IEEE distributed systems Expert-authored articles and resources ONLINE
October 2007 (vol. 8, no. 10), art. no. 0710-mds2007100002

Published by the IEEE Computer Society

Education

Hacker Curriculum: How Hackers Learn Networking

Sergey Bratus• Dartmouth College

Over the past few years, we've all witnessed the hacker community's growing impact on both the IT industry and academia. Host and network exploitation techniques that used to be discussed in "underground" forums are now featured in book series from No Starch Press, Syngress, and other publishers. Furthermore, they've become standard training for security practitioners—see, for example, the SANS Institute's Security 504: Hacker Techniques, Exploits, and Incident Handling course (http://www.sans.org/training/courses.php).

It is now beyond doubt that the hacker community has developed efficient techniques for analyzing, reverse engineering, testing, and modifying software and hardware that challenge their college and graduate-school-educated peers. These techniques have let the community substantially contribute to the state of the art of practical computer security. Many academic researchers recognize that marginalizing the hacker community would be a mistake,[1] and increasing numbers of industry and government security practitioners attend hacker conferences such as DefCon (http://www.defcon.org) and Black Hat (http://www.blackhat.com). In fact, the US Military Academy and the US Naval Academy supported their students' participation in computer-security-related competitions at ShmooCon (http://www.shmoocon.org) and DefCon, respectively.

An important clarification is in order: when I speak of the hacker community's contributions, I refer primarily to the *white hat* and *gray hat* communities. The term "white hat" is usually applied to those who ethically oppose malicious abuse of computer systems. "Gray hats" might run afoul of some computer-related laws, but they're motivated to warn vulnerable parties and minimize damage (for more information, see the "A Gray Area" sidebar). "Black hats," who act for personal gain and without regard for possible damage, typically oppose knowledge transfer and disapprove of public disclosure because it detracts from their own efficiency. For further discussion of these terms, refer to Wikipedia's article on hackers (http://en.wikipedia.org/wiki/Hacker).

A Gray Area

Many unauthorized computer uses are harmful and ill advised. Sometimes, they're merely public nuisances; sometimes, they break various laws. However, lawmaking itself isn't immune to the influence of vested interests or ill-informed political agendas that in fact contravene the greater public good. With the advent of laws such as the US Digital Millennium Copyright Act and its further-reaching state analogs, and in the face of European initiatives to ban broadly defined "hacker tools," illegality of certain computer uses gets further and further divorced from actual or potential public harm. We should remember that even well-known security researchers have been threatened with no less than criminal prosecution for publishing their work. Furthermore, important public-interest disclosures, such as those of flaws in electronic voting systems, became possible only through the willingness of third parties to counteract legal threats from vendors who apparently chose to invest in silencing researchers rather than improving vulnerable products. For more information, see http://www.chillingeffects.org.

For better or worse (and I believe that it's much for the better), hacker knowledge and skills (in varying degrees and aspects) are no longer limited to the isolated few. For example, DefCon, the largest US hacker convention, attracted an estimated 7,000 attendees this year, from high school students to professional security administrators and developers. The hacker culture will undoubtedly continue to attract more participants. Therefore, industry and academia leaders must understand this culture and be aware of its values, unique strengths, and weaknesses—whether they want to benefit from hackers' contributions or defend themselves from the "bad apples" who reject the hacker ethics.

Comparing hacker and developer worlds

Elsewhere, I've outlined several crucial differences between the typical experiences of a traditionally trained developer or computer science student and those of a hacker.[2] I repeat this outline here.

Many aspects of the hacker culture stem from hackers' dissatisfaction with certain trends in the industry and academia. In my opinion, these same trends contribute to the present wealth of software weaknesses and vulnerabilities. To summarize a plethora of observations on the economics of software and hardware insecurity, the typical developer is likely to encounter

- pressure to follow standard solutions—that is, the "path of least resistance" to "just making it work." As long as "it works," detailed understanding is often optional. As a result, developers might not realize the security implications of deviating (intentionally or not) from the standard templates or recognize their inherent weaknesses.
- training that discourages exploring the underlying API because the extra time investment rarely pays.
- a severely limited view of the API, with few implementation details. (Such limitations are often intentional or even part of the vendor's business model.)
- encouragement to ignore or avoid infrequent border cases because investing effort in handling unusual conditions is a "waste of time." As a result, developers might not understand the effects of nonstandard input on the system.
- explicit directions to ignore specific problems that are other developers' domain. (In private communication, I was told that a major vendor advised customers that their product's security was the customers' responsibility. The vendor expected the customers to "just put it behind a firewall.")
- a lack of tools for examining the system's state, let alone changing it outside the API.

