What hackers learn that the rest of us don't

Sergey Bratus

May 17, 2008

1 Hacker community's growing impact

The hacker community has developed a set of approaches to computer technologies, in particular, to analysis, reverse engineering, testing and modifying software and hardware, that considerably differ from those of both IT industry and traditional academia[1]. Over the last few years we have seen the impact of the hacker culture grow significantly: exploits that used to be disclosed only on mailing lists and in "underground" magazines are now published in books[2], while publishers like Syngress and No Starch Press produced entire tracks of "hacker" books differing in style and substance from the accepted formats (and some others jumped on the bandwagon, offering a slew of books with "Hacker" or "Hacking" in the titles, some of which actual hackers sneer at); classic hacker tools made their way into academic curricula and industrial training[3]; consulting services of companies established by hackers are highly sought after.

Some idea about the value contributed by the hacker community to IT security industry can be obtained from the commercial success (and admissions prices) of the BlackHat and similar conferences where hackers present their results to the industry. In academia, a number of researchers and institutions recognized this value as well (see, e.g., Gregory Conti, "Why Computer Scientists Should Attend Hacker Conferences" [4]; also, consider the fact that

[1] A number of groups in academia shares and has influenced elements of the hacker culture. However, these are exception rather than the rule.

[2] e.g., "The Shellcoder's Handbook" by Koziol at al.

[3] e.g., the SANS Institute courses, `http://sans.org`

[4] `CACM,June2006,http://www.rumint.org/gregconti/publications/20050301_CACM_HackingConferences_Conti.pdf`

the U.S. Naval Postgraduate School sent its team to the "Capture the Flag" tournament at Defcon, and won it in 2004[5]). More significantly, important features have made their way into mainstream software after being designed, implemented, and tested in the hacker community. Examples include canary-based stack overflow protection (StackGuard) and executable memory page protection through x86 segmentation (OpenWall, PaX).

1.1 Concerning the word "hacker"

When first used to describe a group of people interacting with computers, the word "hacker" had strong laudatory connotations of deep knowledge driven by insatiable curiosity[6]. Unfortunately, its original meaning became diluted and perverted through decades of media misuse. These days, one needs to be wary when speaking about hackers to unfamiliar audiences – some listeners might assume one is referring to online extortionists or credit card thieves and suchlike. Such usage, alas, persists despite numerous attempts by those in the know to point out the wrong uses of the word.

In this paper, instead of adding another explanation of what we *do not mean* when we say "hacker", we invite the reader to contemplate the following four hypothetical headlines:

1. Locksmith burgles bank's safe.
2. Policeman shoots neighbor.
3. Doctor poisons co-worker.
4. Hacker steals private information.

We treat all three as examples of the same general situation: **"someone with special training and tools misuses them"**. Note, however, that (1)–(3) hardly make us fear and distrust locksmiths, doctors, or the police in general despite their obviously higher capabilities for causing certain kinds of harm.

[5] `http://searchsecurity.techtarget.com/originalContent/0,289142,sid14\_gci1000503,00.html`

[6] See, for example, "The New Hacker's Dictionary", `http://www.ccil.org/jargon/`, or Steven Levy's book "Hackers".

1.2 White hats and gray hats vs black hats

An important note is in order: under the "hacker community" we primarily mean the so-called "white hat" and "gray hat" communities. Under "white hats" we mean hackers ethically opposed to abuse of computer systems, and under "gray hats" – those who may run afoul of existing laws[7], but are motivated to warn the vulnerable and minimize damage. "Black hats" act for personal gain and without regard for possible damage. For further discussion of these terms and their different uses we refer the reader to Wikipedia's article on *hacker*.

White and gray hats publish their research in security–related public venues and e–zines; we note that publishing such materials has made a significant contribution to the improvement of consumer and business computer environments, and has been historically opposed by the "black hats", whose efficiency and ease of operation is significantly reduced by these publications — *Praemonitus praemunitus*. We also include in this definition a number of industrial and academic computer security research groups that are aware of the hacker culture, recognize the value of its contributions, and use elements of the "hacker" approach in their work.

Hacker knowledge and methods are no longer limited to the select few, and the hacker culture will undoubtedly continue to attract more participants, including students and developers. Therefore the leaders of industry and academia need to acquire a better understanding of that culture, and be aware of its values, and its unique strengths and weaknesses, whether they would like to benefit from the contributions of hackers or defend themselves from the malicious "bad apples" who reject the hacker ethics.

