ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

Preventive information flow control through a mechanism of split addresses

SUPELEC, CIDRE LAB, SECCLOUD PROJECT

Deepak Subramanian, <deepak.subramanian@supelec.fr>
Guillaume Hiet <guillaume.hiet@supelec.fr>
Christophe Bidan <christophe.bidan@supelec.fr>

May 13th 2014

JavaScript

- Highly dynamic language.
- Widely used
- HTML5 increases possibilities for leak by providing more functionality
- No proper code encapsulation
- Everything can be changed [Including native functions]
- Functions can be created at runtime [and information flows can be modified]

IFC related work

- FlowFox – [1]
  - Based on the secure multi-execution paradigm
  - Divide labels and then execute them as individual processes.
  - Provides highest formal guarantees.
- Faceted Approach - [2]
  - Based on facets which divides a variable based on its principal.
  - Closest to our model. Requires all paths of the program to be executed.

OWASP Top 10 – 2013

- A1 – Injection
- A2 – Broken Authentication and Session Management
- A3 – Cross-Site Scripting (XSS) [1,2]
- A4 – Insecure Direct Object References [1,2]*
- A5 – Security Misconfiguration
- A6 – Sensitive Data Exposure [1,2]*
- A7 – Missing Function Level Access Control
- A8 – Cross-Site Request Forgery (CSRF)
- A9 – Using Known Vulnerable Components
- A10 – Unvalidated Redirects and Forwards

Probabilistic vs Possibilistic

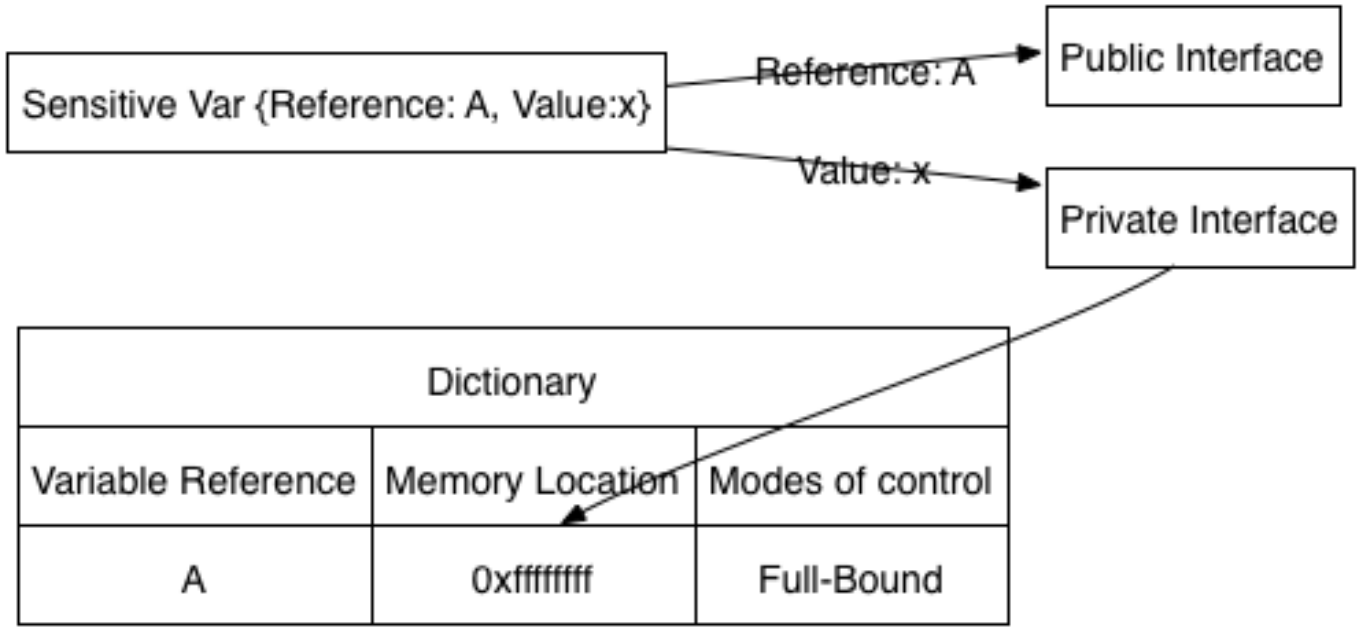
- Possibilistic
  - All investigated models have been possibilistic IFC in the domain of JS. We are yet to find a model using probabilistic IFC in the context of JS.
  - The objective of this approach is to eliminate any possibility of leak if the leak is considered feasible by the model.
  - Subsequently they tend to have a more larger over-approximation (Reduced precision).
- Probabilistic
  - Tend to be slower but more effective in making fine-grained IFC.
  - The objective of this approach is to eliminate the possibility of a leak if the leak has crossed a certain tolerated threshold set in the model
  - Greater precision if effectively utilized but have some performance considerations.

