

# CS 294-28 Lecture 8: NIDS Evaluation

Scribed by Matei Zaharia

September 23, 2009

## The Problem of Funders

- From *The Right Stuff*: “What makes rockets go up? Funding.”
- The paper frames that the DARPA study wanted a single evaluation metric. *There’s no good answer for that, regardless of how well the Lincoln group might have done.*
- PM (program manager) = the person who funds you
- “Food chain”: everyone has a boss, including the PM; their boss, who may not understand network security / metrics / etc as well as the PM, wants something simple to understand and later use to show improvement
- Project management effort scales sublinearly with funded group size (e.g. overseeing \$3M project isn’t 10x more work than overseeing \$300K project); thus PMs prefer large projects
- *The Mythical Man Month*: if 1 hacker can do it in 14 months, can 14 hackers do it in 1 month?
  - The (classic) book says that as you add people to a project that’s late, it goes slower
  - Corollary: research quality scales sublinearly with group size

## Deep Problems Pointed Out in the Paper

- Problem 1: Desire for single metric
- Problem 2: Rich & diverse system behavior
- These two desires are very much at odds with each other
- What we really want is illumination/insight: understand how things work in one context to apply them to another context
- Science: Note that this paper is doing something scientific (a critique) in Computer Science, yet must apologize up front for it because this is so uncommon in our field
  - Computer Science—at least for systems / networking / security—has virtually no reproduction of results

- Furthermore, we have a lot of “personally irreproducible results”: effort is deadline-driven, so we do a lot of last-minute futzing, remeasuring and filtering; then, 3 months later, once the paper is accepted, you can’t even reproduce the original results
- Later papers from the Lincoln group don’t cite McHugh...
- Separating data for development and assessment: Lincoln did this right in the benchmark they created, but other systems/networks work often doesn’t do it
  - \* Note: you usually need a lot more data for development than you expect; don’t make the mistake of being stuck with too little development data
- How to get data for security problems?
  - Very deep problem... not sure how any data you have will generalize
  - Often something developed on one data set won’t work on another
- Raw data is not enough; we also need metadata (info about the data): for example, about a packet trace, we might want:
  - Time and date of trace
  - Loss rate (how can we know this?)
  - Bias (e.g. was the trace captured behind a firewall?)
  - Hostnames (if the trace has only IP addresses, hard to identify hosts a few months later)
  - Known events that occurred during the trace (e.g. internal scans)
- Network security as a field suffers from lack of data (largely due to privacy)
- Anonymizing trace data is generally hard; there is a tradeoff between research utility and risking deanonymization
  - Even if we wanted to do something like renumbering IP addresses, one problem is that there are often scans that go through addresses in numerical order, which can deanonymize the addresses
  - Even releasing vendor codes of MAC addresses can identify specific systems
  - Something like releasing a trace of JavaScripts seen in HTTP traffic might be impossible
  - Netflix prize data was deanonymized using IMDB
- Another model is to have controlled access to data, e.g. DHS’s PREDICT
  - Was hugely delayed by legal + public perception concerns
- Finally, you can use a model where you send a program to an organization and they run it over the data and send you the output
  - The organization may re-run it after changing some fields and diff the output to make sure you aren’t leaking information about them (Vern’s work)
- Spectrum of choices for data sets: Private, Controlled, Public, Synthetic
- On synthetic data:

- KDD Cup data: it is a “kiss of death” to use this in submissions to a good conference
  - \* This data is a distilling of the Lincoln data set for machine learning into features
  - \* Apart from having all the difficulties of the Lincoln data set, another problem is that some features are very specific (e.g. was there a packet with a certain source and destination pair that indicates a particular DOS attack), which means that any ML performed on this is not generalizable
  - \* Later revealed that some attacks were identifiable just by their packets’ TTL ... but many AI algorithms submitted about these were opaque, and authors did not look for insight/illumination until too late to find these kinds of problems
- Moral: understand not only false positives/negatives, but also true positives/negatives (why does your algorithm give the result it gives)
- Useful to contribute reusable components rather than a full system
- Snort/Bro example on comparing with others’ software:
  - Sometimes you discover really bizarre things, like Snort going slower on a faster processor (turned out to be due to cache effects)
  - It also turned out that relative performance was very dependent on data
  - How can we talk about performance in these situations? Main takeaway that they all perform roughly the same, unless you care about factors of 2.
  - Can also turn this problem into a research opportunity, e.g.. by looking at ways to predict performance of Bro or Snort based on characteristics of the data
- NIDS performance:
  - When asked “how fast can Bro go?”, what factors must be considered?
    - \* Analysis performed
      - What protocols? (events monitored and passed up)
      - What local actions? (scripts)
    - \* Traffic mix
    - \* What unit of speed do you want? (bits/s, packets/s, connections/s, requests/s?)
      - Some analysis may have heavy per-connection cost, but not look at payload
  - This is pretty complex; as researchers, we need to not only get results, but to ask how to lucidly present our results to others
- Case study: suppose we want to detect phishing by finding pages or images that are identical but are loaded from different sources; how would we go about evaluating this?
  - First need to obtain some data; for development, we can use existing labelled data, but ideally, we want to run the algorithm on some real, unlabeled corpus of data as well to measure how often phishing occurs there
  - To measure false positives, use random sampling: pick some of the pairs of pages our algorithm reports as matching and manually check whether they are okay
  - To measure false negatives, there’s no equivalent sampling trick; we can’t pick random pairs of pages from the data set and check whether they should have matched

- \* For example, with 10M pages, we have  $10^{14}$  pairs to look at, and most of these are likely to be true mismatches; we won't get an estimate of the false positive rate without looking at a large subset of the  $10^{14}$  pairs.
- \* Instead, can use external info, e.g. labelled phishing sites matched with real sites by some third party.
- \* Alternatively, can inject synthetic phishing sites and see whether the algorithm catches them.
- \* Problem: both external information and synthetic data may have bias.