In a typical academic setting, similar pressures exist in curriculum development. The growing number of topics to cover considerably limits the time students can allocate for any assignment or task. Therefore, instructors tend to carefully plan their teaching environments to minimize any distractions, such as students encountering a complicated border case. For example, creating "wrapper" libraries that isolate the students from the unwanted complexity is common. This often leads to unrealistic teaching environments that impart little of the real world's complexity, giving students false expectations and causing problems when they become industrial developers.

Moreover, unless students are specifically encouraged and given the opportunity to explore the results of various programming errors, they are likely to adopt the seemingly time-efficient "copy-and-paste" approach to the prescribed solution templates without additional exploration. Furthermore, it is common for the instructors to perceive some security-related topics—such as dynamic linking and binary file formats, OS support for debugger operation, and the internals of various programming languages' runtime mechanisms— as too complicated to explain.[2] Not surprisingly, computer security textbooks tend to include introductory chapters explaining these ubiquitous but rarely fully understood entities—for example, see *Exploiting Software: How to Break Code* (Greg Hoglund and Gary McGraw, Addison-Wesley, 2004) or *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* (Jack Koziol et al., John Wiley & Sons, 2004).

The frustration created by these trends is an important driving force behind hacker culture, which eschews the path of least resistance and seeks to fully understand the underlying standards and systems, including their border cases and vendor implementation differences.

In particular, hacker culture exhibits several defining traits:

- Hackers tend to treat special and border cases of standards as essential and therefore invest significant time in reading the appropriate documentation (which isn't a good survival skill for most industrial or curricular tasks).
- Hackers insist on understanding and exploring the underlying API's implementation to confirm the documentation claims.
- Hackers second-guess, as a matter of course, the implementor's logic (one reason for preferring a developer-addressed Request for Comments [RFC] to other forms of documentation).
- Hackers reflect on and explore the effects of deviating from standard tutorials.
- Hackers insist on having tools for examining the system's full state across interface layers and for modifying these states, bypassing the standard development API. If hackers lack such tools, developing them is a top priority.

These tendencies largely define how hackers learn and work.

The hacker approach is based on the keen interest in the internal workings of various operating systems, programming languages, and network protocol stack implementations. Hackers tend to learn about calling conventions and stack layouts, exception-handling mechanisms, system call implementation basics, and various raw socket and routing mechanisms much earlier than the average student—and often they do so at the beginning of their programming careers. This gives hackers a set of tricks that their traditionally educated peers may not be aware of.

The hacker networking curriculum

How do hackers usually learn their skills? Typically, they draw upon multiple sources, including

- classic textbooks that fellow hackers have highly recommended;
- electronic magazines such as *Phrack* (http://phrack.org), *2600* (http://www.2600.org/), and the more recent *Uninformed* (http://www.uninformed.org);
- online forums devoted to specific technical areas, such as OpenRCE (http://www.openrce.org) and Rootkit (http://www.rootkit.com);
- talks and private communications at hacker conventions such as DefCon, ToorCon (http://www.toorcon.org), and Shmoocon;
- source code from released tools; and
- various IRC communities.

These diverse sources share several themes that follow from hackers' preference for gaining *cross-layer* understanding of the studied systems. The multilayered nature of modern computer systems is both an engineering reality and a consciously taught design principle, such as the OSI networking model. (For an introduction to the cross-layer approach of observing and manipulating networks across OSI layers, see *Building Open Source Network Security Tools: Components and Techniques*, by Mike Schiffman [John Wiley & Sons, 2002] and *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*, by Michael Gregg [Syngress, 2006]. This approach had been the de facto hacker standard long before it was codified in these and other books.)