We are going to examine the differences that distinguish the hacker experience from that of most of the traditionally trained programmers, and show how they contribute to the overall improvement of the state of the art in practical computer security.

[7]Many kinds of unauthorized computer uses are harmful and ill-advised. At best they are public nuisances, and at worst should constitute crimes. However, lawmaking is not immune to the influence of vested interests or to ill-informed political agendas. With the advent of laws such as the DMCA and its further-reaching state counterparts, and in the face of initiatives to ban broadly defined "hacker tools", we should remember that even well-known academic researchers have been subjected to threats of criminal prosecution. Unfortunately, laws are not exactly made in heaven, but are enthusiastically interpreted in hell.

2 The "hacker methodology"

Before trying to elucidate the essentials of hacker modus operandi, let us summarize the trends in the industry and academia that are, on the one hand, in direct conflict with it, and, on the other hand, create the wealth of weaknesses and vulnerabilities that provides the hackers of all hat colors with a rich ecology to exploit.

The economics of insecure software and hardware has been widely discussed before. Attempting a brief summary of those observations we note that the typical developer is likely to experience much of the following.

- Developers are under pressure to follow standard solutions[8], "the path of least resistance" to "just making it work"; as long as "it works", detailed understanding is often considered optional.
- As a result, they may not realize the effects of deviating from the above, intended or unintended.
- Developers tend to be implicitly trained away from exploring the underlying API, because the extra time investment rarely pays.
- They are often offered a limited view of the API, with few or hardly any details about its implementation.
- They are de-facto trained to ignore or avoid infrequent border cases, and may not understand their effect.
- Developers may be explicitly directed to ignore specific problems, as being the domain of other developers[9].
- Developers must often comply with lack of tools for examining the state of the system, let alone changing it outside of the API.

In a typical academic setting, similar pressures exists in the area of curriculum development. The growing number of topics puts considerable limitations on student time that can be allocated for any specific one. As a

[8] We note that in some quarters what used to be called a program is now called a *solution*. Nomen omen?

[9] In private communication, a major vendor has been quoted to me as advising the customers that security of their product was the customers' responsibility. The customers were expected to "run it behind a firewall".

result, instructors carefully plan their teaching environments to minimize the probability that the student will be distracted from the task seen as the purpose of the exercise, such as by encountering a complicated border case. For example, it is common practice to create "wrapper" libraries that isolate the students from the unwanted complexity. Also, in OS courses the likely time cost of interacting with real hardware is offset by using software emulations (in operating systems courses the emulator is often that of simplified imaginary hardware).

Often this leads to unrealistic teaching environments that impart very little of the real world's actual complexity, creating false expectations in students and causing problems when they join the ranks of industrial developers[10].

Even if this danger of oversimplification is avoided, the students are still implicitly trained to follow the prescribed patterns without exploration (again, the necessary time investment does not pay) or understanding of the effects of deviating from them. Some topics, perceived as too complicated to explain, simply fall by the wayside (e.g., for OS courses: linking and loading, binary file formats and OS support mechanisms for debugging and tracing, as we illustrate below) and are characteristically repeated in books that deal with computer security, despite clearly belonging elsewhere in the curriculum.

Frustration created by these trends is one of the driving forces behind the hacker culture, which eschews the "path of least resistance" and concentrates on fully understanding the underlying standards and systems, complete with their border cases and vendor implementation differences.

In particular, we can distinguish the following tendencies.

- Hackers tends to treat special and border cases of standards as essential, and invest significant amounts of time into reading the appropriate documentation (which is not a good survival skill for most industrial or curricular tasks).

- Hackers insist on understanding the implementation of the underlying API and exploring it to confirm the claims of documentation.

[10]I once came across a CS introductory sequence that heavily stressed the use of a particular integrated development environment together with an input-output library designed to hide most of the standard system interaction and I/O complexity. Students who had little independent programming experience prior to this sequence, described their first internships as truly harrowing.

- Hackers second-guess, as a matter of course, the implementer's logic (this is one of the reasons for preferring developer-addressed RFC to other forms of documentation).

- Hackers reflect on and explore the effects of deviating from the path of standard tutorials.

- Hackers insist on tools for examining the full state of the system across interface layers, and for modifying these states bypassing the standard development API. If these are lacking, developing them is seen as a top priority.

These tendencies largely define the ways in which the hackers learn and work, and have produced an impressive array of tools, frameworks and exploits.

