Memoryless Subsystems

Whoa, what does it mean?

A memoryless subsystem is a program or procedure on a computer utility which is guaranteed to have kept no record of data supplied when it has completed its task.

Example: An Income Tax program which requires confidential data such as the income and expenditure of a customer, but must not keep a record for anyone else to see later.

So, what is the problem?

Customer wants to run a subsystem requiring him to provide information — some confidential / private and some non-private.

The subsystem cannot be inspected and so appears to the user as a black box.

The system generates output on several streams, containing either private information or non-private information.

The usual approach

Customer encapsulates the program (which appears as a black box), and inspects all output streams.

Fails in general. No algorithm for deciding whether information is contained in an output stream as the program can use a large number of means to generate its output.

What needs to be done is to provide hardware support so that, regardless of how the execution of a program proceeds, its output can in no way depend on private input.

Abstract computer model

Minsky machine!

Instructions:

1. a′ a := a + 1
2. a⁻(n) if a = 0 then goto n else a := a — 1
3. halt stop and display output

Data marks

Storage location has a fixed data mark. If x is a storage location, x is the constant data mark.

M	null	priv
null	null	priv
priv	priv	priv

Information extracted from a register has the data mark of the register attached.

But wait! The position in the program contains information!

New instructions to save the day

Instructions:

1. x' — if x = M(x, p) then x := x + 1
2. $\bar{x}(n)$ — if x = 0 then (if p = M(x,p) then goto n)
   else (if x = M(x, p) then x := x - 1)
3. x*(n) — if x = 0 then (stack(p, p); p := M(x, p); goto n)
   else (if x = M(x, p) then x := x - 1)
4. Return — p, p := unstack()
5. Halt — if p = null then halt

Solution requirements

Theorem 1: A system is secure if and only if the null path (the path through the program while p = null) of the program cannot depend on any priv information.

Suppose the null path can depend on priv info. The implication is that the program can follow two distinct branches depending on priv info.

That is not good at all.

Solution requirements

Theorem 2: While p = priv, there is no way of altering register x for which x = null.

x′	M(x,p) = M(null, priv) = priv ≠ x. Instruction has no effect.
x¯(n)	a) x = 0. Since M(x,p) = priv = p, program jumps to n. b) x > 0. M(x,p) = priv ≠ x. Instruction has no effect.
x*(n)	a) x = 0. Jumps to n irrespective of x and p. b) x > 0. M(x,p) = priv ≠ x. Instruction has no effect.

Solution requirements

Theorem 3: If p = null, no change of path (jump) can take place that depends on priv information without setting p = priv.

Jump can only occur when the register is zero. Let y be a register with y = 0 and y = priv.

1. $y^{-}(n)$ M(y, p) = M(priv, null) = priv ≠ p. Does not jump.

2. y*(n) Jumps irrespetive of p and y. However,
p := M(y, p) := M(priv, null) := priv

Solution requirements

Theorem 4: If p = priv, p can only be reset to null by a Return instruction.

Only a*(n) and Return instructions alter p.

If p = priv, an a*(n) instruction with a = 0 will set p as:

$$p = M(p, a) = priv$$

regardless of the value of a.

Solution requirements

Theorem 5: Suppose the machine started with p = null and that it has halted. Then the system is secure.

The null path cannot depend on any priv info.
- By theorem 2, cannot change any null register while in priv state.
- By theorem 3, cannot sense any priv info w/o moving into priv state.
- By theorem 4, must have returned, since it has halted and is in null state.
- After a return, it must continue executing program immediately after a*(n) instruction and continue along null path. No null registers have been changed. Must be independent of entry into priv routine.

- By theorem 1, the model is secure.