Economic pressures tend to encourage separation of developer expertise that roughly corresponds to layering schemes and their underlying interfaces. That is, interface boundaries often also become boundaries of expertise. Hackers, on the other hand, tend to adopt a cross-layer approach following the data through multiple layers of interfaces, with three guiding principles:

- inspecting the system state or network messages on all possible levels, down to the actual bits;
- injecting arbitrary data into the system or network; and
- identifying implementation peculiarities and second-guessing the implementors' logic.

Inspecting the system state or network messages down to the bit level

Hackers generally don't appreciate environments and interfaces that don't allow for such thorough inspection and often see them as a challenge.

A hacker would begin studying an unfamiliar topic with tools that allow such inspection and would learn about packet capture tools—such as tcpdump, Wireshark (formerly Ethereal), or Kismet—and the underlying libraries—such as libpcap and the Berkeley Packet Filter (BPF)—and their OS support early on. The hacker would then build on that knowledge to develop his or her own tools or to contribute plug-ins to popular tools such as Ettercap. The wealth of code contributions to cover new protocols and network link types demonstrates this tendency.

When a new communication medium attracts enough attention in the hacker community, developing a means of inspecting it becomes a top priority. For example, almost immediately after 802.11 wireless devices became affordable, hackers invested considerable effort in supporting the so-called monitor mode, in which the hardware and firmware passed raw 802.11 link layer frames, including control and management frames, to applications. As manufacturers update their chipsets and firmware, this effort must continue.

On the operating systems side, the same approach favors "live" kernel debuggers that let you inspect (and modify) the working kernel's data structures—for example, the WinDbg + LiveKD combination on Windows (http://www.microsoft.com/technet/sysinternals/SystemInformation/LiveKd.mspx) as well as other Sysinternals tools. For many OS hackers, the route to understanding internal OS data structures was through manipulating them with rootkits and anti-rootkit tools. For a historical perspective, see the "Linux rootkit LKMs HOWTO,"[3] Silvio Cesare's explorations of the Linux kernel module linking and loading mechanisms,[4] and *Phrack* articles on advanced kernel rootkit techniques (see the "Related Resources" sidebar). Also valuable are *Rootkits: Subverting the Windows Kernel*, by Greg Hoglund and Jamie Butler (Addison-Wesley, 2005), and *Designing BSD Rootkits: An Introduction to Kernel Hacking*, by Joseph Kong (No Starch Press, 2007).

Related Resources

For more resources, see the following *Phrack Magazine* articles on advanced kernel rootkit techniques:

- "Hacking the Linux Kernel Network Stack," (http://www.phrack.org/issues.html?issue=61&id=13)
- "Infecting Loadable Kernel Modules," (http://www.phrack.org/issues.html?issue=61&id=10)
- "Kernel Rootkit Experiences & the Future," (http://www.phrack.org/issues.html?issue=61&id=14)
- "Linux On-the-Fly Kernel Patching Without LKM," (http://www.phrack.org/issues.html?issue=58&id=7)
- "Linux TTY Hijacking," (http://www.phrack.org/issues.html?issue=50&id=5)
- "Static Kernel Patching," (http://www.phrack.org/issues.html?issue=60&id=8)
- "Weakening the Linux Kernel," (http://www.phrack.org/issues.html?issue=52&id=18)

Injecting arbitrary data into the system or network

For hackers, the logical next step after observing the system or network is testing its behaviors by manipulating the system's internal data or injecting custom-formatted messages into the network.

Hackers highly prize the ability to perform injection. Therefore, releases of tools and frameworks that enable injection are usually important for the community. Besides providing the groundwork for

further tool development, they're also a major educational opportunity. Studying the source code of such tools strongly benefits both attackers and defenders.

