Education
Hacker Curriculum: How We Can Use It in Teaching
Sergey Bratus and Christopher Masone • Dartmouth College

In the October column "Hacker Curriculum: How Hackers Learn Networking" (http://dsonline.computer.org/portal/site/dsonline/index.jsp?pageID=dso_level1&path=dsonline/2007/10&file=ox002edu.xml&xsl=article.xsl), Sergey outlined several trends in the hacker ways of learning and analyzing systems and networks. Here, we examine how to use elements of hacking experience in the regular computer science curriculum.

Our approach to teaching computer security
Dartmouth’s Computer Science Department has been offering a course in security and privacy since the 2000–2001 academic year. The class’s particulars have varied widely over the years in response to both the ever-shifting nature of the security landscape and the course instructors’ evolving interests. However, the goal has remained the same: give students a sense of the security and privacy issues that arise when software leaves the lab and gets exposed to the misuse and abuse of real users—malicious and otherwise. An introduction to the attackers’ view and methods is an irreplaceable part of this process.

On the technical side, we of course introduce students to network monitoring and manipulation tools, but we also encourage them to “roll their own.” For example, some students have built Yagi antennas from common household items (“cantennas”) to enable long-range directional wireless sniffing. Other students have observed, probed, and attacked networks using both commonly available tools (Ethereal, nmap, hping, Ettercap, and a variety of toolkits for spoofing headers and payloads of level 2 and level 3 network protocols) and home-grown software. They have also developed exploits for legacy application and authentication protocols that used to be deployed on Dartmouth’s network. For most of these exercises, we’ve provided the students with an isolated virtual environment, where such experiments couldn’t affect the college production network. We describe this environment later in this article.

In addition to protecting our production network, creating a virtual hacking environment provides an important benefit for students interested in independently exploring modern technologies. Unfortunately, many network observation and probing activities formerly regarded as natural and largely allowable manifestations of students’ curiosity and desire to learn now tend to be perceived as “cyberthreats” or “theft of service” and dealt with harshly. Arguably, many great researchers would get themselves in trouble if they carried out today the explorations they attempted on their computer systems 10 or 20 years ago. We applaud the efforts of the Hacker Foundation (http://www.hackerfoundation.org), a group that’s dedicated to providing safe working environments in which ethical hackers can hone their skills.

Securing a network against incursion, while generally requiring a high level of technical acumen, involves more than just learning about the adversary’s software toolkit. In addition to introducing students to computer security’s technical challenges, we make sure not to neglect its social aspects. We’ve had the students perform a variety of social-engineering attacks, design and conduct surveys to evaluate users’ online privacy preferences, and use software tools to explore how users’ mental models diverge from technological reality. For example, we show the students how to use a hex editor to expose hidden data in Microsoft Word documents, and then send them home with a mandate to
bring in interesting specimens from the wild. We've seen examples ranging from the humorous (a philosophy paper that was, apparently, originally titled “A Craptastic Load of Crap”) to the worrying (college memos containing usernames of every editor of the document), and everything in between. Showing students what’s really going on “under the hood” of software they use every day—whether an application such as Microsoft Word or a favorite operating system’s networking stack—can open their eyes to the complexity underlying current technology and the attendant nonobvious risks.

In the summer of 2006, Dartmouth brought together a group of computer science faculty, engineering faculty, and IT staff to guide the college’s approach to network security. Out of a desire to build local expertise in this space, the committee decided to reach out to a group of students who had been exposed to our security curriculum and invite them to participate. After training provided by Intelguardians, an external security consulting firm, this team of students, faculty, and staff began a campus-wide network vulnerability assessment. During this process, the group uncovered a number of potential network and application vulnerabilities, enabling Dartmouth's technical staff to address them. The team then helped create a strategy for ongoing testing and remediation, in addition to generating some new lines of research for faculty. This relationship between technical-services staff and a faculty-led student group is, to our knowledge, unique in higher education. The hands-on experience in a real, deployed network has been invaluable in helping students understand the operational constraints faced by network administrators in modern enterprise environments, ameliorating the disconnect that often exists between the academic and operational views of networking and security.


