“A Pseudoperipatetic Application Security Handbook For Virtuous Software”

Introduction

In the past 10 or 15 years an increased awareness of application security[1] (AS) in computing and information systems has resulted in many volumes of material (e.g., Cross 2006, Burnett 2004, Seacord 2005, Clarke 2009). Security conscious developers, testers, and organizations wishing to adopt “best practices” have a lot of work to distill these many volumes of advice and principles into easily implementable and understandable approaches. Following the off-hand suggestion from a colleague (Perkins 2010), I have taken her phrase “virtuous software” as a starting point. In this paper I comb through the *Nicomachean Ethics* [NE](Aristotle 1984) to find appropriate guidance for virtue in AS. As the author is a Canadian federal public servant, one intended audience are his fellow application developers, testers, etc. In this context there is an implicit “values and ethics” framework, tradition, etc. already understood. I will allude to these from time to time but will not discuss them in detail. Ideally, this document would eventually discuss them more concretely, but in the interest of brevity, I have largely omitted reference to the relevant legislation and guidelines. It is, however, vital to realize that at the time of writing developer guidelines for AS are a work very much in progress in my organization, the Canadian federal public service, and indeed in the industry generally.

The selection of NE was arbitrary - I have performed a similar exercise in draft form with Plato’s *Meno* (Plato 1989) and would hope at some later date to investigate

[1] AS is to be distinguished in discussions of computing security from infrastructure security, dealing with antimalware solutions, public key utilities, routing rules in networks, etc. 70% of current exploits and vulnerabilities are in application areas (Sykora 2010) and subsequently AS merits philosophical attention.

other classics of virtue ethics (likely the *Analects* and the *Tao Te Ching*, given that I have many Asian colleagues it might be interesting to do a cross-cultural study). There is, of course, substantial modern work (e.g., that reprinted in Slote and Crisp 1997) on virtue ethics as well (including in ethics of technology; see, e.g., Coleman 2001), but in the interest of manageability this paper restricts itself to NE alone.

Snippets

Nature of technology

Aristotle tells us at NE 1094a1 that all art (or technology, or craft[2]) aim at some good. This should be taken two ways. One is that we should aim to have a goal in mind for what we develop - a truism that extends from the whole to the parts and back again. A second way is that because software aims at a good we should not leave it open to be exploited for some evil.

---

[2] I distinguish technology from craft and science from technology. Technology is explicit action informed by scientific research. By contrast, craft is deliberate action with no (or almost no) scientific basis. For example, whittling a stick aimlessly is neither, but deliberately shaping one to a point because one's father did the same thing or because one "sees what to do" (etc.) is a craft. Applying principles of mechanics to design the optimal point makes the activity technological. (On the science-technology distinction, see, e.g., Douglas 2000.) A distinction between practice oriented activities, however, is not important for the present purposes: technê can be regarded as including at least craft and technology, as well as some "art" in one of its more archaic senses. It is also unclear to me if the craft-technology distinction is ethically important: on the one hand, technology in general would allow more harm (think, e.g., nuclear weapons) because of the greater understanding of reality involved; on the other hand, the double edged nature of the basic science involved entails that the same greater understanding could be used to assist in minimizing harm. See also Diamond 2005 for a patient, fair, corrective to the alas all-to-popular idea that craft is always more environmentally benign than technology.

At 1094b12ff we are reminded that demonstration is often (Aristotle says always, but we can be a bit more uncertain) impossible in ethics and not to ask for more precision than the subject allows. This is important for the software developers who are fond of rules, precise languages and exact specifications. A different mindset is important (though, cf. Turelli 2009 for an approach to software ethics which does lead to some form of demonstration[3].).

1100b15 is about how virtuous activities are more durable than knowledge. I am not sure if this is true in general, but it should be taken as advice on how to approach application security. In particular, since particular classes of exploits wax and wane with specific implementation languages and environments, to cultivate the proper mindset for AS we should instead cultivate better *developers* rather than *merely* read books and take courses. Aristotle would likely say you need knowledge to be virtuous but it is certainly (*pace* Plato) not sufficient.