Address Split Model

- Mechanism to monitor and limit information flows
- This is a preventive enforcement mechanism
- PROBABILISTIC IFC
- The model is composed of the following components
  - Policies
  - Dictionaries
  - Functions
  - Access Control Mechanism
  - Information Flow Control Mechanism

The Split

Variable	Public Interface	Private Interface
Var	Dummy Value	Secret Value



Policies

- There are 2 main types of policies
  - Static policies
    - Policies that are straightforward and valid for that context (using variable names etc)
  - Inferred policies
    - Policies are defined in a way that they can be used in multiple contexts. (Using events etc)
- Policies can be user defined or developer defined. User defined policies have precedence over developer policies.

Policies – some problems

- Need to be isolated
  - Since JS can access complete DOM
- Associating policies to code is a problem
  - Function names are unreliable
  - Currently considering:
    - Function matching [doing a pattern matching between functions]
    - Function hashing [using the hash of the function as a string]
- Headers can be read / modified by extensions

3+1 Permission types

- Purpose:
  - Implemented ON Function TO access variables/fields
- Current models
  - Read / Write / Execute based models [for OS]
    - More access control
  - Tainting models
    - Information flow without fine-tuned access control [Haldar et al. 2005] [Chong et al. 2007]
    - Usually possibilistic in nature [Hauser et al. 2012]
  - Probabilistic models
- 3+1types of permissions
  - 3 permissions for access control + possibilistic information flow control
  - 1 permission for probabilistic information flow control

3 access control permissions

- Full-bound access
  - Read + Write into variables
- Semi-bound access
  - 2 cases
    - Semi-bound read access
      - Functions can only read from variables
      - Example: Any form field with user input can be given this access right
    - Semi-bound write access
      - Functions can only write into variables
      - Example: Many 3rd party widgets (such as twitter feeds) can be given this access right
- Instance access
  - Function gets to use a COPY of the variable which was made available to it at a particular instance.
  - Instance is a point in time where a set of pre-conditions are met.
  - Translator example

1 information flow permission

- Transitive Access
  - Probabilistic information flow
  - Based on entropy based information disclosure
  - Allows for a flexible declassification model
  - The transitive permissions are represented in the variable space (attached to every variable)
  - This is still in the initial stages

Information flow

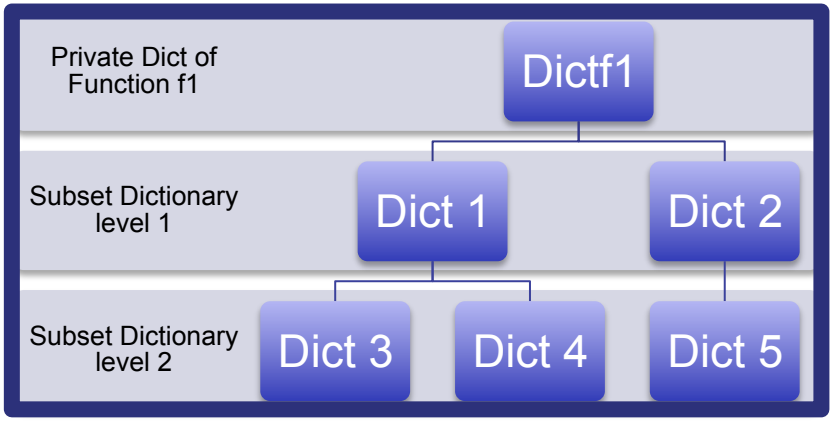
- We intend to allow different weights for individual bits in a string.
- STRING = S(1),T(3),R(1),I(10),N(1),G(1)
- If var a = STRING in a function and a has transitive 0.3 with weights as defined above
- Function f1(a) returns charAt(1) is allowed
- Function f2(a) returns charAt(4) is denied (returns charAt(4) of public interface)
- Example : Driving license numbers

