Teaching the Principles of the Hacker Curriculum to Undergraduates

Sergey Bratus
ISTS
Dartmouth College
sergey@cs.dartmouth.edu

Anna Shubina
ISTS
Dartmouth College
ashubina@cs.dartmouth.edu

Michael E. Locasto
CSIS
George Mason University
mlocasto@gmu.edu

ABSTRACT

The “Hacker Curriculum” exists as a mostly undocumented set of principles and methods for learning about information security. Hacking, in our view, is defined by the ability to question the trust assumptions in the design and implementation of computer systems rather than any negative use of such skills.

Chief among these principles and methods are two useful pedagogical techniques: (1) developing a cross-layer view of systems (one unconstrained by API definitions or traditional subject matter boundaries) and (2) understanding systems by analyzing their failure modes (this approach works well with learning networking concepts and assessing software vulnerabilities). Both techniques provide a rich contrast to traditional teaching approaches, particularly for information security topics.

We relate our experience applying Hacker Curriculum principles to education and training programs for undergraduates, including the Secure Information Systems Mentoring and Training (SISMAT) program and the Cyber Security Initiative at Dartmouth College, which allows undergraduates to perform supervised red team activities on Dartmouth’s production systems.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education

General Terms

Experimentation, Measurement, Security

Keywords

security, information assurance, networking, SISMAT, Hacker Curriculum, teaching failure modes

1. INTRODUCTION

An understanding of information security and assurance holds an increasingly important place in the education of Computer Science students, many of whom will be asked to deal with new security and control challenges arising from a variety of problems, ranging from the privacy challenges of social networks and the use of Electronic Medical Records to a new Smart Energy Grid linked with and controlled by commodity networks and computing systems.

To meet the challenges of modern computer security practice, students must be able to switch from their conditioning by traditional computer science and software engineering curricula to the attacker’s way of thinking. Successful work in the field of information security and assurance requires developing a specialized mindset: the essence of real security issues often lies in the complex interplay between **trust**, **models**, and **control**. It is precisely the skill to question trust relationships and the assumptions underlying controls and models that a budding security specialist must nurture.

Over the years, the hacker community has had a great deal of success doing so, and as a result, it has produced an informal but highly coherent and efficient “curriculum” that approaches the same topics as the traditional curricula from a variety of unusual angles. Far from being a haphazard collection of *ad hoc* “hacks”, this curriculum exhibits a complex and rich — although often implicit — structure. The hacker community has developed “cross-layer, cross-field” methods of approaching computer systems that we can and should learn from [1, 2].

In order to be meaningful and practical, we believe that the computer security curriculum must include both “defender” and “attacker” perspectives and skills in each of the sub-areas of computing (e.g., systems, networks, human-computer interaction, relevant aspects of social engineering). Yet, teaching skills popularly associated with the hacker community still carries a substantial stigma. Many educational institutions frown on the courses taught within their walls having something to do with “hacking,” a topic which they deem to be synonymous with unethical or illegal behavior.

This stigma results from serious misunderstandings that surround the hacker community. In this paper we hope to highlight these misunderstandings and to lay the groundwork for a more productive approach to the discussion about the hacker community’s role in computer security research and practice.

1.1 Overview

This paper relates our experiences stemming from collecting and applying Hacker Curriculum principles to the education of our students and interns. These experiences mainly stem from two programs: the Dartmouth Cyber Security Initiative, a home-grown red team of students (guided by research personnel and Dartmouth IT staff) that help to secure the university’s production networks, and the SISMAT (Secure Information Systems Mentoring and Training) educational outreach program.[1] We will give a brief overview

[1]`http://www.ists.dartmouth.edu/projects/sismat.html`

of SISMAT [6], introduce the notion of the "Hacker Curriculum", enumerate some Hacker Curriculum Principles, and describe the "failure modes" teaching principle as we applied it to the SISMAT program during the summer of 2009.