1105a8 and following reminds us that virtue is concerned with what is harder. This is certainly true of AS. It is easier, at least from the narrow perspective of simply getting something into production, to forget about AS and just "get it working". But we are reminded in Spinoza somewhere that nothing worthwhile is ever easy (and I take it that AS *is* worthwhile).

One can obtain some characteristics of "virtuous" software at 1106a15-17. Aristotle tells us that virtues (here "excellences" reads a bit better) of something allow it to be done/used well. Three characteristics of the virtuous thing here mentioned are (1) it

---

[3] I will not venture a view on whether or not Turelli's specification language would allow demonstrations in ethics which would meet Aristotle's strictures concerning the demonstrable as that would take us very far off topic.

must be of good condition, not easily broken, (2) it allows well-use of its functions and (3) there is not some other use of its proper functions. I will take these in turn.

Take point (1): those involved in creating software are familiar with tools, implementations, etc. being “broken”. One way in which something can be broken is if it suffers from a variety crippling bugs. As it happens, these crippling bugs are often also AS vulnerabilities. For example, crashing when a user pastes too much text into a text field is a bug, and a symptom of broken software. It is also a possible location for an AS exploit in many environments and contexts (see, e.g., the ur-paper on the subject, Aleph1 1996).

On point (2), the same example (i.e., inserting too much text into a text edit control) applies. Users expect their applications to be stable, easy to use and to meet their “business needs”. They do not expect, and should not want, their applications to be vulnerable to drive by attacks and exploits. They do not want their servers and workstations to be bogged down by crippling malware or to be a source of spamvertising.

Point (3) continues the second point admirably in what one might call the “opposite direction”. I take it here that the lesson is something like using “right tool for the right job”. A well designed application has a well defined purpose for existence and use. Applications with less well-defined purposes are more easily exploitable and harder to maintain in general.

Nature of virtue

Aretê can be translated as both (or either of) “virtue” or “excellence”. Thus in a way this handbook is fulfilling the important ethical injunction given to Bill and Ted: “Be

excellent to each other" (Matheson and Solomon 1989). This extends to the users, testers, fellow developers, infrastructure maintainers (do you want to explain to the networking folks that you crowded out the chief statistician's emails due to the inadvertent spam-machine you allowed to be created?) and indeed to the community generally.

Aristotle asserts (1096a25) that there is more than one virtue. This should remind us that there are competing properties to build into our (computing and more narrowly application) systems. Security itself is (arguably) several different virtues in one (e.g., reliability, non-repudiability, non-disclosure, etc.), and is itself only one virtue amongst several of relevance to computing (others might include development time/ cost, performance, ease of use, etc.). However, Aristotle's view is contentious historically: there is much debate over the "unity" of the virtues. Since developers (etc.) think of these virtues as able to be traded off against each other, it would follow (if they are indeed virtues) that the unity thesis is wrong. On the other hand good *developers* (rather than their products) try to balance the relevant "subvirtues" and are aware that one area can affect the others.

Happiness is said to be the chief good (1097b1, 1097b20). We need not take a stand on this thesis here: however, how Aristotle argues for this viewpoint is via a "proper function" argument. Whatever the chief good of our software is, and whether or not you believe humans have (a) proper function(s), it seems much more plausible that technology, including software, does. This can be taken simply as what it is designed to do. Security vulnerabilities in software are ways in which the proper functions of something get subverted, or alternatively, the proper functions are imperfectly realized. Virtuous software is similar; at 1139a18 Aristotle returns to the concerns

over “proper function” and tells us they apply to things, generally, and not merely to a specific class of things - persons. (I assume that computer applications are things, or at least that Aristotle’s notion can generalize.) It is thus plausible to claim that specifications of a computational system are (some of the) ways to attempt to delimit and “well define” its proper function(s).

When talking about virtues, it is important to realize that I (and Aristotle) am not talking about merely moral traits in the narrow sense such as charity, justice, compassion, etc. but also about more intellectual ones (as we are reminded at 1103a5), such as intellectual honesty, diligence, thoroughness of testing etc. What exactly these traits should be shall be discussed in due course.

