Organizing Large Scale Hacking Competitions

Nicholas Childers, Bryce Boe, Lorenzo Cavallaro, Ludovico Cavedon,
Marco Cova, Manuel Egele, and Giovanni Vigna

Security Group
Department of Computer Science
University of California, Santa Barbara
{voltaire,bboe,sullivan,cavedon,marco,manuel,vigna}@cs.ucsb.edu

Abstract. Computer security competitions and challenges are a way to foster innovation and educate students in a highly-motivating setting. In recent years, a number of different security competitions and challenges were carried out, each with different characteristics, configurations, and goals. From 2003 to 2007, we carried out a number of live security exercises involving dozens of universities from around the world. These exercises were designed as "traditional" Capture The Flag competitions, where teams both attacked and defended a virtualized host, which provided several vulnerable services. In 2008 and 2009, we introduced two completely new types of competition: a security "treasure hunt" and a botnet-inspired competition. These two competitions, to date, represent the largest live security exercises ever attempted and involved hundreds of students across the globe. In this paper, we describe these two new competition designs, the challenges overcome, and the lessons learned, with the goal of providing useful guidelines to other educators who want to pursue the organization of similar events.

1 Introduction

Computer security has become a major aspect of our everyday experience with the Internet. To some degree, every user of the Internet is aware of the potential harm that can come from its use. Therefore, it is unsurprising to see that computer security education has also improved substantially in the past decade, in terms of both the number of university undergraduate and graduate level courses offered and the type of educational tools used to teach security concepts. This increase in the importance of computer security is also reflected by the offerings of the job market. For example, at `www.computermajors.com` it is stated that while entry-level salaries for computer science professionals specializing in web development start at around $75,000, "those who specialize in computer and online security can earn up to $93,000." [2]

An important aspect of computer security education is hands-on experience. Despite the importance of foundational security classes that focus on more abstract concepts in security, such as cryptography and information theory models, Internet security issues often require substantial hands-on training in order to be understood and mastered. Thus, it is important to improve security training by

C. Kreibich and M. Jahnke (Eds.): DIMVA 2010, LNCS 6201, pp. 132–152, 2010.

providing novel approaches, which complement the existing, more traditional educational tools normally used in graduate and undergraduate courses on computer security.

A class of these tools is represented by *security competitions.* In these competitions, a number of teams (or individuals) compete against each other in some security-related challenge. As an educational tool, these competitions have both advantages and disadvantages. A notable advantage (and the main reason why these events are organized) is that competition motivates students to go beyond the normal "call of duty" and explore original approaches, sometimes requiring the development of novel tools. Another advantage is that students usually operate against a determined opponent while under strict time constraints and with limited resources thus mimicking a more realistic situation than one can reproduce using paper-and-pencil *Gedanken* experiments. Unfortunately, security competitions have one major disadvantage: they usually require a large amount of resources to design, develop, and run [10,11].

Designing security competitions is hard, as they need to be at the right level of difficulty with respect to the target audience. If a competition is too difficult, the participants become frustrated; if a competition is too easy, the participants are not challenged and will lose interest. Ideally, a competition will provide a variety of challenges of differing difficulties such that all participants of various skill levels are both challenged by the tasks and gratified by success. An additional design challenge is how to evaluate the participants and score their actions. Ideally, a scoring system is fair, relevant, and automated allowing the best participant to be clearly identified beyond any reasonable doubt. However, scoring systems must be hard to reverse-engineer and cheat, as it would be ironic to have a security competition requiring "adversarial" approaches and "oblique" reasoning that utilizes a scoring system reliant on the "good behavior" of the participants.

Developing security competitions is time consuming since a specific competition can seldom be reused. For example, consider a competition where the flaws of vulnerable applications have to be identified by the participants. Once a competition ends, it is likely that the vulnerabilities discovered will be discussed in blogs and "walk-through" pages.[1] After the disclosure of vulnerability details, these services cannot be reused and new ones must be developed.