1.2 "Ethical" Hacking

We use the term "hacker" in the most non-pejorative sense possible. From this perspective, ***hacking*** **is the skill to question trust and control assumptions expressed in software and hardware, as well as in processes that involve human(s)-in-the-loop (a.k.a. "social engineering").** The ethical hacker community promotes the development of such skills in many ways and venues open to the public, such as conferences, e-zines, and websites. Hacking can be seen as a branch of engineering knowledge, and it shows the makings of an engineering discipline in (rapid) development.

Unfortunately, *hacking* has become a loaded term, particularly since the media has found it to be a convenient description of criminal mischief. We, however, see a need to separate the special knowledge, mindset, and skill from ill-advised, nuisance, or criminal behavior that might abuse this knowledge. Hackers possess valuable knowledge, a fact that certain governments understand and appreciate, although initiatives to recruit and hire hackers often meets with sharp criticism [4], with popular and vocal perception that hackers are somehow inherently threatening. This perception dominates in the media and periodically feeds into sensationalist reports of imminent cyber-terror and similar themes, such as occasional calls on the public to "give up certain freedoms" in order to "fix the Internet." [8]

As a community, hackers are defined by their skills and specific knowledge, similar to locksmiths, doctors, or martial artists. A doctor knows ways to harm humans, and might criminally abuse this knowledge. A locksmith is equipped to crack banks' vaults. A policeman is trained to use and is armed with deadly weapons. Yet none of them is defined by the potential misuse of the special skills they possess. Similarly, hacking is a special technological skill that can be misused, but should not be defined by its misuses.

We believe that this *knowledge-centric* or *skill-centric* approach best explains the activities, successes, and aspirations of the hacker community, and it inevitably leads to the imperative to learn from this community rather than shun it.

1.3 Culture Shock

Our experience teaching computer security in the SISMAT program and elsewhere leads us to believe that the most important service we can perform for our students is a *security culture shock*. Teaching students to question their assumptions, however, takes a great deal of effort: they simply cannot be told "question everything" because this spare advice provides them with no roadmap, specific principles, common tools, or peer-reviewed methods for intelligently questioning security assumptions. The Hacker Curriculum contains just such material, albeit largely uncurated.

Since the early days of personal computing, teaching environments in which aspiring young programmers learn their trade have gotten increasingly more efficient at imparting marketable skills without having the students "waste" time on puzzling out various computing failures unrelated to their intended learning tasks. But just as teaching environments that hide the underlying engineering complexities and challenges get better, something is lost: an experience of the many *failure modes* of computing environments.

This approach conditions many students to implicitly trust their programming environment — whether or not it is a real production environment or one constructed explicitly for teaching to mask complexity and to minimize potential confusion. Students also treat API layer interfaces as boundaries of exploration and desirable competence. **Nothing can be more damaging for the development of a security mindset than this type of conditioning.**

For example, software engineers tend to formalize their trust assumption in "layer models:" or patterns that their system designs follow (*e.g.,* the 7-layer OSI (Open Systems Interconnection) networking model). The borders of these layers become natural boundaries of trust (*i.e.,* blocks below boundaries are counted on by developers to not present surprises — changes in behavior, syntax, or semantics), and therefore become limits of expertise. **It is the hacker mindset and skill to question such assumptions of engineering trust in real world systems.**

2. SISMAT 2009

The SISMAT program provides a comprehensive approach to education and outreach for undergraduates and their faculty mentors who are interested in information security and assurance topics, but who lack the necessary local resources and expertise to gain traction in the area. SISMAT provides undergraduates with an intensive, two-week seminar and laboratory course in computer security. In addition, we arrange summer internships on information security topics for each SISMAT participant, and we coordinate with their faculty mentor to design a research project for the next academic year. The SISMAT program also includes a mentor development weekend; participants' mentors are invited for discussions and curriculum development.

While we have reported on our experiences with the pilot year (2008) of SISMAT elsewhere [6], in this paper we focus on how we applied Hacker Curriculum principles to specific lessons in the SISMAT lecture and lab agenda. For further details on SISMAT itself, we refer the interested reader to our previous paper [6]. The lecture and lab exercises for SISMAT 2009 focused largely on specific network security and ethical hacking skills, including traffic manipulation, PKI (Public Key Infrastructure), and code exploitation of C/x86 programs.

