Assignment 1: Safety & Proof
15-316 Software Foundations of Security and Privacy

Total Points: 50

1. **The least significant vulnerability (10 points)** In lecture we saw several examples of widely-used memory safety defenses that fall short of providing a rigorous guarantee. But even defenses that seem strong in principle can leave unprotected vulnerabilities if the defense itself is not implemented correctly.

   Consider a memory safety policy that aims to restrict an untrusted program from reading or writing outside the range `0x8000300`–`0x8000400`. Such a policy would be important on a platform that does not provide a virtual memory system, so all processes must share the same address space, and each is given a specific range to work with. To enforce this policy, a runtime monitor interprets each statement of a C-like language, keeping track of the values of variables and pointers. Whenever a statement would cause a pointer to be dereferenced, the monitor checks its state to determine the address that the pointer refers to, and terminates the untrusted program if executing the statement would violate the above policy.

   For example, suppose that a malicious program author knew that a variable `y` had been allocated at `0x80003F2`, `0xF` bytes below the top of the policy region `0x8000400`. The following attempt to violate the policy by reading as many bytes past `y` would fail. As shown on the right, the monitor keeps track of each variable's address and value, so before a statement executes the monitor can terminate the program, as it would do before the third line is executed: the monitor's state reflects that `x` points to `0x8000401`, so dereferencing this pointer would violate the policy.

```
int *x = &y;
x += 0xF;
printf("Secret is: %d\n", *x);
```

	Monitor state			
	x	&x	y	&y
initial	0x0000000	0x8000300	0x0000000	0x80003F2
after line 1	0x80003F2	0x8000300	0x0000000	0x80003F2
after line 2	0x8000401	0x8000300	0x0000000	0x80003F2

   However, the monitor has a bug that causes it to track the results of bitwise-or operations incorrectly. Regardless of the operands passed to the bitwise-or, the monitor's state for the result will always reflect a value with `0` in the least-significant bit.

   Show how this can be exploited by providing a program that will violate the policy when executed alongside the buggy monitor, by writing to address `0x8000480`. Remember to also explain in clear language how your program takes advantage of the bug, and why you think that it will violate the policy. If it clarifies your answer, please provide a statement-by-statement summary of the monitor's state as in the example above.

2. **Sheffer stroke (25 points)** The Sheffer stroke is a binary operation over propositions that is true exactly when at least one of its arguments is false. Formally, its semantics are:

$$I \models P \uparrow Q \text{ iff } I \not\models P \text{ or } I \not\models Q$$

First, provide inference rules for the Sheffer stroke by filling the elided content below (5 points).

$$(\uparrow\text{L})\ \frac{\ldots}{\Gamma, P \uparrow Q \vdash \Delta} \qquad (\uparrow\text{R})\ \frac{\ldots}{\Gamma \vdash P \uparrow Q, \Delta}$$

Then, prove that your rules are sound by arguing rigorously, and separately for each rule, that the validity of the premises implies the validity of the conclusion (20 points).

Solution.

3. **Functional completeness (15 points)** There is more to the Sheffer stroke than initially meets the eye: it is one of only two propositional operators that is *functionally complete*, so it plays an important role in several practical applications of logic. Functional completeness means that all possible operations over propositions can be defined in terms of Sheffer stroke. Thus, we would not lose anything essential if we were to insist that formulas only contain atomic propositions and $\uparrow$ connectives, and this would allow us to define semantics and proof rules for just this one operator.

   The following equivalences show how the propositional connectives covered in lecture can be expressed in this way.

$$\begin{array}{ll} \textit{negation}\ \neg P \leftrightarrow P \uparrow P & \textit{implication}\ P \rightarrow Q \leftrightarrow P \uparrow (Q \uparrow Q) \\ \textit{conjunction}\ P \wedge Q \leftrightarrow (P \uparrow Q) \uparrow (P \uparrow Q) & \textit{disjunction}\ P \vee Q \leftrightarrow (P \uparrow P) \uparrow (Q \uparrow Q) \end{array}$$

$$\textit{equivalence}\ P \leftrightarrow Q \leftrightarrow (P \uparrow Q) \uparrow ((P \uparrow P) \uparrow (Q \uparrow Q))$$

   First, show that the above equivalence for implication ($\rightarrow$) is valid by using your rules from the previous question to write a sequent calculus proof (10 points). Note that it can be difficult to typeset proofs that use $\leftrightarrow$R, so you may break your answer into two proofs instead, showing $P \rightarrow Q \rightarrow P \uparrow (Q \uparrow Q)$ and $P \uparrow (Q \uparrow Q) \rightarrow P \rightarrow Q$ as separate trees.

   Then, *explain* how you would use parts of your proof to show that the following rules are sound by derivation (5 points).

$$(\Rsh\text{L})\ \frac{\Gamma, P \uparrow (Q \uparrow Q) \vdash \Delta}{\Gamma, P \rightarrow Q \vdash \Delta} \qquad (\Rsh\text{R})\ \frac{\Gamma \vdash P \uparrow (Q \uparrow Q), \Delta}{\Gamma \vdash P \rightarrow Q, \Delta}$$

   You are not required to explicitly provide a derivation, but state which part of your proof from earlier you would use, and any other rules that are needed to incorporate your earlier proof. In addition to the inference rules presented in lecture, you may refer to either of the weakening rules WL,WR shown below.

$$(\text{WL})\ \frac{\Gamma \vdash \Delta}{\Gamma, P \vdash \Delta} \quad (\text{WR})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash P, \Delta}$$

   Solution.