Assignment 3: The Memory Sandbox
15-316 Software Foundations of Security and Privacy

Total Points: 50

1. **Practice makes perfect (10 points).** Use the read-over-write axioms to prove the validity of the following using a sequent calculus deduction:

$$0 \leqslant k < U, j \neq k, i = j \vdash \texttt{Mem}\{i \mapsto e\}\{j \mapsto f\}(k) = g \rightarrow \texttt{Mem}(k) = g$$

Solution.

2. **Unfinished business (10 points).** In lecture 7, we discussed two cases of the structural induction used to prove the security of SFI. Complete the inductive case for `while` commands. That is, assuming that Equation 1 is valid for α whenever $0 \leqslant s_l \leqslant (x \,\&\, s_h) \mid s_l \leqslant b_h < U$:

$$\forall i. \neg(s_l \leqslant i \leqslant s_h) \wedge \mathtt{Mem}(i) = v_i \rightarrow [\alpha]\mathtt{Mem}(i) = v_i \tag{1}$$

Prove that it is also valid for $\mathtt{if}(Q)\,\alpha\,\mathtt{else}\,\beta$.

Solution.

3. **Leaky sandbox (30 points).** Consider the following language, which resembles a simplified assembly language.

$\texttt{and}(x, y)$	Take the bitwise-and of variables x and y, store the result in x
$\texttt{or}(x, y)$	Take the bitwise-or of variables x and y, store the result in x
$x := c$	Copy a constant c into variable x
$x := y$	Copy the value stored in y to x
$x := \texttt{Mem}(y)$	Read the memory at address stored in variable y, save result in x
$\texttt{Mem}(x) := y$	Store the value in y at the address pointed to by x
$\texttt{if}(Q)\ \texttt{jump}\ x$	If Q is true in the current state, jump to the instruction pointed to by x

Programs in this language are sequences of instructions indexed on integers 0 to n, and we refer to the instruction at index i of program α with the notation α_i. Note that there are no expressions other than constants and variables in this language. Instead, results of operations are stored in variables, and can be moved into memory when necessary. Think of variables as acting like registers, so to implement the computation $w := (x \,\&\, y) \mid z$ from our language in lecture we would write the program:

$$\begin{array}{ll} 1: & \texttt{and}(x, y) \\ 2: & \texttt{or}(x, z) \\ 3: & w := x \end{array}$$

It is *not* possible to write $w := \texttt{or}(\texttt{and}(x, y), z)$ because neither $\texttt{or}(\texttt{and}(x, y), z)$ or $\texttt{and}(x, y)$ is a variable, and updates to variables can only be written with other variables, constants, or memory reads on the right hand side.

Note that just as you should assume that any memory reads outside the bounds of $[0, U)$ will result in an aborted trace, you should assume that any attempt to `jump` to an address outside the bounds of $[0, N)$, where N is the number of instructions in α, will also abort the trace.

Part 1 (15 points). We want to implement a sandboxing policy for this language using software fault isolation. So the proposal is to replace all memory read and write operations as follows. Assume that $s_l = \texttt{0x15316000}$ and $s_h = \texttt{0x15316fff}$, so the memory sandbox is contained in the range of addresses $\texttt{0x15316000} - \texttt{0x15316fff}$.

$x := \texttt{Mem}(y)$	becomes	$\texttt{and}(y, \texttt{0x15316fff})$ $\texttt{or}(y, \texttt{0x15316000})$ $x := \texttt{Mem}(y)$
$\texttt{Mem}(x) := y$	becomes	$\texttt{and}(x, \texttt{0x15316fff})$ $\texttt{or}(x, \texttt{0x15316000})$ $\texttt{Mem}(x) := y$

Additionally, we want to prevent jumps from leaving a code sandbox restricted to the range of instruction addresses $\texttt{0x00000a00} - \texttt{0x00000aff}$. So each indirect jump is rewritten as follows.

$\texttt{if}(Q)\ \texttt{jump}\ x$	becomes	$\texttt{and}(x, \texttt{0x00000aff})$ $\texttt{or}(x, \texttt{0x00000a00})$ $\texttt{if}(Q)\ \texttt{jump}\ x$

Any untrusted code is rewritten using these rules prior to being executed. Unfortunately, we didn't have time to prove that this implementation of SFI is secure.

Explain why this instrumentation is vulnerable to memory reads and writes outside the memory sandbox, and provide an example program in the language that exploits violates the policy. For full credit, be sure to explain in words how your example results in a violation of the sandbox policy.

Solution.

Part 2 (15 points). Propose an alternative implementation in this language for the policy in Part 1 that is secure. You may assume that the untrusted code is not allowed to modify some variables that you select, but be sure to clearly state this and any other assumptions that your solution requires.

Solution.