The academic world has long struggled with how to best incorporate information security topics and education in the classroom; providing hands-on training is challenging [9, 7, 10] from practical, ethical, and legal standpoints. In SISMAT, the focused nature and full day length of the labs and lecture allowed for more leisurely exploration of corner cases and difficult concepts [5]. We recognize that typical semester-length courses do not necessarily enjoy this advantage in pace.

Finally, the *Cyber Security Initiative* (CSI) is a joint undertaking of our campus IT and the Department of Computer Science. It actively involves students in the ongoing campus-wide security assessment of this large and diverse network infrastructure. Members of the CSI group, guided by senior researchers in computer security, are thereby exposed to the complexities and engineering challenges of a real-world production network, which provides another essential form of culture shock to students never previously exposed to such environments and the practicalities of administrating them [12].

3. A CROSS-LAYER APPROACH

The use of abstraction plays a fundamental role in computer system design. This fundamental principle of software engineering serves as the primary tool for managing complexity in computational systems, and it leads to the construction of systems organized in layers, with clearly defined interfaces between each layer. From the point of view of a programmer working within one layer, other layers are trusted to behave as specified, and he does not have to worry about *their* complexity. Thus interfaces at layer boundaries become not only contracts but also natural boundaries of programmer specializations and core competencies. Examples of layer boundaries include those defined by the OSI (Open System Interconnection) network model, those separating application code from libraries, userland and kernel code separation defined by system calls, and kernel modules from the Windows DDK/WDK.

Naturally, most courses designed to develop a core competence concentrate on programming entirely within one layer of the system or the network. Indeed, from the point of view of managing the complexity of teaching a subject or managing a software project, this approach is probably the most efficient. As a consequence, the students or trainees have enough non-trivial logic in their programs to worry about and cannot also afford excursions into substantially different and differently structured material. Eventually, this approach becomes ingrained and remains an important side-effect of their professional training. The process of building or creating computing and information systems requires the careful use of abstraction to manage an overwhelming amount of complexity.

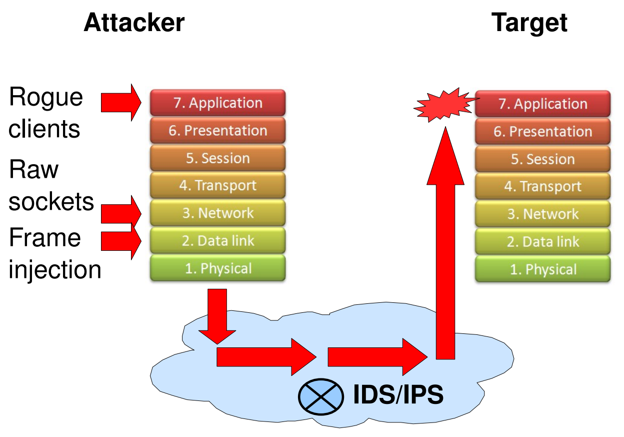

Figure 1: Hackers take the *cross-layer* view of systems and networks. They trace, examine, and craft data flows through layers and interfaces.

Although abstraction proves useful for building things, it can hinder the process of breaking, decomposing, repairing, or otherwise analyzing them. For example, the ability to develop an understanding of a basic software attacks and achieve successful delivery of an exploit to a target requires an almost completely orthogonal approach. We illustrate this approach in Figure 1, using exploitation of a user-level application as an example.

The key observation is that the attack payload (*e.g.,* a shellcode embedded within crafted input to the target) will be acted upon by multiple layers of the sending (attacker) system, the transmitting network (which may include an IDS (Intrusion Detection System) or IPS (Intrusion Protection System) that must remain unaware of the attack) and, finally, the target. None of the transformations of data in transit occurring in these layers should interfere with the exploit payload for the attack to succeed, and additional transformations might need to be applied so as to avoid detection and reaction by the IDS or IPS.

