The secret of incorporating security into functional testing

Working remotely all year is fun sometimes, but it's times when I get down here to Atlanta and sit down with the guys and gals who build some of the tools you all use for software security that I really get to mess things up :)

Conversation today was around tools and use-cases for the tools in the stream of creating more secure software. My experience in this industry over the last several years has taught me that you have to fashion the tools to the use-case. Even if you give me a fantastic hammer I still won't be a great carpenter. This applies to development and software security easily... but it's a tough problem to solve.

My specific point of attack here is the QA community, more specifically the functional testers. We've been talking for years about functional testers and the fact that many of us believe you can leverage this community to up the game, even if a little bit, on software security. I don't have any delusions that functional testers will all of the sudden overnight blossom into security professionals... that's silly ... but perhaps we can have a reasonable discussion about empowering QA functional testers to do some of the traditional «low hanging fruit» testing that is a drag on the higher-skilled security professional.

Whether you agree with that or not, my proposal is that with the right tools it can be done. More than just the right tools, with tools that appropriately match the use-case of the functional tester... so I've started collecting a list of things functional testers would require to add in the security.

At the top of that list is the requirement to **not disrupt existing productivity**. This is reasonable - but how do you accomplish that if you're dropping in a totally new 'thing' for the functional testers to test with. You have a few options ... you can hide the 'new' functionality as a new knob or dial or switch into the existing interface - thus losing some of the impact - or you can try to convince them to modify their process as little as possible to accommodate. My brain and experience tells me the trade-off of losing some functionality for the sake of simplicity is worth it. Functional testers won't know how to use a security testing tool to the same extent as security professionals, they simply won't be able to tell whether depth first or breadth first makes more sense and in which case. They probably won't be able to tell whether the right session-state parameter or cookie is being used, but that's OK. Doing some of that for them to gain economy of scale in terms of more testers is a fair trade. I'll expand on this point as we talk through it internally :)

The next thing that always comes up from functional testers is the notion of ***time***. Rarely does a functional tester say to me «sure, I can take on a new workstream, I've got lots of time!» That's just nuts. So how to we incorporate security testing with minimal time additive? I think we have solutions for that already... in technology that allows you to offload work. When you're going to be doing security scanning using a tool like Nessus, for example, you can set it up on one workstation, but then push it to a «worker» machine which will run a worker thread which just performs work and reports back to you or the management console. This is already happening in our products and I see others are replicating this technology too in AppSec. It just makes sense....

The other thing that's high on the list added responsibility. This is sort of similar to the time notion

The secret of incorporating security into functional testing

above, but responsibility means that functional testers have to understand and «sign off on» defects they track. If you don't fully understand a security defect you can't make a decision on whether it's real of not, and whether you're going to log it as a 'real critical defect' or a 'false positive' or de-prioritize it to 'high severity' or something else entirely. Fundamentally functional testers can't be responsible for security defects validation - so where does that leave us? I think this is solve-able too ... we need good workstream, and automation. Having everything in a contiguous workflow is possible, it just has to hit the right people's tools in the right order for the right reason - it can be done... the challenge is that most QA orgs I've been to have slight subtleties in how they operate so a one-size-fits-all isn't possible.

So now you're left building a highly modular, scaled-down (not «dumbed down», hate that), non-disruptive tool for functional testers to do basic security testing. This isn't easy. I'm going to stop you right there if you're about to tell me to outsource the 'testing' completely to you as a 3rd party and just forget AppSec. That's simply stupid, to give up responsibility like that... no organization ever has «*tested themselves secure*» ... ever. Software security-as-a-service has tremendous value when you've got the process integrated with your development lifecycle and there's more to it than just testing. Then you're just checking a box, and you don't actually care about building low-risk applications... you're checking a box. Not that there's anything wrong with that... OK there is something seriously wrong with that from a security perspective, but it's not always a security decision (haha ...).

So what's the magic answer to getting functional testers, which outnumber security assets in a typical organization 20:1, on board with adding security testing into their workstream? All of the above discussed points... and stay tuned, there's a lot more to this discussion. I'm setting up some round-tables with QA managers all over the place - so if you know someone who runs a QA organization, or you ARE a QA manager, please contact me as I'd love to have you part of this discussion.

Thanks for reading, if you want to leave a comment also please leave your Twitter handle so I can attribute it to you! ...more on this very, very soon, and as it happens.

Cross-posted from Following the White Rabbit