `implementation` may satisfy multiple `concept`s

Magnolia's static analysis friendliness

- Magnolia avoids "dynamism"
  - no pointers, carefully controlled aliasing
  - no runtime passing of code (e.g., no higher-order functions)
  - abstract data types, not objects
    - concrete type and operations known at compile time
  - makes up for restrictions with extensive support for static "wiring" of components
- Magnolia promotes use of semantically rich concepts
  - a `concept` may specify (some) semantics as `axiom`s
  - an operation may specify use limitations as `guard`s

declaring in Magnolia—what & how

- platform-specific required permission information (per operation, per **implementation**)
  - as a predicate expression—commonly need &&, sometimes ||
  - to be collated into an inference result for a **program**
  - e.g.,
    ```
    alert RequiresPermission unless pre SNS_SERVICE()
    ```
- platform-agnostic, abstract permission error names (once each)
  - to allow for error-handling in portable code
  - e.g., `alert NoPermissionSocial <: NoPermissionCloud;`
- mappings between platform-specific, concrete errors and error names (per operation, per **implementation**)
  - for the compiler to implement the mapping
  - e.g., `alert NoPermissionSocial if post value == E_PRIVILEGE_DENIED`


domain engineering an exporter: data extraction and outputting

```
concept DataSrc = {
  use World;
  use DataCollection;

  procedure readAll(upd sys : System, out coll : Coll);
};

concept DataTgt = {
  use World;
  use DataCollection;

  procedure writeAll(upd sys : System, obs coll : Coll);
};
```

runtime permission errors

```
implementation Permissions = {
  alert NoPermission;
};
```

platform-specific permissions

```
implementation HarmattanPermissions = {
  use Permissions;
  predicate TrackerReadAccess() = Permission; // Harmattan
  predicate TrackerWriteAccess() = Permission; // Harmattan
  predicate GrpMetadataUsers() = Permission; // Harmattan
  // ...
};

implementation SymbianPermissions = {
  use Permissions;
  predicate ReadUserData() = Permission; // Symbian
  // ...
};
```

Pardon the verbose syntax!

Symbian-native contacts reader implementation

```
implementation SymbianNativeContactsSrc =
  external C++ datasrc.SymbianContacts {
    require type System;
    require type Coll;
    require SymbianPermissions;
    procedure readAll(upd sys : System, out coll : Coll)
      alert RequiresPermission unless pre ReadUserData()
      alert NoPermission if leaving KErrPermissionDenied
      /* more alerts ... */;
  };

satisfaction SymbianNativeContactsIsDataSrc = {
  use DataCollection; use World; use SymbianPermissions;
} with SymbianNativeContactsSrc
  models DataSrc;
```

same for Harmattan

```
implementation HarmattanQtContactsSrc =
  external C++ datasrc.HarmattanContacts {
    require type System;
    require type Coll;
    require HarmattanPermissions;
    procedure readAll(upd sys : System, out coll : Coll)
      alert RequiresPermission unless pre
        TrackerReadAccess() && TrackerWriteAccess() &&
        GrpMetadataUsers()
      alert NoPermission unless pre haveQtContactsPerms()
      /* more alerts ... */;
  };

satisfaction HarmattanQtContactsIsDataSrc = {
  use DataCollection; use World; use HarmattanPermissions;
} with HarmattanQtContactsSrc
  models DataSrc;
```

portable code, against platform-agnostic interfaces

```
implementation DefaultEngine = {
  require DataSrc;
  require DataTgt;

  procedure exportData() {
    var sys : System = initialState();
    var dat : Coll;
    on NoPermission in readAll
      dat = emptyColl();
    call readAll(sys, dat);
    call writeAll(sys, dat);
  }
};
```

one program configuration

```
program SymbianContactsSaver = {
  use DefaultEngine;
  use DefaultWorld;
  use DefaultDataCollection;
  use SymbianNativeContactsSrc;
  use CxxFileOut;
};
```


permission inference

- Magnolia compiler assembles a `program`—only relevant `implementations` are included from codebase

