Why Offensive Security Needs Engineering Textbooks

or, how to avoid a replay of "crypto wars" in security research

Sergey Bratus, Iván Arce, Michael E. Locasto, Stefano Zanero

August 22, 2014

Offensive security—or, in plain English, the practice of exploitation—has greatly enhanced our understanding of what it means for computers to be trustworthy. Having grown from hacker conventions that fit into a single room into a distinct engineering discipline in all but the name, offensive computing has so far been content with a jargon and an informal "hacker curriculum". Now that it is unmistakably an industry, and an engineering specialization, it faces the challenge of defining itself as one, in a language that is understood beyond its own confines—most importantly, by makers of law and policy.

Currently, lawmakers and policy makers have no choice but to operate with pieces of our professional jargon that got publicized by journalists. But writing laws based on professional jargon is dangerous: it will be misunderstood by lawmakers and judges alike. It's not the wisdom of the judge or the legislator that is in question, it's their ability to guess the course of a discipline years in advance.

Consider the concept of *unauthorized access* at the heart of (and criminalized by) the Computer Fraud and Abuse Act (CFAA). The un-anticipated, "unauthorized" uses of today will be primary uses or business models of tomorrow. When CFAA was written, connecting to a computer on which one had no account was pointless. Cold-calling a server could serve no legitimate purpose, as none were meant for random members of the public; each computer had its relatively small and well-defined set of authorized users. Then the World Web Web happened, and connecting to computers without any kind of prior authorization became not just the norm but also the foundation of all related business. Yet the law stands as written then, and now produces conundrums such as whether portscans, screen-scraping, or URL crafting are illegal, or even whether telling journalists of a successful URL-crafting trick that revealed their email addresses could be a felony (as in the recent US v. Auernheimer case). Even accessing your own data on a web portal in a manner unforeseen by the portal operator, as in case of ApplyYourself users who could see their admission status prematurely may similarly be a crime under CFAA (for discussion of these cases and different institutions' reactions to them see S.W. Smith, *"Pretending that Systems are*

Secure", IEEE Security and Privacy, 3 (6): 73-76, November/December 2005).

Lawmaking with regard to offensive security artifacts has already started. Article 6 of the Budapest Convention on Cybercrime requires signatories to issue laws that criminalize *production, sale, procurement for use, import, distribution or otherwise making available of [..] a device, including a computer program, designed or adapted primarily for the purpose of committing any of the offences* it established as criminal; Germany and UK have since enacted laws targeting so-called "hacking tools". Although, to the best of our knowledge, no prosecution of security researchers has yet taken place under these laws, they have had non-trivial chilling effects. More recently, *intrusion software* has been categorized by the December 2013 Wassenaar Arrangement as dual use technology subject to exports control; such software is defined as capable of *extraction of data or information, from a computer or network capable device, or the modification of system or user data* or *modification of the standard execution path of a program or process in order to allow the execution of externally provided instruction.* This is, of course, what debuggers, and hypervisors do, let alone all varieties of JTAGs; although the document futher stipulates that *"Intrusion software" does not include any [..] hypervisors, debuggers or Software Reverse Engineering (SRE) tools [..]*, the above functional description fits them perfectly.

Such language demonstrates the challenge we face. As native speakers of the jargon, we understand that an exploit, a rootkit, and a defensive module that inserts itself into a piece of software are likely to use the same technique of reliably composing their own code with the target's; however, lawmakers do not see their unity.

Will jailbreaking or composition beyond well-defined APIs such as DLL injection survive these challenges? Many sufficiently advanced techniques in both defense and exploitation perform some of a debugger's or linker's tasks without being either debuggers or linkers; new debugging and dynamic linking techniques often look like exploitation. For example, BlackIce Defender, the first Windows firewall, linked itself into the kernel by "modifying the standard execution path" to defend the system, and even patented the technique that many rootkits rediscovered since; Robert Graham tells the story in "The Debate over Evil Code."[1] "Bring Your Own Linker" has long been a composition pattern for both offense and defense.[2]

Proposals for stricter regulation of exploits are not hard to come by. A good example is provided by Stockton and Golabek-Goldman[3], which makes an aggressive and ill-informed call for regulation (and spells ∅day with a symbol for "empty set"). It defines "weaponized" on its first page to mean "disrupt, disable, or destroy computer networks and their components" and then on the next page claims that *"Criminals buy and use weaponized ∅day exploits to steal passwords, intellectual property, and other data [..]"*, even though disabling or destroying a compromised computer in order to steal passwords or secrets is