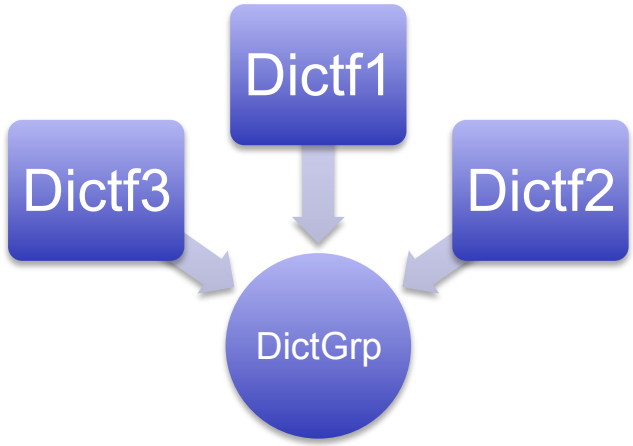
Dictionaries

- What are dictionaries?
  - Dictionaries a.k.a. "access tables" are data structures
  - provide a function side reference for the "3+1 permission types"

Dictionaries

- Private dictionaries
  - Unique to every function
- Subset dictionaries
  - Allows chaining of dictionaries
- Group dictionaries
  - Allows sharing of dictionaries between functions
- Instance dictionaries
  - Allows sharing of the copy of the variable





Dictionaries

PRIVATE DICTIONARY FUNCTION F1

Variable reference	Type of access	Variable Address
a	Full-Bound	0x1234
b	Instance	TD(b,submit.onClick)
[SUBSET DICTIONARY: HighDictionary]		

INSTANCE DICTIONARY

Variable reference	Event	Value
b	submit.onClick	13

GROUP DICTIONARY HighDictionary

Variable reference	Type of access	Variable Address
d	Full-Bound	0x4894
e	Semi-Bound Read-only	0x9812
[SUBSET DICTIONARY: LowDictionary]		

Functions

- Each function ⇔ own privileges to access various private variables
- Specifying policy for all functions is a tedious process
  - Harder with dynamic language like JS
- Reduced and effective function policies are required
- Hence,

  We classify them into 4 main types of functions

Functions - Classification

- Self-sufficient functions
- Utility functions
- Inheritance functions
- Guest functions

Self-sufficient functions

- Policies are well-defined
- If even one policy is specified for a function, it becomes a self-sufficient function
- Self-sufficient functions include functions with either
  - User defined policies
  - Inferred policies

Utility functions

- This is the default state of a function
- The definition is taken from the "modularity of a function" perspective
- The callee function is considered as a module of the caller
  - It shares the dictionary of the caller function for that instance and hence has the functionality intact
  - Eg. $.POST() would be a utility function

Inheritance function

- These functions are created by other functions
- Self-sufficient function creates functions whose privileges will not exceed its parent's
  - A utility function can only create utility functions

    [or in the rare event, self-sufficient functions]
- The inheritance function will continue to be bound to its parent when things change.
  - if (parent drops privileges) => (inheritance function drops privileges)
  - If (parent is deleted) => (inheritance function drops all privileges) ≠ utility function

```
function
addFunction()
{
 var s =
document.createElem
ent("script");
  s.type = "text/
javascript";
  s.text =
'function ab1()
{ alert(1); }';
  $
('head').append(s);
}
```

Guest functions

- Can use properties of a self-sufficient/inheritance function temporarily
- Pre-conditions must be met
  - These include the occurrence of a particular event in the system etc.,
- The set of values that can be used can be restrictive
  - For example, the policy may specify the variables of the function that can be accessed, the maximum number of reference lookups, maximum number of instances of the function that are allowed a lookup etc.,
- DEMO JSONP & asynchronous communication