We note that the IDS or IPS has to treat the data streams flowing through it differently than the targets by taking various heuristic shortcuts, because it is typically responsible for examining traffic to many targets, and thus does not operate on the same computational power "budget" as an individual target, and cannot match its data processing precisely. Both Ptacek and Newsham's famous paper [11] and Paxson's paper [3] discuss different IDS evasion techniques that leverage differences in network stack implementations between the targets and the IDS to create different views of the packet stream at the IDS and the target. As a result, the IDS sees the attack stream as innocuous (or at least something that fails to trigger any known attack signatures), and in the worst case can cause the IDS to silently fail open.

We also note that while the payload must take the usual path to the targeted application, the attacking machine is under no such constraints: data it *produces* need not follow normal processing paths and packet composition rules. An attacker's program, therefore, may be instrumented to (1) form the attack packets and send them directly, bypassing the kernel's session and transport layers, or even the network layer (for LAN attacks) to inject straight into the link layer (as shown on the attacker side of Figure 1; and (2) suppression of some of the system's normal functionality (so that the attack tool's responses do not get accidentally invalidated by the kernel's normal reactions to malformed or unsolicited input).

Thus, successful engineering of exploits requires sufficient understanding of all the transformations that take place on the path from the attacker to vulnerable code in a running process on the target. Consequently, successful defenses must incorporate the same **cross-layer** understanding.

Not surprisingly, attack and network security tools in general depend on several libraries that provide ways to bypass the standard flow of data through the network stack. An example of the network security toolchain (described in great detail in Michael Schiffman's book "Building Open Source Network Security Tools: Components and Techniques") can be found in Figure 2.

Nurturing the ability to understand the limitations of algorithms, libraries, APIs, tools, and software systems requires a significant investment of time. Nevertheless, if we do not encourage and permit students to exercise this freedom within the formal curriculum, we run the risk of continuing to produce students that lack an appreciation of software security issues [13].

4. FAILURE MODES ACROSS NETWORK LAYERS

Typical CS network exercises seldom start with a mission of discovery: having to find out such facts about a network as what hosts are present in it, which services are actually available on them, what kinds of packets get filtered on entry ("ingress") or exit ("egress"), and similar reconnaissance activities. Instead, most network introductions center on abstract models of communication, brief glimpses of protocol diagrams and packet header layout charts, particular APIs or client-server libraries. Furthermore, such exercises typically assume that the transaction endpoint(s) are known in advance and available, *i.e.,* communications are not hindered, fully or partially, by adverse network conditions, misconfiguration, router access lists, firewall settings, server configuration, and the like. While such an approach suffices for a gentle introduction to the topic of network communication, it almost wholly concentrates on the way things *should work* rather than the ways that they might be *made to fail.*[2]

[2]Admittedly, failure modes may not necessarily be appropriate for teaching some topics: for example, teaching CS 1 by enumerating all the possible Java compiler warnings and errors does not seem practical, although students probably would appreciate some commentary on and interpretation of these common warnings.

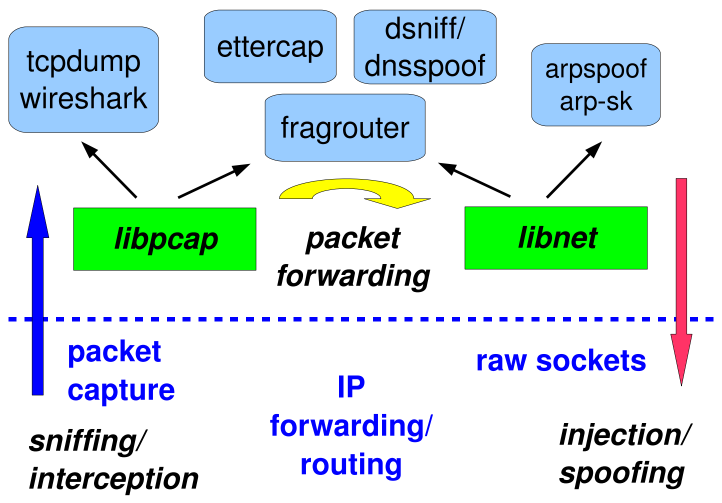

Figure 2: Open source network security toolchain allows bypassing OS network stack layers, enabling extraction and injection of packets from and into the kernel.

