CS 380S - 0x1A Great Papers in Computer Security
Fall 2012

MIDTERM

November 1, 2012

DO NOT OPEN UNTIL INSTRUCTED

YOUR NAME: ______________________________

Collaboration policy

No collaboration is permitted on this assignment. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade.

Midterm (45 points)

Problem 1 (3 points)

Why is calling `free()` twice on the same memory object in a C program a potential security problem?

Problem 2

In the DEC Alpha assembly language, all instructions are 4-bytes wide and must start on an aligned 4-byte boundary. Here are some examples:

- `br` *Ra*, *disp*
  An unconditional relative branch. This instruction stores the address of the next instruction in *Ra* and then skips *disp* instructions, where *disp* may be negative. For example, `br r13, -5` jumps back 5 instructions (this may happen in a loop, for example).

- `jmp` *Ra*, (*Rb*)
  Jump to register. Stores the address of the next instruction in *Ra*, then jumps and starts executing code at address *Rb*.

- `ldq` *Rv*, *disp* (*Ra*)
  Load. Takes the memory address contained in register *Ra*, adds *disp* to it, and loads the value of the memory location at this address into register *Rv*.

- `stq` *Rv*, *disp* (*Ra*)
  Store. Takes the memory address contained in register *Ra*, adds *disp* to it, and stores the value of register *Rv* into the memory location at this address.

- `bis` *Ra*, *Rb*, *Rc*
  Compute bitwise OR of *Ra* and *Rb* and store it into *Rc*.

- `and` *Ra*, *Rb*, *Rc*
  Compute bitwise AND of *Ra* and *Rb* and store it into *Rc*.

Problem 2a (3 points)

Fault isolation requires inserting special checking code before every *unsafe* instruction, *i.e.*, an instruction that may potentially write or execute memory outside the fault domain. For example, a store instruction `stq` *Ra*, 0(*Rb*) is unsafe if it cannot be statically checked that the address contained in *Rb* is within the fault domain's data segment.

In the following list, circle the instruction(s) which can be unsafe:

- `br` *Ra*, *disp*   where *disp* falls within the fault domain's code segment.
- `jmp` *Ra*, (*Rb*)
- `ldq` *Rv*, *disp* (*Ra*)
- `bis` *Ra*, *Rb*, *Rc*

Problem 2b (2 points)

Suppose that the unsafe store instructions are "sandboxed" as follows. We use dedicated registers $r20$ and $r21$ to store, in the positions corresponding to the segment identifier part of a memory address, all-zero bits and the segment ID bits, respectively. If the code contains an unsafe store instruction `stq r2, 0(r1)`, it is replaced by the following three instructions:

```
and r1, r20, r1
bis r1, r21, r1
stq r2, 0(r1)
```

How can you subvert the safety of the system that uses this sandboxing mechanism?

Problem 2c (3 points)

Suppose communication between fault domains is implemented as follows. For each fault domain, the trusted execution environment inserts special "stubs" (little snippets of code) into a special region of that domain's code segment. Because the code of the stubs is trusted, it may contain unsafe instructions. Furthermore, the stubs are the only part of the fault domain's code segment that is allowed to have instructions branching outside of this code segment.

When a trusted caller calls an untrusted function, it branches to the "entry" stub, which copies arguments, saves registers that must be changed when switching fault domains, and passes control to the untrusted code. When the untrusted code returns, it jumps directly to the "return" stub in its code segment, which restores the context and returns to the caller.

How can you subvert the safety of the system that uses this cross-domain communication mechanism?

Problem 2d (3 points)

How should you implement the "stubs" for cross-domain communication so that they cannot be subverted? You may explain or draw a picture.

Problem 3 (6 points)

Imagine a static-analysis tool for checking source C code to ensure that it satisfies a certain set of rules. Each rule is expressed by a finite-state automaton, with a special ERROR state. As the checker scans the code, it keeps track of the current state in the automaton. If a state labelled ERROR is ever reached, then the checker reports an error in the code.

Draw finite-state automata representing the following security rules. If you believe the rule cannot be expressed by a finite-state automaton, explain why.