[1] `http://blog.erratasec.com/2013/03/the-debate-over-evil-code.html`

[2] Bratus et. al, "Composition Patterns of Hacking"

[3] Paul N. Stockton and Michele Golabek-Goldman, *Curbing the Market for Cyber Weapons*, Yale Law & Policy Review, Dec. 2013

counter-productive; in fact, it would be just plain stupid, as it would alert the victim of the breach and likely eliminate the value of stolen password or data. Apparent lack of familiarity with the field, however, doesn't stop the authors from calling for prosecution of security researchers under the CFAA, a law so broad and vague that prominent legal scholars argue it should be void for vagueness (Orin S. Kerr, *Vagueness Challenges to the Computer Fraud and Abuse Act*, Minnesota Law Review, 2010).

If anything, we can expect more laws and regulations on the basic artifacts of our profession. The only way for us to avoid overly broad formulations that would snare every technique we use is to develop a language that puts offensive computing in perspective of other computer engineering.

In short, we need textbooks and textbook definitions that describe offensive computing—so that policy makers need neither puzzle over jargon nor have to design their own language, both approaches being potentially disastrous to the future state of practical computer security.

1 Why offensive computing matters for security in general

> If you shame attack research, you misjudge its contribution. Offense and defense aren't peers. Defense is offense's child. —John Lambert[4]

Exploitation is programming. It is the kind of programming that every programmer should if not directly practice then at least understand the capabilities and limits of—because it will be practiced on his code. Our security is only as good as our understanding of this kind of programming, because it's essential nature of general-purpose systems (or perhaps of all rich enough computing systems) to allow a myriad other executions than merely the intended ones. Until all possible latent, unintended execution models are understood, they can neither be eliminated nor triaged.

Security and trustworthiness of code means attacker's inability to program it. In computer science theory, we emphasize results that show what can and cannot be programmed; in fact, our very notions of computer architectures derive from these results. Programmers and designers of a trusted system must be equally focused on what can or cannot be programmed on (or, against) their code—no less than a theorist is concerned with what can or cannot be computed by particular execution models, type systems, automatic theorem provers, verifiers, and the like.

The strongest kind of trust in systems security, just as in cryptography, derives from some programs provably not existing—or at least from their existence being highly unlikely. Ciphers are only trusted because no efficient algorithms

[4] `https://twitter.com/JohnLaTwC/status/442760491111178240`

to solve certain algebraic problems are believed to exist. Cryptographic protocols are only deemed trustworthy when no sequence of attacker manipulations of their messages can interfere with their transactions, and so on.

To stress the role of anticipating and precluding attacker's programs in the realm of cryptographic protocols, Anderson and Needham call protocol designer's task *programming Satan's computer*:

> "In effect, [protocol designer's] task is to program a computer which gives answers which are subtly and maliciously wrong at the most inconvenient possible moment. [..] we hope that the lessons learned from programming Satan's computer may be helpful in tackling the more common problem of programming Murphy's"[5]

For applied systems tasks the primitives of adversarial programming may be different, but the essence of trustworthiness is the same: such attacker programming must fail, preferrably due to provable impossibility of certain tasks.

We can trust any system only so far as we understand its unintended programming models (so-called "weird machines"[6], building on prior work by many others, such as Gera's *About Exploits Writing* [7]) and their limits. Exploits are merely artifacts and expressions of this understanding; the essence of the discipline is the skill to discover, validate, and generalize such models. Yet no research activity can develop without free exchange of its artifacts, and the discipline of systems security needs to develop a lot further before we can trust it even to the same extent as we trust analysis of cryptographic protocols.

Exploits are the primary tools in exploring the unexpected, latent models of programming that are inherent to the ways we currently build computing systems. Thus we must be able to speak about them in all their unity and differences, and to be understood.

2 Exploits: research or development? Proof-of-concept or "weaponized"?

Compared with software engineering, arguably its most closely related field, security focuses much less on its engineering process. Unlike software engineering, which continually invents new processes and methodologies, and has an industry-wide shared vocabulary for the outcomes of different process stages (such as "design", "architecture", "prototype", "alpha-", "beta-", and "production" quality, etc.), security industry does not appear concerned with defining its process or its product through the stages of its development and maturity.

