So, you'd like to buy a security test tool?

This is a list of questions that you can use to grill the security tool sales people that knock on your door. (If you are a security tool sales person, this list helps you to prepare your pitch.)

Collected observations from week 3 assignment, Software Security course at University of Helsinki and Aalto University, 2014-2018.

Tool capabilities

- What are the real capabilities of the tool? Can it work with exactly your flavor of language and your frameworks?
    - Do not believe the slideware or "coming soon" claims, try it out yourself
    - Exact versions of frameworks that support is claimed for, and those you are using?
    - Can it work with your dependencies? What if you've customised your dependencies' code?
- Especially with dynamic (run-time) analysis, how does the tool get to "see" parts of your software that are hidden behind a lot of business logic or GUI interaction? Will it miss major parts of your functionality?
- Does it support more modern additions to your language and frameworks? Is it kept up-to-date?
    - Esp. open source projects can be academic projects that have been abandoned. When was the last commit made into an open source tool? Has it kept up-to-date with new security research?
    - Vendors that acquire tools (specifically acqui-hire) might just put the tools into deep freeze and milk them until nobody buys them; past "brand names" may be soon ghosts
- What's the real-life false positive rate for *your* kind of code (not a toy example drawn from an open source project)? How can the vendor provide you with this data?
    - In addition to false positives, does it report a bunch of issues that are theoretically valid findings, but in practice impossible to exploit?
    - Will it produce such an avalanche of bugs at first that everyone will be mortified and demotivated? Can you tell which ones of those are really important?
- How tight is the feedback loop to developers? This has multiple sides to it:
    - It may be best if the feedback comes immediately in the editor / IDE, as the developer still has the context and can learn the most, and fixing the bug is fast (no need to recall the mental context)
    - With rapid deployment cycle, one might not have the luxury of waiting for a long time for results – think about deploying 10 times a day; can you wait 12 hours for the results?
- Can the tool work with something that is not finished yet?
    - E.g., does not need a compiled, built and executing target to be useful?

- How late in the development cycle does the tool come in? Is this where you would like it to come in?
    - Usually, stuff you find earlier in the development cycle is cheaper to fix than late in development cycle. Stuff that is found in a penetration test may be the most expensive to fix.
- How expensive is it to fix the issues found by the tool (early vs. late, quality of details provided by the tool)
    - Are the results actionable, or do they necessitate a lot of studying or are difficult to reproduce?
    - Are the results understandable to your own staff, or would you have to rely on outside help? What would this cause to the Total Cost of Ownership?
- Can you run the tool in test or deployment automation, or does it require a human to use it?
- Can the tool produce results that are machine-readable?
    - Can you base your test / deployment automation gates ("promotion gates") on the tool output, or does it need a human to understand the results?

Buying / acquiring the tool

- Is there any real way in which you can make a sales argument of the tool internally in your organization to:
    - The developers / testers (fears: false positives, breaks their coding habits, slowness, extra documentation work)
    - The person with the wallet (value of bugs removed vs. tool cost)
- Can you determine the tool's capabilities without investing a large amount of time into evaluation and separating the marketing message from actual technical capabilities?
- Does the sales process try to get you to commit before even trying?
    - "Evaluation licence" costs; cannot even evaluate before paying something; trying to leverage "sunk costs fallacy"
    - Non-Disclosure Agreements
    - Vendor salespeople trying to get one of your non-tech managers on the hook and bypass your technical people
- Can you run the tool in an environment where you can process confidential information (e.g., customer-owned source code) with it?
    - Does it want to "call home", or do the processing "in the cloud"? Is this acceptable to you?
    - Does the tool vendor learn something from your use of tool and its findings? Is that a problem?
- Do you need a support contract or buy training or consultancy to:
    - Set up the tool
    - Customize the tool
    - Maintain the tool (e.g., new rulesets)
    - Effectively use the tool (esp. if you are not a security person)
- How does the support contract or training add up to the Total Cost of Ownership?

- On what platform does the tool run? Is it compatible with your developers' platforms, your Continuous Integration / Test Automation platform, or anything else you have in-house?
- Is the licence per-seat, per-project, per-loc, floating, one-time, annual, or what?
    - Can you change the licence mid-flight if it turns out the uptake of the tool was not what you hoped?
    - If your business scales up a lot, do the tool costs rise with your revenue / customer base, or are they tied to something else?
    - If your business shrinks or gets refocused, do the tool costs go down with your new focus?

General considerations

- Is everyone's expectation about tool's capabilities on realistic level?
    - Does the tool provide any kind of metric against which its performance can be assessed later? Can we tell it was "worth it"?
    - Is the tool intended to bring in security knowledge ("canned knowledge"), enforce a policy or guideline, or help an expert? After determining this, is this what is really needed in your specific case?
- Is the tool geared towards compliance checks or security improvement, or both?
    - Compliance to security requirements does not necessarily mean security
    - Security improvement does not always leave evidence that can be used for compliance checks
- Does the tool help to find implementation *bugs* or logical *flaws*? If it finds bugs, does it only find *known vulnerabilities* or can it find something that nobody knows about yet?
    - Different tools target different things, but you should be clear as to what your most pressing problems are. If you develop your own code, you aren't that interested in scanning for known vulnerabilities in 3rd party code. If you mainly integrate a lot of 3rd party code, you might very well be more concerned about that than the simple "glue logic" you are adding.
- Does the tool fit the protocols and inputs you actually have in your (test) systems? I.e., for example with scanners and injectors, can you actually run them in an environment that gives a truthful picture of your system to the tool, or do you need to create "toy problems" for the tool?
- Many tools do prioritization & risk ranking of findings. Is that compatible with your organisation's, or is all that eye candy just useless to you? Does the tool force you into a specific risk management process?
    - Or would fancy, colourful pictures and graphs be something that are a selling point to you internally?
- If the tool has any kind of (web) GUI, did someone try to actually use it? Don't buy it before you've actually yourself tried it. The sales technical engineers are trained to use it, and usability can be actually horrible if you haven't used it before.