1. TCP-Unfriendly. TCP’s “congestion control” mechanism relies on users to respond appropriately to network congestion by backing off their sending rate. One potential problem with this mechanism is what’s called (by economists) the “tragedy of the commons”. Suppose Alice knows that everyone else obeys TCP’s congestion control mechanism. Then if she continues sending at the same rate, ignoring congestion control, everyone else will slow down a little bit more and she will get better service from the network. So Alice has no motivation to obey TCP congestion control (other than the fact that not doing so involves finding or writing her own TCP stack - details, details) and in fact neither does anyone else. But if no one obeys the mechanism, the network (commons) becomes useless, a tragedy.

(a - 10 pts) Bob the Network Builder has an idea about how to solve this problem. He reasons that congested routers can see the exact state of a TCP connection. So if a particular connection does not slow down in response to dropped packets, or speeds up it’s sending rate too quickly, the router can send a RST packet to the host that is not obeying the “Additive Increase, Multiplicative Decrease” rules. This will cause the connection to close, much more painful than just dropping an odd packet or two. From a security standpoint, what’s the problem with Bob’s idea – that is, if I’m an adversary intent on sending at a high rate, can I circumvent this mechanism? Are there adjustments to this mechanism that might limit the number of instances where your attack is viable?

(b - 10 pts) When Bob realizes that reset packets aren’t sufficient, he proposes a more direct attack: blacklisting. Under this idea, routers that notice TCP senders that don’t respond to dropped packets appropriately will just stop routing packets for that sender. Why is both ineffective against adversaries and a generally bad idea if adversaries got wind of it.

2. Denial of Service Denial. Sly and Carl are really concerned about the possibility of DoS attacks against their web server program. Since one way to defend against DoS attacks is to make the attacker do more work, 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. This approach extends the time needed for an attacker to fill the web server’s request queue.
Inspired by BitTorrent, Carl has a different suggestion for preventing DoS. In Carl’s solution, whenever client C downloads a page, he also downloads an ActiveX control that acts as a mini web server for that page and its contents only. Then when the main server starts to be overloaded, it uses HTTP redirects to point new clients to servers running on old clients. The new clients can then download the pages from old clients directly, without using any more of the main server’s bandwidth.

(a - 10 pts) Will Sly’s scheme work, or not, and why? Give a detailed explanation.

(b - 10 pts) Will Carl’s scheme work? Why or why not? (Note that there are clearly some implementation issues to address, such as avoiding the use of clients that are unreachable due to firewalls or closed browsers, but let’s assume these are adequately solved)

3) Virus Virii Sam has invented a brand-new virus detector, ViruSniff, and he claims it is 100% effective - if executable \( F \) is a virus, then \( \text{ViruSniff}(F) \) will output “VIRUS!!!” (i.e. it will have no false negatives).

(a - 2 pts) Does ViruSniff violate the undecidability of the halting problem? Why or why not? (Hint: is there a simple program that can do exactly what Sam says ViruSniff can do?)

(b - 4 pts) 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 - 4 pts) Let’s pretend that ViruSniff’s emulation signature matching turns out to be very very accurate (no false negatives), but very very slow. Imagine that your boss will only let you slow down the company computers so much, meaning that you can only emulate and signature match a fixed number of instructions. How would you go about changing ViruSniff, still leveraging that signature detection, in order to detect a virus that uses your evasion strategy from part B? Is there a way to still have a system that provides no false negatives? What does such a system do to the false positives? Is there a way to build the system to have a trade-off between false positives and false negatives that your original evasion strategy can not function perfectly against?
**Paper Review** If you are a graduate student, you are expected to also include with this exercise set a paper review of one of the papers posted as graduate reading over these two weeks. Your paper review is not supposed to be a full paper in and of itself, but rather a summary in your own words and some evidence that you’ve thought about what the paper is trying to do, if they achieve their goals, how useful their goals are, and how it could be improved. Your paper review should have the following components.

- A brief summary *in your own words* about what the paper was about. You should cover what problem the paper was trying to solve or highlight, how the authors proposed achieving their goals, and how the authors evaluated their success. This should NOT simply be a re-statement of the abstract hit with a thesaurus until un-recognizable, I do not mind if you did not fully understand the paper, if something was confusing, or you did not understand a concept, feel free to state that. (1-2 paragraphs)

- State one problem with the paper. Every paper has a problem, be it an evaluation that does not actually measure what it should, a poor assumption, an incorrect threat model, an obvious attack/defense, etc. Find one thing that you think is a flaw in the paper, and briefly explain why you think it is a flaw. (2-3 sentences)

- State two possible things that could be done for future work. This could be expanding the attack/defense to work under different assumptions or in different deployments. It could be a different set of evaluations. It could be an improvement to how the system is constructed. Try to be specific. (1-2 sentences each)

Choose a paper from the weeks listed as “Botnets/DDoS” or “Malware”.