For example, the overwhelming majority of programmers have to deal with linking (and, every once in a while, with obscure linking errors), and every Linux UNIX distribution nowadays relies on dynamic linking. Yet the linking mechanisms and the corresponding parts of the binary file formats are hardly covered in the standard CS curriculum, and just about the only available book that goes into sufficient depth to cover this topic is M. Levine's "Linkers and Loaders". Programmers learn to interpret and fix the errors, as well as to avoid situations that create them, but they usually remain in the dark about the actual mechanisms that cause them, whereas hacker publications explain these mechanisms[11] in much technical detail, and provide tools for examining and manipulating them, such as ELFsh[12].

It is worth noting that although many aspects of the programmers' daily activity are directly affected by the design of the binary file format "insides", the knowledge of these is considered somewhat esoteric. Clearly, hackers who studied this have an advantage over the typical traditionally trained programmers.

C++ offers another example. Countless Object Oriented programming books explain the concepts of overloading and inheritance, both in abstract terms and on specific examples, using a variety of pedagogical techniques. Nevertheless, students find themselves at a loss often enough when asked to predict the outcome of mixing overloaded and virtual functions, let alone the

[11] E.g., a number of articles in Phrack 51, 54, 56, 59, 61

[12] `http://elfsh.asgardlabs.org/`

effects of multiple inheritance with both virtual and non-virtual functions present. Indeed, a whole culture of job interview puzzles has sprung up around such "trick questions". A hacker interested in the topic would likely start with the implementation of these mechanisms (name mangling used by compilers and linkers, and *vtables*[13]), after which the answers become clear, if not trivial.

The interest in internal workings of various programming language mechanisms is characteristic of the hacker approach. To the best of my knowledge, a hacker is likely to learn about calling conventions and stack layouts, exception handling mechanisms such as stack unwinding and setjmp/longjmp, and the basics of syscall implementations much earlier than the average student, and often they do so right in beginning of their own programming career. This gives them a different "set of tricks" that their peers who follow a more traditional curriculum are not even aware of.

Another example of a tool ubiquitously used but rarely fully understood is the debugger. Almost all programmers have used one, yet understanding of the underlying operating system and hardware features is relatively rare, and the number of available books covering the subject is in low digits[14] Yet a wealth of information on the subject was available to hackers for a long time, e.g., on the legendary Fravia's reverse engineering site; nowadays, the best resource to find out about the finer points of debuggers is the OpenRCE site[15]. Hackers had a clear advantage in this area – in fact, the industry has eventually learned from them and borrowed many anti-debugging and so-called content protection tricks developed by the hacker community[16]

3 The "hacker" curriculum

As we pointed out above, a number of topics in the education of a typical hacker is either missing from the standard Computer Science curriculum, or is presented in a radically different fashion. In particular, tools for injecting arbitrary data (usually prepared and formatted by several layers of APIs) into

[13]Described, e.g., in Phrack 58#8

[14]The 1996 book "How Debuggers Work: Algorithms, Data Structures, and Architecture" by Jonathan B. Rosenberg has been recently complemented by the "Hacker Disassembling" and "Hacker Debugging" in the "Uncovered" series by Kris Kaspersky.

[15]`http://openrce.org`

[16]Such borrowing can be ill-advised, as the recent case of the "Sony Rootkit" has shown: `http://en.wikipedia.org/wiki/2005_Sony_BMG_CD_copy_protection_scandal`

the studied environment are indispensable and make an early appearance in the hacker track; by the same token, examining the low-level state of the system ("uncooked" even by development kits) is considered essential. For example, when learning networking, a hacker is likely to immediately encounter tools based on the *libpcap* and *libnet* libraries for capturing and constructing raw packets respectively; the view of OS networking provided by hacker tutorials is a cross-cut of the implementation details of the entire stack, together with tools for affecting it on every level[17].

We are going to illustrate the above points with the example of the "Phrack" hacker e-zine, and discuss its education potential in the standard CS curriculum. We chose "Phrack" because of its long history and its reputation within the hacking community. It also enjoys the distinction of being amply quoted by academic security publications[18].

While gathering material for a computer security course, I realized that most of the background material necessary to understand modern vulnerabilities and exploits was either conspicuously absent from the traditional CS curriculum or relegated to obscure footnotes generally ignored by the students. I was able to find some of the necessary material in a number of a recently published books such as "Exploiting Software" by McGraw and Hoglund, "Shellcoder's Handbook" by Koziol et al. Yet, as my recommended book list grew beyond ten items, I had to look for a different source to fill the gaps.