currently in Magnolia

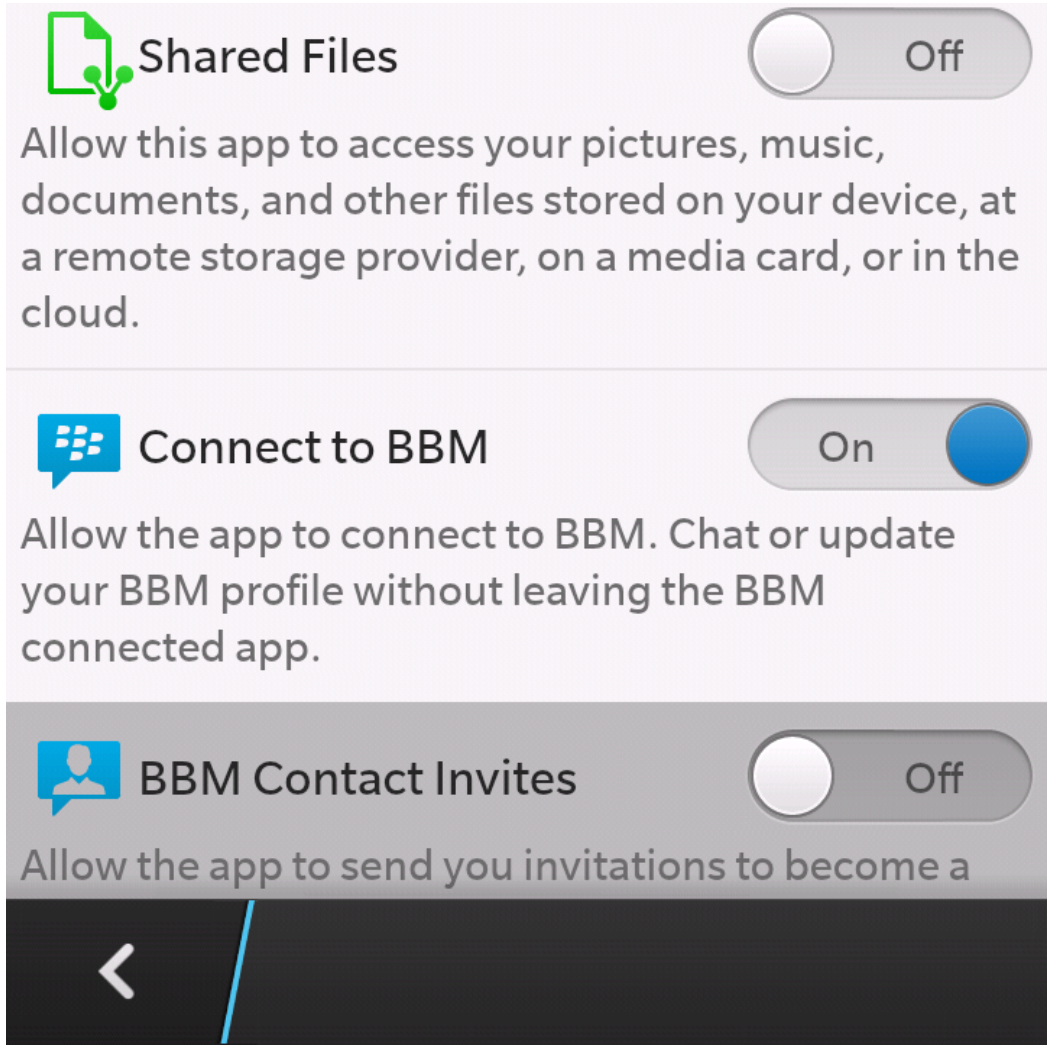
- accounts for all operations that appear in the `program`
  - any dead-code elimination happens after inference
- build a *set* of permissions, always picking left choice from $P_1||P_2$ expressions
  - e.g., $(P_1||P_2)\&\&(P_2||P_1) \rightarrow \{P_1, P_2\}$

more ideally

- would do some data-flow analysis to disregard obviously unreachable invocations
- would build a permission *expression*, and turn it into a set only afterwards, more optimally, according to a policy
  - e.g., favor less sensitive permissions


gain: permission management solution

- tools support for avoiding runtime errors due to permission underdeclaration
  - assuming correct and complete annotations, and grantable & granted permissions (toggleable in BB10 and iOS)
- language support for handling runtime permission errors portably



cost: annotation effort

- may be able to amortize annotation cost over many projects and configurations
  - unlike when manually declared in a per-project-configuration manifest file
- a way to store and perhaps share domain knowledge
  - "I know this implementation of this API requires these permissions"


Anyxporter—permission management test app

`https://github.com/bldl/anyxporter`

- cross-platform codebase, organized as concepts
- one "Magnoliafied" build configuration, with permission inference
  - Magnolia's integration with configuration and build tools still needs work

conclusion

- permissions are a concern to smartphone app devs
- we proposed a solution for permission management
  - requires no pre-existing permission tooling
  - can be applied to cross-platform codebases
  - no separately declaring permissions for each program
- we tried out the solution
  - by integrating permission support into Magnolia
  - by inferring the permissions of a cross-platform app

Anyxporter—contact data export

```
<?xml version="1.0" encoding="UTF-8"?>
<Contacts> ...
  <Contact> ...
    <ContactDetail>
      <DefinitionName>DisplayLabel</DefinitionName>
      <Label>Tero Hasu</Label>
    </ContactDetail>
    <ContactDetail>
      <DefinitionName>EmailAddress</DefinitionName>
      <EmailAddress>tero.hasu@ii.uib.no</EmailAddress>
    </ContactDetail>
    <ContactDetail>
      <DefinitionName>Guid</DefinitionName>
      <Guid>000000003e7be123-00e18ae873575ee5-41</Guid>
    </ContactDetail> ...
  </Contact> ...
</Contacts>
```