**Hacker materials in the standard curriculum**

When preparing to teach the security course, we found hacker sources such as Phrack and other e-zines, released tools and their HowTos, and Defcon and other hacker convention materials extremely useful. The major obstacle that students have faced appears related to their previous programming experience, which is primarily that of a developer rather than a tester, reverse engineer, or attacker. In a nutshell, developers are rewarded for sticking to tried-and-true recipes of making things work and avoiding nonstandard and nonportable features. They also generally learn to trust API and interface documentation. In effect, they're confining themselves to narrowed models of working environments, whereas in reality, the assumed limitations don’t exist or can be bent by the attacker.

This conditioning’s role shouldn’t be underestimated. From an undergraduate student coding his or her homework assignment to a professional developer striving to meet a deadline, programmers are under pressure to produce working, easy-to-understand code as soon as possible. This leaves 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 proprietary software vendors tend to discourage that last activity, sometimes in an extremely heavy-handed manner).

To learn security skills, students and developers must be able to switch from this developer conditioning to the attacker way of thinking. Exposure to the hacker culture through hacker conferences and publications, as well as through comprehensive collections of security-related tools, papers, and exploits such as Packet Storm (http://packetstormsecurity.org) and Milw0rm (http://www.milw0rm.com), can provide the necessary culture shock that counteracts previous experiences. We believe that such exposure should be integral to every in-depth security curriculum.
Recipes for preventing particular kinds of exploits are only a small part of the value these materials provide. Their primary contribution lies in facilitating a deeper understanding of the underlying systems. They do this by

- exposing system designers’ implicit assumptions and
- concentrating students’ attention on the big picture of the system and its environment, especially on issues typically glossed over.

**A hands-on learning environment: Challenges and lessons**

Most computer security teachers would like to give their students hands-on familiarity with the intelligence-gathering and attack methods being discussed. The ability to observe attacks from an administrator’s or a network analyst’s viewpoint is invaluable and usually provides deeper insights into topics beyond the security exercise proper, such as OS and network protocol design.

Such a teacher, however, faces three significant obstacles to making this a reality in his or her course. First, most such activities require root access on a machine, which college administrators might not be willing to give to students. Second, network administrators are mostly unwilling to see attack traffic (such as malformed packets that might upset their production equipment if accidentally misdirected) crossing their networks. However, they’ll probably consider setting up an isolated network to be too big an investment for a particular class. Third, even if enough machines with root access for everyone can be spared and a network can be set up, loading these machines with interesting exploits and tools and setting up the targets for the respective exercises require more preparation and time than are typical for an average computer science course.

Unfortunately, even when all this work is undertaken (as it was in our computer security course), the resulting system is too specialized and fragile to share with instructors elsewhere or to easily update. So, the potential benefits of community participation and incremental improvement through community contributions are lost.

For our course, we developed a set of tools that let the instructor easily deploy a simulated network of virtual GNU/Linux machines and configure it for the course’s tasks of network reconnaissance and remote exploitation. Further development plans include GUI tools for creating and managing the simulated network and for automating the creation of an emulated machine from a LiveCD image (of penetration-testing or forensic flavor, such as Backtrack, WHAX, Helix, or Trinux) or a hard drive image.

This collection of tools and the architecture of our simulated network dramatically simplifies preparation of environments in which host and network attacks can be safely observed and practiced, accidental damage easily contained, and reusable exercises easily added. We believe that further development of this environment would make preparation and sharing of exercises such as sniffing and redirecting traffic on switched networks, disrupting and hijacking connections, and exploring the effects of traffic insertion and deletion almost trivial. It would also bring preparation of more sophisticated exercises within reach of an unaided instructor.