Terms occasionally used to qualify important industry artifacts such as *exploits* do not appear to have consensus definitions; perhaps the best example is

[5] `www.cl.cam.ac.uk/~rja14/Papers/satan.pdf`

[6] `http://langsec.org/papers/Bratus.pdf`

[7] `http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=About_Exploits_Writing`

the use of "weaponized"[8] to refer to a certain grade of readiness or effectiveness (or ease-of-use?) that must inspire awe in the prospective buyer (note also how such use in turn affects misuse in policy proposals, as quoted above).

Even terms purely technical in origin raise questions regarding their usefulness, e.g., the use of "memory corruption" in advisories.[9] Even the typically used term *remote code execution* is somewhat ambiguous, since it obscures whether introduction of external code by remote party is necessary, or full control is achievable by manipulating the platform's existing code, with remotely crafted data inputs acting as the de-facto exploit program.

It gets worse when we get to characterizing intentions of a particular research or engineering activity. Suppose some lawmakers would like to protect security research results while attempting to curb what they see as software developed with ill intent. Yet our industry's language lacks the ability to clearly distinguish research results from engineering artifacts. An in-depth technical description of a software vulnerability may or may not be equivalent to an actual exploit program that leverages said vulnerability. How much detail and analysis do you need to consider the two equivalent? Is it possible to regulate one but not the other? And if so, to regulate what exactly?

Even though there is a lot of architecting, programming, and testing involved in producing what could be called "commercial grade exploit"—all activities that can be more closely associated with software engineering than with research as such—this nuance seems to be lost on much of the security industry, and certainly on the outside world, which speaks of "vulnerabilities", "PoCs", "triggers", "payloads" and "weaponized exploits" as if they were interchangeable. Given such usage, the difference between an open source research tool, and a commercially backed software product that includes exploits is too nuanced to explain (see, e.g., Iván Arce's RSA 2005 presentation[10] on the subject.)

All the more so, a "textbook" gradation of exploits with respect to their power and reliability is necessary. As direct consequence of such a gradation, an evaluation of effort necessary to elevate privilege from any given exploit achievement becomes desirable. In other words, it is not enough for a customer of an engineering effort to know that a product or design is flawed; one might want to know how deeply the rabbit hole goes.

In plain English, what does it mean for software to withstand a particular kind of adversarial audit or testing? Once a vulnerability has been found, how general is its description as presented in an advisory or an exploit? Does the description need to capture an entire class of related vulnerabilities or merely a particular instance of an exploitable bug? How far should an exploitable bug be pursued by the researcher beyond the creation of code that exploits a particular platform or platforms? How resilient is the exploit against defenses such as address space randomization, non-executable memory, various canaries and other memory integrity checks? How resilient can it become after a man-

[8] http://blog.coresecurity.com/2009/11/05/speaking-the-language-of-it-security/

[9] http://www.riskbasedsecurity.com/2013/08/memory-corruption-and-why-we-dislike-that-term/

[10] http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=rsa2005_quality_of_exploit_code

month of engineering effort by the exploit developer, and how qualified should this developer be to pull it off?

For all of these, there appear to be neither accepted answers, nor a common language to provide then. Our industry still lacks a consensus vocabulary to describe the generality of knowledge about a flaw as encapsulated in an exploit or an advisory. For example, has the primary effort been spent on the discovery of the flaw or on constructing the exploit machine? How likely is the flaw to be present and/or exploitable in other instances of related codebases? Is the exploitability of the flaw an (un)happy accident, or does it reveal a general principle applicable even beyond related codebases?

Most of these answers become clear to experts after a careful study of the exploit, but no textbook or other authoritative publication captures them, and so makes it hard to explain the insights and the impact. Not surprisingly, it is a often a hard task to explain the impact of an "attack paper" to academics not versed in exploitation, as they too lack the terms for different degrees of impact and generality, and have no referent in industry language.

In short, a "Rainbow Series" for offensive computing suddenly sounds like a good idea.

3 Common Criteria or FIPS for offensive computing?

Contrast the lack of terms to describe the generality, the resiliency, or the reliability of an exploit with the well-known criteria for government procurement of trusted computing systems, such as the Common Criteria or the FIPS certifications. Their different levels enumerate processes and methodologies applied in development of the software, with those at higher levels expected to provide relatively stronger assurance. A ranking, however imperfect, of software construction and testing methodologies is implied, with respect to their relative power to provide assurance and verification.

