1. Senior Thesis Big Ideas

Whats needed to integrate information control flow into Twizzler

2. Papers to read

- [ ] Flume
- [ ] Flow Limited Authorization
  - Dont worry too much about the math
- [ ] Isolation Without Taxation
  - wasm / software fault isolation
- [ ] Static IFC in Rust
- [x] Capabilities for Information Flow
  - ~~lowkey probably really important~~
- [ ] On Access Control, Capabilites, Their Equivalence, and Confused Deputy attacks
- [ ] Macaroons

3. Questions

4. Twizzler Notes

- Objects
  - maybe persistent data that is identified by unique 128 bit object ID
    - used to provide a contiguous reigion of memory for semantically related data
  - mapped on demand
  - sets access policy by programming the MMU
    - what does program the MMU even mean?
  - can expand objects to be accessible across systems
  - created by the `create` syscall
    - can take in a existing object id, in which twiz will use COW
  - deleted using the `delete` syscall
    - objects are ref coutned so once reference count reaches Zero, it can be deleted
- FOT
  - exists inside each object
  - allows increased ID space without increasing pointer size
  - i think it allows for daisy chaining / russian dolling of objects
  - each entry flags have rwx permissions, that are requests which will be enforced by access control
  - names are also allowed in FOT entries
    - they say it enables late binding but im not really sure what that means
    - names are passed onto a resolving function, allowing more flexibility than unix paths
  - managed by `libtwz`(unsure how up 2 date this is)
  - `ptr_lea` (load effective address) virtual -> physical
  - `ptr_store` virtual pointer -> persistent

    - ptr translations are cached to improve performance
- Views
    - enable a program to map objects for access
    - laid out like a page table
    - each entry contains an object ID, and rwx permission bits
    - upon page-fault, fault handler tries to handle by
        - copy on write (unsure what this means)
        - checking permissions (mentioned earlier)
        - maps in an object
        - if it cant handle, raises exception to user-space
    - have 2 Syscalls
        - `set_view`: allows a thread to change to a new view
            - can let a thread execute a new program
            - jump across programs to i.e. accomplish protected task
        - `invalidate_view`: lets a thread inform kernel of changed / deleted entries
- Security Contexts
    - threads run in these
    - contain a list of access rights for objects
    - are persistent
    - implemented via virtualization hardware which maps virtual memory to

    intermediate object space which specifies access rights, which then -> physical memory.

Notes From Papers

5. Capabilities for Information Flow

- a transformation that takes any source program in a simple language, and outupts a

secure program in a language with capabilities
- static instead of runtime solution?
-

5.a. Index

1. Secure Composition

   Its a challenge to integrate unsecure third-party code with secure code. Blind integration would have security hazards so something else must be done

2. Integration vs Seperation

   Yahoo, Facebook, and Google basically take a subset of javascript and allow that to be integrated with their secure systems. They perform static analysis on the JS code, and rewrite sensitive parts and insert dynamic checks for security.

3. Capability Enforcement

   Google's model uses *object-capability model* where capabilites are unique, uncloneable references for accessing critical resources.

   "In capability systems, preventing data to flow is harder than preventing capabilities from being propogated, even if we consider overt communication channels"

Lowkey dont understand the significance of the above quote? spend some time thinking about it

4. Information-Flow policies

   Papers Goal: build on the existing work of capability enforcement, while providing "mashup" designers (designers who combine multiple systems, secure or not), with a light and abstract way to specify security policies in source programs.

   - **EXAMPLE**

     Loan calculator application that opertes on secret data, but collects statistics about feature usage. Natural security policy is to assume the income is a secret source, the statistics gathering as a public sink, and require *noninterference* (all outputs to the public sink are independent of inputs from secret source).

5. Transformation

   given an arbitrary source program in a simple imterative language, produces a secure program in a language with capabilities.

   Two types of flows

   1. Explicit Flows: when secret data is passed to a public destination
      - To assign an expression to a variable, you require a capability to write to the variable. If expression contains a secret, the transformation ensures that no cap to write the value of the expression to the public sink is passed.
   2. Implicit Flows : information is leaked through branching on secret data and exposing different publicly-observable behavior in branches.
      - When entering branches of a conditional with secret guard, or body of loop

      with secret guard, ensure that no capability to write the value of the expression to a public variable is passed.
      - How are we ensuring that its not vulnerable to timing attacks based on secret value?
        - i.e. the secret expression could proportional to # of loop iterations.
        - is this relevant towards information flow control or something else? pretty sure it is
   3. information flow can be tracked by controlling capabilities of code segments
      - all that is needed are references and scopes
      - something about this statement feels rusty

6. Security and Premissiveness

   1. regardless of the source, the transformation output satisfies security conditions of non-interference
      - Does that mean its not capable of expressing other security clauses?
   2. the transformation is as permissive as dynamic monitoring, and the transformed program adheres to the same behavior as the source program.
   3. their transformation can be seen as the inlining of a reference monitor into the program.

5.b. Core

- AFAIK their security model only targets non-interference, blocking the notion of release when appropriate.

- paper mostly talks about the rules and semantics behind their transformation.
- uses their inlined transformation to perform capability checks at runtime, with global state maps that hold “referential” capabilites infront of memory, protecting writes outward to public sinks.