Virtues can have their good with respect to a state or an activity (1157b6). Software involves both: data structures hold state, and activities are algorithms (or perhaps, events in event-oriented programming paradigms) implemented as programs. Consequently both are important in AS. For example, managed strings in languages like Java or C# are virtuous in a security-conscious way[4] that the “raw” strings in C are not.

Finally, towards the end of NE (1179b1-2 ff.) Aristotle reiterates that one has to *use* virtue. Software developers should read books on AS, attend courses, maybe even

---

[4] The qualifier “security-conscious” way is somewhat important, as one could argue there are non-security virtues of software. These are outside our focus in the present work, but speed is clearly one of them, a trade-off made when one decides between managed strings (Java, etc.) and raw arrays with no bounds checking (C). My colleague’s “virtuous software” phrase thus does require some qualification and amendment, especially with speed vs. security (as in the string case) is a common (alas) perceived (?) trade off.

listen to this talk, but most of all they should get hands on experience in many aspects of the problems (design, implementation, test, etc).

How to obtain virtue

1103a20 tells us that virtue comes from habit. While Aristotle gives an argument from etymology to support this conclusion, we need not be speakers of Greek in order to adopt the conclusion. Thus, just as we developers (etc.) have learned to implement certain sorts of algorithms and the syntax and semantics of our languages by practicing and observing the skills of experts, so can we learn some of our virtuous software principles by practice. A lot of what we do (and even know) is *tacit*[5] (Polanyi 1974 [1958]) and thus cannot be learned or necessarily taught explicitly - it must be learned by example and by doing (1103a32). This is in part why I have encouraged my colleagues to follow better practices even where it doesn't matter as much - such as implementing a good password system (store hashes of the passwords, not the passwords!) even if it is just for an in-house online course system's grading and quiz module. Thus when it comes time to writing something where it really matters (in our workplace, to protect confidential data) they will have (partially) developed a habit to think in such terms.

Aristotle also says one can train in virtue by being in company of the good (1170a11). No doubt he would agree that this extends to things and to persons (as

---

[5] I am not committed to the degree of tacit knowledge defended by Polanyi nor necessarily the conclusions for science and technology he draws. See also Wimsatt (2007) for further takes on the tacit knowledge used in technology - explicitly mentioned is debugging, which is a notoriously difficult skill to learn. Much more needs to be said on this topic: perhaps, given its partially epistemic character it would do to discuss it in the context of virtue epistemology. But in any case that is for another time.

he talks about virtues of both) and subsequently one can train in AS both by good example code and by observing others make good AS decisions. How one identifies these when one has no prior knowledge is an instance of the paradox of the knower in Meno's form (Plato 1989), of course, and one we've confronted as we try to bootstrap our more AS-conscious development practices.

As a final way to obtain virtue, sometimes legislation is proposed to stimulate virtue (e.g., one proposed justification for tobacco taxes). Aristotle discusses this as well (1180a6 ff.). Arguably this is one way to interpret (at least in part) the ethics legislation which governs my workplace. However, it is for many simply a reminder. One has to internalize something like the values first or one is simply acting out of fear or what Aristotle would no doubt call an unvirtuous prudence.

What results

In addition to the merits of having better written software from the AS standpoint, at 1101b31 we are told that there are extrinsic benefits to virtue as well: we get to perform noble deeds and will acquire praise. Hence, I recommend to managers and compensation specialists to put in place something to reward good AS development and practice.

Aristotle also tells us of various outcomes that befall the vicious, in addition to discussing the benefits of the virtuous. In so doing he partially also handles an objection some have to "moralizing software development". (I have been told that by focusing on the ethics of software too much one crowds out the need to actually release something to one's clients, for example. I have also been told that the practice is simply overkill since exploits or vulnerabilities will not be seen by our users.) Aristotle explains that the results of the vicious need not always be disgrace

(1122b31). In this way we can recognize that sometimes milder sanction (if anything at all) is appropriate. For example, if I tell another developer his code ought to store cryptographic strength hashes of passwords and not the passwords themselves, I may very well sound critical, true, but I am not trying to invoke a punishment or the like, at least for the first infraction. (Of course, in my case I have no supervisory role anyway, but the point remains.)