In contrast, hackers concentrate on failure modes of network communications, because distinguishing between different kinds of and reasons for transaction failures is critical for network reconnaissance (identifying potential targets and obstacles to exploitation). For example, scanning tools such as *nmap* crucially depend on interpreting signs of connection failure, firewall probing tools such as *firewalk* imitate routing failures, and so on.

In the SISMAT training, we direct the students to explore a variety of network failure modes, and, as a side effect, discover basic port scanning and firewalking techniques. In particular, we direct them to observe the result of attempted connections to non-existent hosts within and outside their LAN, to services not available on a host or filtered via different mechanism, in the presence of invalid and missing routes, *etc.*

As a result, the students gain enough confidence to start looking for failure conditions specifically rather than dismissing them as unintelligible, and to extract valuable reconnaissance information from those conditions.

4.1 Beyond the Socket

Typical CS curricula start with network programming APIs such as sockets and the client-server application model, and then expands "upward" to application protocols and distributed applications, and "downwards" to the details of TCP and mathematical models of congestion control. These topics, however, tend to be taught in separate courses and, as a result, tend to exist separately in the minds of most students.

Hacker analysis takes a different approach, connecting the different networking layers and programming abstractions from the outset. In particular, it is typical for a hacker to acquire an early understanding of which specific kinds of packets and OS network stack data structures result from each of the system calls involved in establishing a socket, and how different stacks differ in this regard.

An intermediate level step (although sometimes taken as a first one in hacker-style networking), is to build the capability for crafting arbitrary packets and injecting them into the network. This step is extremely important from the "culture shock" perspective – it helps the student realize that (1) network packets and frames are malleable and (2) the network by itself is merely a medium that would happily carry along any packet that looks well-formed to the equipment it is comprised of; the network contains hardly any trust guarantees.

The next step is the understanding of how to affect the OS network stack's states and thereby exploit them by sending crafted packets. Ptacek's paper [11] provides a good introduction to the topic and the source for the tool fragrouter gives an excellent and concise set of practical examples. These resources start the students thinking on how traffic patterns they describe could be generated. We then introduce them to *libnet* and *libnet*-based tools from simple ones such as *sendip*, *nemesis*, and *packetdude*, to advanced environments such as *scapy*. As a further example of the type of "problem" that arises with this "failure mode" teaching pattern, the students must consider how to suppress their OS stacks' natural reactions to incoming packets generated as a side-effects of the above manipulations. A simple example is presented by disabling ICMP redirects that would normally accompany ARP poisoning; `http://sockstress.com` provides a more comprehensive example. This pattern allows us to introduce some of the rich capabilities of the iptables/netfilter architecture and discuss its system of hooks within the Linux network stack, as well as the paths taken by a packet through that stack.

5. ROOTKITS: DECEPTION IN DEPTH

Similar to the cyber-defender maxim of "defense in depth", attackers also employ multi-layer deception schemes to throw the administrators off their trail and conceal evidence of malfeasance. In fact, this defense maxim may have been shaped by attacker practices that pre-dated it. We illustrate this "defense in depth" principle – critical for defenders to understand, since a lower level deception can be protecting a higher level one, and vice versa, in Figure 3.

We note that developing efficient rootkits is a task that requires careful understanding of a number of kernel data structures, as well as the ways those are used by different functions of the kernel, such as the difference between how scheduling and reporting code paths

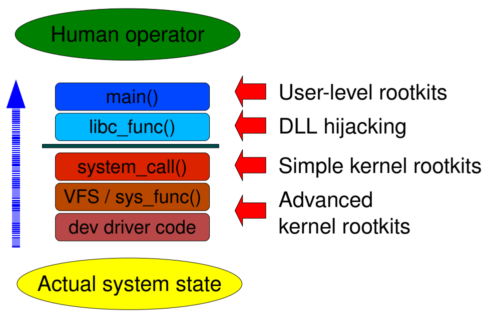

Figure 3: Compromising lower layers of the OS enables deception along the normal data flow path to human operator.

access and affect these data structures. Thus reading of openly available rootkit code not only helps the students to familiarize themselves with these threats, but also may provide valuable insights into OS kernel architecture itself. As with networking, a recommended reading list of helpful hacker materials on rootkits can be found at the Web site `http://hackercurriculum.org/`.