A network hacker will certainly encounter the libnet library and its derivatives, which let programmers send custom Ethernet frames and IP packets. Some of these packets mirror the normal network stack functionality, whereas others, although RFC compliant, are highly unlikely to be produced by most stack implementations. Some might be downright malformed. Sophisticated packet-level frameworks such as Ettercap or Scapy can be used to manipulate local area network or wireless local area network (WLAN) environments (for example, by affecting various state tables of hosts, switches, gateways, access points, and other network components). Or, hackers might also opt for advanced Layer 2 spanning tree and virtual local area network protocol manipulation with a tool such as Yersinia. In each case, the emphasis is on observing and learning from the network's reactive behavior to custom stimuli.

The hacker culture highly values both understanding the finer points of a protocol's intended functionality and putting unused protocol features to unintended uses such as covert channels or testing nonobvious network properties (including those that ISPs prefer their customers not to know about). Dan Kaminsky's popular "Black Ops of TCP/IP" talk series (available at http://www.doxpara.com) is perhaps the most graphic illustration of this trend.

Commodity 802.11 wireless networking provides another great example. As per the 802.11b/g specification, control and management frames on open or WEP-protected networks aren't authenticated, creating an opportunity for an attacker posing as either a wireless client station or an access point to manipulate the local wireless network. Link layer injection tools (such as the Airjack driver for Prism-based chipsets, http://xprobe.sourceforge.net) and the subsequent driver patches for other 802.11 platforms (culminating in Joshua Wright and Mike Kershaw's cross-driver LORCON framework, http://802.11ninja.net/lorcon) demonstrated the basic insecurity of unauthenticated 802.11b/g networks and prompted the move to more secure wireless standards. (*Wi-Foo: The Secrets of Wireless Hacking*, by Andrew Vladimirov, Konstantin Gavrilenko, and Andrei Mikhailovsky [Pearson, 2004], provides a detailed survey of the history of 802.11 security, auditing, and exploitation tools.)

Equipped with injection capability, a hacker can explore border and special cases of operating systems and protocol stacks at will. Such exploration generally leads to increased understanding of the target's security properties and, besides helping improve the state of the art, points out problem areas that administrators and security practitioners can improve (for example, attack signatures to watch out for or particular protocol features to normalize[5] or disable).

Identifying and second-guessing implementation peculiarities

Awareness of the implementation differences in standard protocols and mechanisms and the ability to second-guess the closed-source systems' implementors is a core hacker competence. Of course, it's not a solely hacker-specific skill—many developers who work in closed-source environments develop this skill while debugging undocumented features of their target platforms.

The popularity of implementation fingerprinting techniques illustrates this point. Typically, a hacker learns the finer points of TCP/IP by using the popular Nmap (http://www.insecure.org), hping (http://www.hping.org), or Xprobe (http://xprobe.sourceforge.net) tools for fingerprinting IP stack differences between different operating systems. These tools exploit the fact that core implementations and their derivatives respond differently to unusual or nonstandard TCP, UDP (User Datagram Protocol), and ICMP (Internet Control Message Protocol packets). (For the basics of fingerprinting and attack uses of TCP/IP, see *Network Intrusion Detection*, by Stephen Northcutt and Judy Novak [Sams, 2002]. For a detailed analysis of ICMP scanning techniques, see Ofir Arkin's fundamental paper.[6]) A nice counterpoint to these tools is techniques that disguise the target stack's identity by modifying its responses.[7,8]

With the advent of commodity wireless networking, fingerprinting efforts were carried into that arena[9,10] and helped uncover important driver vulnerabilities leading to OS kernel-level exploits.[11] Implementation differences can have much wider implications than just creating opportunities for fingerprinting or locating vulnerable configurations. Thomas Ptacek and Timothy Newsham's seminal research demonstrated that differences in IP fragmentation handling and TCP stream reassembly can also blind a network intrusion detection system (IDS) to crafted packets lethal to the target host.[12]

Understanding how packets or frames can be intercepted, crafted, and injected is crucial for network defenders. For example, concerned administrators would observe how their switches react to classic ARP poisoning or CAM table overflow attacks and would test their IDS systems for fragmentation-related weaknesses with fragroute or fragrouter tools. Similarly, they would take advantage of 802.11 drivers that support RF monitor mode and frame injection to run various auditing tools on their wireless networks.