The electronic hacker magazine Phrack (`http://phrack.org`), dedicated to disclosing new exploitation techniques, provided an excellent selection of short articles with hands-on introductions to the missing background topics. These topics ranged from loading and relocation of binaries, dynamic linking, binary file formats, OS support for tracing and other debugging, memory allocation schemes, and IP stack implementations to common features of modern hardware hardly ever discussed in computer architecture courses (where they were apparently sacrificed in favor of hammering home various kinds

[17]See the recently published books "Building Open Source Network Security Tools: Components and Techniques" by Mike Schiffman, and "Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network" by Michael Gregg for examples of this approach, which has become the de-facto hacker standard long before it was codified in these and other books on the subject.

[18]In particular, the article "Smashing the stack for fun and profit" by Aleph One in Phrack vol. 7 issue 49 (1996) has become a standard reference for that type of attacks.

of RISC-vs-CISC arguments[19]). For a number of years Phrack has comprehensively covered the emerging exploitation methods from stack smashing, integer overflows, return-to-libraries, format string, and heap manipulation to GPS jamming and esoteric hardware faults. Its range of coverage of network issues is equally diverse.

I found that Phrack's short and to the point introductions to these topics were often the students' first encounter with the respective subjects. Despite their daily use, these topics were never a part of the students' awareness of their computing environment. From the attacker's point of view, this is the best of possible worlds: an attack vector that the defender is not even aware of is the most effective.

It is no coincidence that both Phrack articles and the best of recently published books on the fundamentals of computer security have to start with general exposition of a number of "mundane" technical topics in OS and networking. This situation suggests that these topics are being unduly overlooked in the present curriculum. Indeed, it is hardly possible to secure systems without awareness of how they really work and how their basic mechanisms are implemented. Concentrating on only a few links in the chain is never enough for the defender.

I found that Phrack and similar resources also helped the students to develop the hacking approach to exploring or second-guessing the logic of the respective implementations – a key attacker skill. The main obstacle towards developing the hacking approach was that the students' attitude, apparently conditioned by their previous programming experience, was that of a developer rather than a tester, a reverse engineer, or an attacker. In a nutshell, a developer is rewarded for sticking to tried-and-true recipes of making things work and avoiding non-standard and non-portable features, generally learns to trust API and interface documentation, etc. In short, developers intentionally confine themselves to working within narrowed models of computing environments, for better productivity or compatibility, whereas in reality such confines do not exist or can be bent by the attacker.

The role of this conditioning should not be underestimated. From an undergraduate student coding his homework assignment to a professional developer striving to meet a deadline, programmers are under pressure to

[19]As a result, it is not untypical for CS graduate students after such a course to know much about about an imaginary architecture but very little about their actual desktop PC.

produce working, easy-to-understand code as soon as possible, leaving them no time to "question everything", explore less-used features of libraries and protocols, or puzzle out how particular APIs are implemented (not to mention that the latter activity tends to be discouraged by proprietary software vendors, sometimes in an extremely heavy-handed manner).

To learn security skills it is necessary for the students and developers to be able to switch from this developer conditioning to the attacker way of thinking. Exposure to the hacker culture through hacker conferences like Defcon and others, Phrack and similar publications, as well as to comprehensive collections such as Packet Storm, (`http://packetstormsecurty.org`), helps this by providing the necessary culture shock, an "aha moment" ("Oh, so this is how it actually works!"), and should, in my opinion, be an integral part of every in-depth security curriculum. Recipes for preventing particular kinds of exploits are only a small part of the value provided by these materials. Their primary contribution lies in facilitating a deeper understanding of the underlying systems, by exposing the implicit assumptions of its designers, and by concentrating their attention on the "big picture" of the system and its environment, especially on issues typically glossed over.

Conclusion

The hacker culture has accumulated a wealth of efficient, if different from those accepted in the industry and academia, values, practices and approaches. In particular, the "curriculum" experienced by a hacker while learning his skills is substantially different. Yet, in many respects it produces impressive results that enrich the other cultures, and its influence and exchange of ideas with these others are growing. Therefore understanding and describing these "curriculum" and approaches is becoming more important day by day. Ignoring or marginalizing the hacker culture means passing up unique opportunities and valuable knowledge — at our own risk.