6. CONCLUSION

We posit that the largely undocumented and informal "Hacker Curriculum" contains a number of principles and methods that can prove useful in educating undergraduates in the art of ethical computer security assessment. We applied the "failure mode" principle during a summer cyber-security training camp for undergraduates. By enabling them to view the state of the OS network stack and the network itself as they made both crafted and legitimate requests for resources, they gained an appreciation for the limits and outlines of networks as well as a measure of competence with standard network details and technology.

Hacking involves the ability to develop and nurture a cross-layer view of systems rather than one where abstraction, APIs, or traditional subject domain limits create artificial boundaries. In addition, we find that exploring and understanding the failure modes of CS concepts and computer systems (particularly with regard to networks) provides a useful teaching technique.

Acknowledgments

The authors would like to thank the SIGCSE reviewers for their suggested revisions to the paper and particularly for their suggestions for our presentation during the conference. The authors would also like to thank Greg Conti for his help in organizing the CISSE working group in June of 2009, from which many of the ideas for this paper grew. In addition, we would like to thank the student participants in the SISMAT 2009 course for their willingness to explore these techniques with us. The authors would also like to acknowledge the support of the I3P, Dartmouth College, and DHS for the work described in this paper. This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

7. REFERENCES

[1] S. Bratus. Hacker Curriculum: How Hackers Learn Networking. *IEEE Distributed Systems Online*, 8(10), 2007.

[2] S. Bratus. What Hackers Learn That the Rest of Us Don't: Notes on Hacker Curriculum. *IEEE Security and Privacy*, 5(4):72–75, 2007.

[3] M. Handley, V. Paxson, and C. Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of the USENIX Security Conference*, 2001.

[4] M. E. Kabay. Hiring hackers (part 1): British Government Puts a Foot In It. `http://www.networkworld.com/newsletters/sec/2009/081009sec2.html`, August 2009.

[5] D. L. Knox, P. J. DePasquale, and S. M. Pulimood. A Model for Summer Undergraduate Research Experiences in Emerging Technologies. *SIGCSE Bull.*, 38(1):214–218, 2006.

[6] M. E. Locasto and S. Sinclair. An Experience Report on Cyber-Security Education and Outreach. In *Proceedings of the Annual Conference on Education in Information Security*, 2009.

[7] P. Y. Logan and A. Clarkson. Teaching Students to Hack: Curriculum Issues in Information Security. In *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 157–161, New York, NY, USA, 2005. ACM.

[8] J. Markoff. Do We Need a New Internet? `http://www.nytimes.com/2009/02/15/weekinreview/15markoff.html`, February 2009.

[9] P. Mateti. A Laboratory-based Course on Internet Security. In *SIGCSE '03: Proceedings of the* 34th *SIGCSE Technical Symposium on Computer Science Education*, pages 252–256, New York, NY, USA, 2003. ACM.

[10] B. A. Pashel. Teaching Students to Hack: Ethical Implications in Teaching Students to Hack at the University Level. In *InfoSecCD '06: Proceedings of the 3rd annual conference on Information security curriculum development*, pages 197–200, New York, NY, USA, 2006. ACM.

[11] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. In *Snort.org*, January 1998.

[12] R. Speers and E. Tice. Cyber Attacks on the Dartmouth College Network. *Dartmouth Undergraduate Journal of Science (DUJS Online)*, Fall 2009. `http://dujs.dartmouth.edu/fall-2009/cyber-attacks-on-the-dartmouth-college-network/`.

[13] G. White and G. Nordstrom. Security Across the Curriculum: Using Computer Security to Teach Computer Science Principles. In *Proceedings of the* 19th *National Information Systems Security Conference*, pages 483–488, October 1996.