**The architecture**

The host kernel was a custom-built Linux 2.6 kernel; the guest kernels were User-Mode Linux (UML) 2.4 kernels from the Debian/unstable distribution. We compiled the host kernel with the SKAS patch for better performance of the guest kernels and a more traditional memory layout in them. This memory layout was necessary for practicing buffer overflow exploits on the simulated machines; without the SKAS patch, guest UML kernels allocate stacks for their processes in a different virtual address range. Since we set up the original system, we’ve also experimented with Qemu and VMWare’s VMPlayer, other virtualization options that are now freely available.
File system images for the guest UML virtual machines were essentially Debian/unstable images. The simulated client and server machines used copy-on-write (COW) files that shared the same backing file system image. The simulated firewall/router machine used a separate Red Hat 8.0 distribution that happened to have a richer set of available IPtables modules.

For network simulation, we used TUN/TAP devices and kernel-bridging support in the host kernel (which is another reason for building a custom Linux kernel rather than using a stock one). The simulated network configuration included two virtual bridges that joined the virtual machines’ TUN/TAP interfaces. We chose this UML networking solution over simpler setups discussed in UML documentation because we wanted the students’ virtual machines to join the simulated network automatically after being rebooted from within. (This let students ssh into their designated machine, use the reboot command, and still be able to log in again in a couple of minutes.)

The result was the analog of a firewalled switched network, on which the students could

- practice various ARP (Address Resolution Protocol) poisoning and IP spoofing attacks,
- observe and use scanning and covert channels with packet capture and raw packet creation tools,
- hijack TCP connections,
- experiment with simple routing, and
- try to bypass firewalls.

We assigned each student a virtual machine, on which he or she had full administrator privileges. The ssh daemon handled authentication on login via the student’s public/private key pair. A management script cloned the “base” client machine for a new student, gave it a static IP, joined it to the specified bridge, and copied his or her private key into the file system. Additionally, each machine contained an administrator’s key and password. (So, the students could disable administrative access over the simulated network. However, they couldn’t disable direct modifications of their file system images via loopback mounting by root on the host machine, which is how we updated their environments when necessary.)

Additionally, we used the same cloning script to create target machines running various servers or cron-driven scripts simulating user activity (periodic TCP and UDP sessions, HTTP and DNS requests, and so on). Quickly manufacturing both the images and the user session scripts greatly accelerated the development of environments for attack exercises.

**Collaborative exercise development—a vision**

This environment could eventually become a community resource that sets a standard for hands-on security teaching.

By using a standardized virtual environment, the teachers who develop new exercise setups (including server images, scripts for agent behavior, and so on) will be able to easily share them with colleagues using the same environment. So, the extremely laborious part of developing new hands-on exercises could be distributed throughout the community. New exercise setups could be viewed as plug-ins for a common architecture; various free software projects have proven this development model’s benefits again and again. (For this effect to take place, the common shared architecture must be well thought out and carefully implemented. Accordingly, we consider the problem of designing such a system for a virtual instructional environment to be of nontrivial research interest as a software engineering problem.)

To facilitate this collaboration, we propose creating a central repository of images, scripts, and other instructional materials. Modern virtual machines can make checkpointing snapshots of their state (VMware and Qemu). So, instructors can quickly deploy exercises that they have downloaded from such a portal, potentially saving hours of preparation.
An important part of this central repository will be a well-annotated, searchable collection of published exploits, made available to instructors for developing new scenarios. We propose building on existing public repositories such as Packet Storm, enhanced with a collaborative annotation scheme. Such a resource would significantly simplify the preparation of cyber exercises, putting them in reach of small groups of dedicated instructors.

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

Christopher Masone is a PhD candidate in Dartmouth College’s Computer Science Department. Contact him at cmasone@dartmouth.edu.

Related Links
- "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 and Christopher Masone, "Hacker Curriculum: How We Can Use It in Teaching," *IEEE Distributed Systems Online*, vol. 8, no. 11, 2007, art. no. 0711-mds2007110002.