The hacker culture has accumulated a wealth of practices and approaches that, if nonstandard, have proven their value and efficiency. In many respects, it produces impressive results that enrich other computing cultures, and its influence and exchange of ideas with these other cultures are growing. So, understanding the hacker learning experience 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.

In the next part of this article, I will describe how we used several of the unifying themes of the hacker experience I've mentioned here in teaching computer security at Dartmouth College. I also offer a vision for extending our approach to a community framework to lower the preparation cost of hands-on network security exercises for students.

Acknowledgments

This article continues the study of hacker curriculum that I began in "What Hackers Learn That the Rest of Us Don't," published in *IEEE Security and Privacy*.[2] Readers familiar with that article will find new material here in "The hacker networking curriculum" and the following sections.

References

1. G. Conti, "Why Computer Scientists Should Attend Hacker Conferences," Comm. ACM, vol. 48, no. 3, 2005, pp. 23–24; (http://www.rumint.org/gregconti/publications/20050301_CACM_HackingConferences_Conti.pdf).
2. S. Bratus, "What Hackers Learn That the Rest of Us Don't: Notes on Hacker Curriculum," IEEE Security and Privacy, vol. 5, no. 4, 2007, pp. 72–75.
3. "(Nearly) Complete Linux Loadable Kernel Modules," Mar. 1999; (http://packetstormsecurity.org/docs/hack/LKM_HACKING.html).
4. S. Cesare, "Runtime Kernel Kmem Patching," VX Heavens, Nov. 1998; (http://vx.netlux.org/lib/vsc07.html).
5. M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," Feb. 1998; (http://www.icir.org/vern/papers/norm-usenix-sec-01-html).
6. O. Arkin, "ICMP Usage in Scanning: The Complete Know-How," ver. 3, June 2001; (http://www.sys-security.com/archive/papers/ICMP_Usage_v3.0.pdf).
7. N. Provos, "A Virtual Honeypot Framework," Proc. 13th Conf. Usenix Security Symp., Usenix, 2004; (http://www.usenix.org/publications/library/proceedings/sec04/tech/full_papers/provos/provos.pdf).
8. K. Wang, "Frustrating OS Fingerprinting with Morph," presentation at DefCon 12, 2004; (http://www.defcon.org/images/defcon-12/dc-12-presentations/Wang/dc-12-wang.pdf).
9. J. Ellch, "802.11 VLANs and Association Redirection," Uninformed, vol. 2, Sept. 2005; (http://uninformed.org/?v=2&a=3&t=pdf).
10. J. Cache, "Fingerprinting 802.11 Implementations via Statistical Analysis of the Duration Field," Uninformed, vol. 5, Sept. 2006; (http://uninformed.org/?v=5&a=1).
11. J. Cache and D. Maynor, "Hijacking a MacBook in 60 Seconds or Less," video, Washington

Post, 2 Aug. 2006; (http://www.washingtonpost.com/wp-dyn/content/video/2006/08/02/VI2006080201424.html).
12. T.H. Ptacek and T.N. Newsham, Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, tech. report, Secure Networks, Jan. 1998.

Sergey Bratus is a senior research associate in Dartmouth College's Computer Science Department. Contact him at sergey@cs.dartmouth.edusergey@cs.dartmouth.edu.

Related Links

- "Hacking Tricks Toward Security on Network Environments," Seventh Int'l Conf. Parallel and Distributed Computing, Applications and Technologies (http://doi.ieeecomputersociety.org/10.1109/PDCAT.2006.66)
- "Hacking the Business Climate for Network Security," Computer (http://doi.ieeecomputersociety.org/10.1109/MC.2004.1297316)
- "Getting to Know Your Enemy," IEEE DS Online (http://doi.ieeecomputersociety.org/10.1109/MDSO.2004.13)
- DS Online's Software Engineering Community

Cite this article:

Sergey Bratus, "Hacker Curriculum: How Hackers Learn Networking," IEEE Distributed Systems Online, vol. 8, no. 10, 2007, art. no. 0710-mds2007100002.

Department Editor:

Maria Ganzha, Education Archives; Visit Maria's homepage (http://www.ganzha.euh-e.edu.pl)