- Immediately before each call to `strcpy(dest,src)`, the program must check the length of `src` by calling `strlen(src)`.

- Each temporary file used by the program must be created using `mkstemp()`, written, and eventually closed.

- The return value of every call to `malloc` must be immediately checked to ensure that it is not NULL.

Problem 4

Problem 4a (2 points)

Suppose an HTTPS page links to an HTTP iframe where both are loaded from the same domain. The browser shows a mixed content warning dialog. Explain the risk of clicking OK on this dialog.

Problem 4b (2 points)

Suppose an HTTPS page links to an HTTP iframe where the two are loaded from different domains. Should the browser display a warning dialog? Explain.

Problem 5 (3 points)

Tatebayashi, Matsuzaki, and Newman (TMN) proposed the following protocol, which enables Alice and Bob to establish a shared symmetric key K with the help of a trusted server S. Both Alice and Bob know the server's public key K_S. Alice randomly generates a temporary secret K_A, while Bob randomly generates a new key K to be shared with Alice. The protocol then proceeds as follows:

$$\begin{array}{llll} \mathit{Alice} & \rightarrow & \mathit{Server} & \mathit{enc}_{K_S}(K_A) \\ \mathit{Bob} & \rightarrow & \mathit{Server} & \mathit{enc}_{K_S}(K) \\ \mathit{Server} & \rightarrow & \mathit{Alice} & K \oplus K_A \\ & & & \text{Alice recovers key } K \text{ as } K_A \oplus (K \oplus K_A) \end{array}$$

In this protocol, Alice sends her secret to the Server encrypted with the Server's public key, while Bob sends to the Server the new key, also encrypted with the Server's public key. The Server XORs the two values together and sends the result to Alice. Therefore, both Alice and Bob know K.

Suppose that evil Charlie eavesdropped on Bob's message to the Server. How can he, with the help of his equally evil buddy Don, extract the key K that Alice and Bob are using to protect their communications?

Assume that Charlie and Don can engage in the TMN protocol with the Server, but they don't know the Server's private key.

Problem 6 (4 points)

Integrity is an important element of an information flow policy. Suppose there are two levels of integrity, T for Trusted and U for Untrusted. Intuitively, untrusted data should not be allowed to corrupt trusted data. That is, data from untrusted variables should not be allowed to flow to trusted variables.

Examine the following four statements, which have integrity labels as subscripts on variables. Explain which statements are secure, which are insecure, and why.

1. $X_T := Y_T + Z_U$

2. $V_U := Y_T + Z_U$

3. if X_T then $Y_T := X_T$ else $V_U := Z_U$

4. if V_U then $Y_T := X_T$ else $V_U := Z_U$

Problem 7 (4 points)

Consider the following lattice of security labels (the higher the label, the more confidential the information):

```
         TopSecret
        /        \
   Classified  Secret
        \        /
          Public
```

and consider the following pseudocode:

```
Public     w
Classified x
Secret     y
TopSecret  z

p = w - x
if (y != 0) then
   q = 1
   if (p == 0) then
      r = 0
   endif
else
   s = 1
endif
t = z - z
```

For each of the variables `p, q, r, s, t`, write the *minimal* security label that it can be given so that the above code is secure according to the Bell-LaPadula model.

Problem 8

Consider building an inline reference monitor for preventing undesirable information flows. The monitor is added to a program by rewriting its source code. The goal of the monitor is to track the flow of secret inputs through the code and stop execution if the program is about to reveal one of these secrets through a public output.

Problem 8a (2 points)

What are the issues in implementing such a reference monitor?

Problem 8b (2 points)

Suppose the program is certified using the method described in the Denning and Denning paper. How does this change your answer to the previous question?

Problem 9

A typical delegation certificate enables a machine whose public key is K to perform actions with the authority of user U. Informally, this certificate states: "If K says it is speaking for U, believe it."

Problem 9a (3 points)

What is the technical meaning of "say" and how does the recipient verify that it was a particular machine that said something?

Problem 9b (3 points)

Suppose the delegation certificate stated instead “K speaks for U”. What would be the security consequences?