A similar ranking of attack and assessment methodologies may be possible, with respect to their power of revealing flaws. The similarity would, of course, extend to the cautions and provisos that apply to software construction methods, namely, that their ranking is relative rather than absolute, and provides evidence of effort invested rather than proof of security in any given sense.

However, no such ranking is enshrined to date in a form available to industry outsiders. Some policymakers may understand that certain grades and levels of offensive skills, activities, and artifacts are indispensable to security education of every computer professional. They may understand that major advances in computer security have been made by "Citizen Science" of hacking and only then adopted by industry or academia, and that curbing this citizen science by turning the respective activities into legal minefields will shrink the talent pool of "cyberdefenders". Yet even so they lack the concepts and terms to clearly distinguish activities they want regulated from the basic tools of the discipline.

Moreover, perhaps their very ideas of what they want regulated will be changed once a proper language that shows the relative importance of offensive activities is available.

4 Have we learned the lesson of the "crypto wars"?

The 1990s were a formative decade for the commercial Internet in the US. Unfortunately, during this same time the US government policy was to treat strong encryption as a threat and to control implementations of certain cryptographic algorithms as munitions, subject to vigorous enforcement of export regulations. In 1993 the author of the original PGP software, Phil Zimmerman became the target of an FBI investigation for munitions export without a license, which lasted till 1996. At the same time a series of failed technological "solutions" and mandates, such as the backdoored-by-design Clipper chip[11] and third-party key escrow were promoted as a legally safe way for telecommunications industry to implement compliant encryption—which would have essentially amounted to pretend security.

Export restrictions on artifacts of cryptography have doubtlessly harmed its practical progress. Not only Johnny Q. Public still can't encrypt[12], but John the Special Agent can't encrypt either![13] No matter where one stands on whether and how much the latter should be allowed to wiretap the former, John certainly has things to hide and in fact a duty to hide them—in which he is conspicuously failing.

Could it be that *both* of these failures are due to the fact that deployment of strong crypto was stymied just when today's dominant communication protocols and infrastructure were rapidly developing? The fact is, they ended up leaving crypto behind, and matured without incorporating cryptography at their core. Superiors of John the Special Agent may have had visions of him using separate, special technologies vastly stronger than Johnny Q. Public's and obtained from sources untainted by the weaknesses of public commodity communications; it appears this vision was wishful thinking.

If having to pretend that poor cryptography was secure because practically exploring stronger crypto was a legal minefield led us to this point, where would pretending that computers are secure because of a likely minefield arising in exploitation engineering lead us from here? It will likely be worse, because the field of cryptography by 1990s already had mature mathematical theory not easily undercut by the drag created on its engineering practice. Systems security, on the other hand, is only building up its theoretical foundations, and is in need of much more feedback and generalization of its practice.

[11] M. Blaze. "Protocol Failure in the Escrowed Encryption Standard." Proceedings of Second ACM Conference on Computer and Communications Security, Fairfax, VA, November 1994

[12] `www.usenix.org/events/sec99/full_papers/whitten/whitten.pdf`

[13] `http://www.usenix.org/event/sec11/tech/full_papers/Clark.pdf`

If the practice of exploring the programming of programs' faults becomes subject to regulation as vigorous as the 1990s "Crypto Wars", will this practice develop enough to warn us before unsecurable designs come to dominate in critical infrastructure, power management, medicine, or even household appliances beyond any hope of replacement? Will we be surrounded by an Internet of Untrustworthy Things just as we are surrounded today by an Internet of Things that Can't Keep a Secret (or at least are no help to an ordinary person for doing so)?

Offensive computing—by now a research and engineering discipline that cuts across many technologies and abstraction layers—is central to security and trustworthiness of computer systems. However, the further one stands from security research, the less prominent the role of offensive computing appears. Even in the eyes of traditionally trained computer scientists and engineers this role looks somewhat peripheral; in the view of policy makers offensive computing is often completely marginalized and confused with criminality and ill intent.

These diverging views of offensive computing are a clear and present danger to the development of the discipline, and thus to our hope for improving trustworthiness of everyday computing. Without a concerted effort to claim its place, offensive computing will end up being further marginalized, nearly impossible to practice outside of costly legal protection, and completely impossible to practice as a citizens' science.

In order to protect our discipline, we need to make sure that good approachable textbooks or at least comprehensive dictionaries exist for it, that put it in proper perspective not only to experts but to much broader audience. Distracting as the task of writing them may be, failure to communicate the importance of offensive research will be a lot more damaging in the long run, to all of us and to the society that our research ultimately serves to protect.