**Security Testing**

What seems to be the consensus of most (and maybe all) software engineers that have pondered about security testing? Well, it’s hard. Of all the articles I had to choose from for this topic there seemed to be a running theme: Security testing is a difficult feat to accomplish correctly and adequately. And as it were, I landed on the one article I felt fit this premise, *Why Security Testing is Hard* by Herbert H. Thompson. In this short reflection I’ll summarize two important points that seem to be key in the author’s manifestation.

**Side-Effect Behavior**

According to Thompson, side-effect behavior is what happens when a test case only checks for a specific behavior to see if it produces what it is intended to for proper functionality. As a result, the software may also “perform some other action overtly, or in some cases more subtle which the testers might not detect” (Thompson 83). This is a classic example of what happens when a tester is not testing for defects but rather just proper functionality within their programs. Unfortunately, this is where most bugs come from. This side-effect is easily cloaked because for all intents and purposes to the tester, the application performed correctly. In an attempt to visualize this Thompson creates a picture of a circle with a squiggly line on top of the outer band creating an amoeba-like structure. The inside of the circle is green portraying proper functionality. Where the squiggly line overlaps the inside of the circle the color is blue showing “missing or incorrect functionality: most functions being here.” And outside the circle but inside the squiggle, it’s red; displaying “side-effect functionality: most security bugs are here” (Thompson 84). Concluding that security bugs are outside the circle and are *side-effects* of normal functionality.

**Dependency, Design, & Implementation Insecurities**

Depending upon libraries is more common today than ever before. And, as the software

being made is much bigger and is constantly being inflated to fit the new technology supporting it, it collects dependencies that might inherit security flaws unknown until properly put together. Also, designing new software can be difficult for untrained professionals and a company just wanting the best applications or newest innovations can easily overlook simple design flaws that create security issues. As Thompson describes, “Programmatic interfaces can sometimes bypass security controls creating gaping security holes” (Thompson 85). And lastly, long spans between check time and time-of-use leaves the door open for ‘man-in-the-middle attacks.’ Here someone can get in between the implementation process from testing to roll-out and cause havoc.

**Need For New Tools & Approaches**

Security and testing are obviously incredibly important tenets inside of the software management life-cycle and security testing is difficult. Unfortunately, most of time spent on them is when it is too late. Most companies don’t spend the time or money necessary on testing, while others spend an enormous amount of time and money on security insurance. After-the-fact thinking is not a sustainable future for any innovatively-savvy company. I believe the marriage of security and testing is the best approach to solve this dilemma. By combining the two and calling it security testing alone changes a projects scope. This will allocate resources from top-minded professionals from the offset and open the proper channels at the proper times to create real, permanent fixes. Engineers should make specially suited testing programs, design security testing before development begins, and implement stricter rules and regulations. Saving time and money by being proactive and not reactive can be a hard sell to most stake-holders; but at least you won’t have to advertise for user-trust or hope for massive insurance payouts.

**Reference(s)**

H.H. Thompson, “Why Security Testing Is Hard,” IEEE Security & Privacy, July/August 2003, pp 83-86.