Happiness is supposed to also be a consequence of virtue (1177a2). This is of course very contentious historically. Moreover, what is definitely true is that software developers generally hate finding and fixing bugs. Since software vulnerabilities arise as a result of bugs[6], one might think that Aristotle is wrong at least in our area of discussion (and hence generally). But at least in this area there is another way to look at it. An AS-aware development practice works in part to minimize bugs, so the *fix* stage which is so hated hopefully will be minimized too. However, to be fair, this is something of a “hill climbing” situation - it is nice to get into the next valley, but to get there requires effort and sometimes it is easier (or perceived to be such) to stay where one is. I have encountered resistance to my proposals, suggestions and ideas and expect more.

Who and What is Virtuous

At 1105a30, we are given characteristics of the virtuous agent. She must (1) have knowledge, (2) must choose acts for their own sakes, and (3) her actions must proceed from firm and unchangeable character. (1) and (2) can easily be adapted for systems development. (1) includes both the fragment of knowledge of the subject

[6] Bugs here include mistakes in the software implementation itself as well as poorly designed systems due to incomplete or otherwise imperfect specifications.

matter plus the knowledge of languages, tools, etc. which are used. as well as knowledge of some vulnerabilities, their remediations, etc. (2) tells the developer that she must not make coding, design, etc. decisions arbitrarily. This suggests proper use of change management systems like jira or the bug-tracking features of Team Foundation Server in order to document how decisions were made so as they can be seen to be for a good reason in the context of the system. (3) is a bit awkward. I would claim we need not adopt this part of Aristotle's ethics at all, except in one limited sense. I propose we change it to reflect the usual institutional ethics we are already subject to - we should be resistant to certain sorts of untowards outside forces and influences. Moreover, it is a reminder that there is a part of ethical systems generally which is "more solid" in the sense that it is independent of any particular developer[7] (etc).

What are specific virtues?

Courage is the first (1115a5 and following) Aristotle discusses. He tells us there are five sorts. One important sort for the security conscious computing professional is intellectual courage, particularly in an environment where the approaches, techniques and considerations are new. Also important can be a courage to confront authority: sometimes application security involves whistleblowing. These two might not exactly be what Aristotle had in mind; however they are nevertheless forms of courage vital when implementing, testing and sometimes even just using computing and information systems. One can, as I have, found security flaws by accident, and

[7] *Nota bene*: I am not claiming it exists independently of all developers (never mind all people), nor taking a firm stand on any matter of social or ethical ontology: whatever one might think of such things, there is at least a "legal fact" about the existence of such legislation and also the history of the Canadian (and common law generally) legal system and its relation to the public service, etc.

so courage enters when it comes to patiently explaining what one has discovered to the relevant designers, implementers and maintainers.

Aristotle next discusses temperance (1117b24 ff.). Although I am not sure that this virtue has any use in application security, it does not follow that it is not worth discussing herein - for three reasons. First, it is not clear that all virtues need apply to all activities, and second it isn't obvious how to construct a list of virtues - notoriously there is disagreement on what they are. This is all the more clear once Aristotle mentions pride at (1123b33 ff.). In some contexts at least, Christians regard pride as a sin - and hence not a virtue. But here it is on Aristotle's list. It seems to me the resolution is as follows: at least qua software developer, tester, etc. it might be possible to have a canonical list of virtues. As yet nobody in my organization has tried to write one up. (A future project for me?) But in a pluralistic society it is difficult to generalize further. This is not a concession (either way) to ethical relativism other than in the mere sociological fact of ethical disagreement. Third and finally, one would have to hold a rather unusual viewpoint to hold that all of the NE would be relevant to a topic Aristotle could not have even conceived; it does not follow from this that none of it is relevant and so a mention of temperance reminds us of this (admittedly banal) point.

