CSci 5271: Introduction to Computer Security

Exercise Set 5 **due: Friday, December 6th, 2019**

Ground Rules. You may choose to complete these exercises in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it; use the Canvas groups mechanism to indicate your group members if there is more than one. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or textbook. An electronic PDF copy of your solution should be submitted on the Canvas by 11:59pm on Friday, December 6th.

1. Virus Virii. (25 pts) Sam has invented a brand-new virus detector, ViruSniff, and he claims it is "100% effective" — if executable file F is a virus, then ViruSniff(F) will output "`VIRUS!!!`".

(a) Does ViruSniff's claim conflict with the undecidability of the halting problem? Why or why not? (Hints: is there another term besides "effectiveness" that describes that statistic that Sam claims is 100%? Is there a simple program that can do exactly what Sam says ViruSniff can do?)

(b) Unlike the hypothetical considered in (a), in fact some hackers reverse engineer ViruSniff and post its algorithm online. It turns out that ViruSniff does processor emulation of the first 10000 instructions of an executable, and then applies a fancy signature matching algorithm (that no one seems to understand) to the sequence of instructions and memory changes to decide if the program is a virus or not. Explain how to change any program that runs for at least 10001 instructions, and does not trigger the `VIRUS!!!` alert, to propagate a virus such that the altered program will also fail to trigger the alert. What does your strategy say about Sam's claim?

(c) Given your knowledge of the attack from (b), how might you enhance ViruSniff to work against the new virus-writing strategy? Evaluate the potential effect of your change on the false-positive rate.

2. Denial of Service Denial. (20 pts) Sly is concerned about the possibility of DoS attacks against his web server program.

Sly has developed a new module for his web server that he claims will prevent DoS attacks by slowing them down. In Sly's module, every incoming HTTP request is put into a queue, with a timestamp and a "delayed" bit marked as false. When it is ready to serve a request, the web server takes the first request in the queue. If the "delayed" bit is false and there are no other requests from the same IP address in the queue, it serves the request immediately. If the "delayed" bit is false and there is at least one other request from the same IP address in the queue, the "delayed" bit is set to true and the request is re-inserted at the end of the queue. If the delayed bit is set to "true," then the request is served **if** the current time is at least 1 second greater than the request timestamp, and **otherwise** the request is sent to the end of the queue again. Sly's idea is that this will allow the site to deal with requests from legitimate users in preference to DoS attack requests.

Will Sly's scheme work to prevent a DoS attack from making his web server unusable by normal users? Give a detailed explanation.

3. Remailer doppelgangers. (20 pts) The "Sybil" attack is a general attack on security protocols that involve many computers or identities. The basic idea of the attack is to acquire as many identities as necessary to violate whatever assumptions the protocol makes about parties working together. Anonymity schemes could potentially make such attacks easier, although many of the most popular schemes make it fairly easy to prevent this (for example, it's easy to block Tor users from using your website at all, if you want). On the other hand, many anonymity schemes can themselves be vulnerable to Sybil attacks.

Recall that remailers are anonymous email servers that essentially implement a cascade of mixes. The basic mix cascade work as follows: each node assembles a "batch" of messages to decrypt and jumble together. If the batching works by waiting until N messages are received, the $N-1$ attack can be applied: the adversary sends $N-1$ messages to the mix, whose destinations he knows. Then when a sender sends the N^{th} message, its destination is obvious. One possible defense against this is for the mix to wait to mix a batch until it has seen messages from K different senders. Explain why the Sybil attack makes this defense ineffective.

4. Vote (often) by mail. (15 pts) The reason many security folks and cryptographers who work on voting object to "vote-by-mail" or widespread use of absentee ballots is the possibility of *coercion*: it is easy to "sell" your vote (where the price could be such things as lack of physical or mental harm, or continued employment, instead of cash) because the "buyer" can watch you fill out your ballot and mail it in. One commonly proposed countermeasure to this attack is to allow each voter to cast multiple ballots, with only the most recently submitted ballot being counted. Discuss some of the trade-offs involved with this defense. If you were a vote buyer in an election with this defense deployed, what might you do? Can you think of any other negative side-effects?

5. Cut and Choose. (20 pts) Many cryptographic voting protocols use something called a "zero-knowledge proof scheme" at some point in the protocol to convince one party that another party has followed the protocol. For example, one party (the "prover") may need to prove she knows a certain secret without revealing the secret to the other party (the "verifier").

A core idea in many of these protocols is the "cut and choose" technique in which the prover produces two options for the verifier: if (and only if) the prover can guess which option the verifier will choose, she can "cheat" the other player. As long as the verifier follows the proof protocol, he can only be fooled with probability $\frac{1}{2}$. Repeating this process many times can make it exponentially difficult for the prover to cheat.

We can also apply this principle to a paper-ballot system. Suppose that a state-level authority sends ballot boxes to each precinct in the state. These boxes are locked to prevent the precincts from tampering with them, e.g. by removing votes. However this leaves the possibility of a converse problem: how do the precinct authorities know that the locked boxes delivered to them are empty? A malicious state-level authority could "pre-stuff" the ballot boxes before they even get used. On the other hand, if we just gave the precincts keys, they could open the boxes to see that they were previously empty, but then they could stuff the boxes themselves.

Suppose the "acoustic side channel" of shaking the boxes to hear if anything rattles is out of bounds. Describe instead how to use a cut and choose protocol to prevent the state-level authority from "pre-stuffing" the ballot boxes. Specifically, if there are k precincts, your protocol should allow the state authority to cheat with probability at most 2^{-k}.