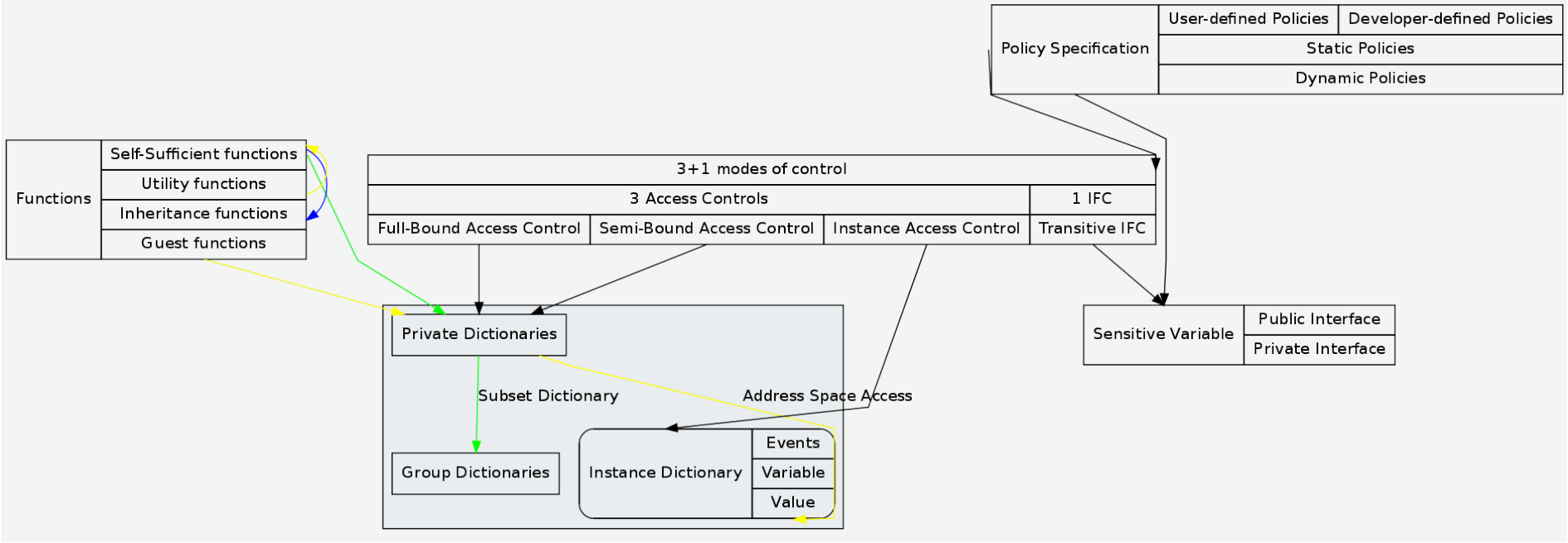
Example policies

- F3 ⇔Full-Bound >>> FV3

- IF (passwordBox IN form) THEN function@[form.onSubmit] ⇔(INSTANCE(form.submitButton.onClick)) >>> passwordBox.text ;

passwordBox.text ⇔ TRANSITIVE(SIZE(8N,15N), ENTROPY(0.3), PATTERN(([0-9]|[a-z]|[_])*)

ASD CONCEPTS



Classification & Declassification

- Classification
  - Policies + Rules
- Declassification [3 ways]
  - Guest functions [Temporary privilege escalation]
  - Instance access [Protective declassification]
  - Transitive access [Selective declassification]

Example

- a, b, c, d are global objects
- Function f1 has full-bound access to a,b
- Function f2 is a utility function
- Function f3 is a self-sufficient function with semi-bound read only access to b, c
- d is public

```
Function f1()
{
        a.name = b; //f1 has complete access to object a.. b's access rights are added to a.name.. So now  f3 has access to a.name
        d = a; //f1 gets complete access to d and d becomes private.
        a = c; //f1 has no access to c. a's private value now is equal to c's public value. [since d still points to ex value of a, the value remains].f1 retains access to a
        f2(d,b); //works perfectly!
        f2(a, c); //no access to c since f1() has no acess
        f3(b,c); //works perfectly!
        f3(d,b); //no access to d !
        a = b+c; //Since f1 has no access to c, only b's properties are added to a after computation with c_public.f3 can access this value of a.f1 retains access to a
        a = b+d; //Since f1 has access to both a and b, an intersection of the 2 properties are added to a. so here f1 retains control of a but f3 does not.
}
```

- Function has write access to a, read access to b and c
  - Private a = Private b + Private c
  - Then, function loses write access to a and gains read access

- Function has write access to a, read access to b and no access to c
  - Private a = Private b + Public c
  - Then, function loses write access to a and gains read access

- Function has read access to a, read access to b and read access to c
  - Public a = Public b + Public c
  - [Cannot modify Private a]
  - No change in function privilege

KEEP TRACK

- SECCLOUD PROJECT

www.seccloud.cominlabs.ueb.eu

Our IFC dataset repository

git clone https://gforge.inria.fr/git/inflow/inflow.git

EMAIL: subudeepak@gmail.com