There are many virtues, including nameless ones (1126b20, 1127a13). This is important too: we might find that AS requires recognizing character traits and behaviours that have well defined earmarks but don't have names. One of them that I think is vital for AS (and software development generally) is "thinking outside the box". This is not mere creativity, but sort of an "off-the-wall" creativity, as the ability to think like an exploiter of software is truly different from the mentality used in

traditional development, etc. Reading of many exploits one often wonders how they were imagined[8]; in order to protect against them and, more so, to protect against the as yet undreamed of attacks, requires a different mindset than has been traditionally inculcated in software professionals.

Justice is a virtue (1129a1), and Aristotle tells us that justice encourages other virtues. However, in order to implement this suggestion in an AS context, we need a better understanding of justice, a certainly well-disputed notion. For example, sometimes the law is regarded as unjust (as in the traditions of civil disobedience) but Aristotle tells us that the just is the lawful (1129b1). On the other hand, he also says that the just is the proportional (1131b19 ff.). Some classes of vulnerabilities involve “disproportion”, after a fashion: for example, a denial of service vulnerability involves excessive consumption of a resource (bandwidth, CPU time, storage space, etc.); a buffer overrun is a disproportion between memory usages.

Conclusion

We have seen how it is possible to discuss many areas of application security in the framework of one particular work of virtue ethics. Future work includes analyzing similar material, particularly from other virtue traditions, seeing what can be adopted from all of them and providing guidelines for how to adopt the relevant character traits for developers, testers, software “architects”, etc.

References

Aleph1. 1996. “Smashing the stack for fun and profit”, Phrack 49.

[8] The SQL injection attacks in discussed Clarke 2009 which use T-SQL’s WAITFOR statement are particularly ingenious in my view: they show how not providing error messages to the user (on a web page, say) can only slow down the determined exploiter, not make it impossible.

Aristotle. 1984. “Nicomachean Ethics”. In *The Complete Works of Aristotle*, vol. 2 (ed. Jonathan Barnes). Princeton: Princeton University Press.

Burnett, Mark. 2004. *Hacking the Code: ASP.NET Web Application Security*. Burlington: Syngress.

Clarke, Justin. 2009. *SQL Injection Attacks and Defense*. Burlington: Syngress.

Coleman, Kari. 2001. “Android arete: Toward a virtue ethic for computational agents”. Ethics and Information Technology, vol. 3 no. 4.

Crisp, Roger and Michael Slote (eds.), 1997. *Virtue Ethics*. Oxford: Oxford University Press.

Cross, Michael. 2006. *Developer’s Guide to Web Application Security*. Burlington: Syngress.

Diamond, Jared. 2005. *Collapse: How Societies Choose to Fail or Succeed*. New York: Viking.

Douglas, Keith. 2000. “The Scientist / Technologist Distinction (or, who gets to play with toys?)” Unpublished paper, UBC course PHIL 536A Winter 2000 semester.

Matheson, Chris and Solomon, Ed. 1989. *Bill and Ted’s Excellent Adventure*. Hollywood: MGM.

Plato. 1989. “Meno”. In *The Collected Dialogues of Plato: Including the Letters*. (ed. Edith Hamilton, Huntington Cairns). Princeton: Princeton University Press.

Perkins, Evelyn. 2010. Unpublished comment, meeting of the Secure Coding Practices Working Group, Statistics Canada.

Polanyi, Michael. 1974 (1958). *Personal Knowledge: Towards a Post-Critical Philosophy*. Chicago: University Of Chicago Press

Seacord, Robert. 2005. *Secure Coding in C and C++*. New York: Addison-Wesley Professional.

Sykora, Boleslav. 2010. Lecture material, Learning Tree International Course 940.

Turelli, Matteo. 2009. "Translating Ethical Requirements into Software Specifications". Goldberg Graduate Award Presentation at NA-CAP@IU 2009.

Wimsatt, William. 2007. *Re-Engineering Philosophy for Limited Beings: Piecewise Approximations to Reality*. Cambridge: Harvard University Press