Running a security competition is challenging, because the competition's execution environment is often hostile and thus difficult to monitor and control. Therefore, it is of paramount importance to have mechanisms and policies allowing for the containment of the participants and for the easy diagnosis of possible problems. In addition, security competitions are often limited in time, thus unexpected execution problems might make the competition unplayable, wasting weeks of preparation.

Annually, since 2003, we organized an international, wide-area security competition involving dozens of teams throughout the world. The goal of these live exercises was to test the participant's security skills in a time-constrained setting.

[1] See for example, the walk-through for the 2008 DefCon qualification challenge at `http://nopsr.us/ctf2008qual`

Our competitions were designed as educational tools, and were open to educational institutions only.

From 2003 to 2007, each edition of the competition was structured as a Capture The Flag hacking challenge, called the iCTF (further described in Section 2), where remote teams connected to a VPN and competed independently against each other, leveraging both attack and defense techniques. In this "traditional" design, borrowed from the DefCon Capture The Flag competition (CTF), teams received identical copies of a virtualized host containing a number of vulnerable services. Each team's goal was to keep the services running uncompromised while breaking the security of other teams' services. Subsequent editions of the competition grew in the number of teams participating and in the sophistication of the exploitable services. The design, however, remained substantially unchanged.

In both 2008 and 2009, we introduced completely new designs for the competition. In 2008, we created a security "treasure hunt" where 39 teams from around the world had to compromise the security of 39 dedicated, identical target networks within a limited time frame. In 2009, 56 teams participated in a competition where each team had to compromise the browsers of thousands of simulated users, compromise the users' banking accounts, and finally make the users' computers part of a botnet.

In addition to this paper's content, we provide scoring software, virtual machines and network traces from each of our previous iCTFs.[2] The software and virtual machines are useful for the creation similar competitions and the network traces are useful for researchers working on intrusion detection and correlation techniques.

This paper describes these new competition designs, how they were implemented and executed, and what lessons were learned from running them. By providing a detailed discussion of the issues and challenges overcome, as well as the mistakes made, we hope to provide guidance to other educators in the security field who want to pursue the organization of similar competitions. In summary, the contributions of this paper are the following:

- We present two novel designs for large-scale live security exercises. To the best of our knowledge, these are the largest educational security exercises carried out to date. Their design, implementation, and execution required a substantial research and engineering effort.
- We discuss the lessons learned from running these competitions. Given the amount of work necessary to organize and the current lack of documentation and analysis of such events, we think this paper provides a valuable contribution.

This paper is structured as follows. In Section 2, we present background information on both security competitions in general, and on the iCTF in particular. Section 3 presents the design of the 2008 competition and its characteristics. Section 4 describes the competition held in 2009. In Section 5, we describe the

[2] All files can be obtained at `http://ictf.cs.ucsb.edu/`

lessons learned in designing, implementing, and running these competitions. Finally, Section 6 briefly concludes.

2 Background and History

The best-known online security challenge is the DefCon CTF, which is held annually at the DefCon convention. At DefCon 2009, eight teams received an identical copy of a virtualized system containing a number of vulnerable services. Each team ran their virtual machine on a virtual private network, with the goal of maintaining uptime on and securing a set of services throughout the contest whilst concurrently compromising the other teams' services. Compromising and securing services required teams to leverage their knowledge of vulnerability detection. Compromising another team's service allowed teams to "capture the flag" thus accumulating attack points, whereas securing services allowed teams to retain flags and acquire defensive points. Several of the former DefCon CTFs followed a similar design [3].

Despite DefCon's nonspecific focus on security education, it inspired several editions of the UCSB International Capture The Flag (iCTF). One of the major differences between UCSB's iCTF and DefCon's CTF is that the iCTF involves educational institutions spread across the world, whereas the DefCon CTF allows only locally-connected teams. DefCon therefore requires the physical co-location of the contestants thus constraining participation to a limited number of teams. By not requiring contestants to be physically present, the UCSB iCTF additionally allows dozens of remotely located teams to connect to the competition network.

The UCSB iCTF editions from 2003 to 2007 were similar to the DefCon CTF in that the participants had to protect and attack a virtualized system containing a number of vulnerable services. A scoring system actively checked the state of these services, ensuring their availability. In addition, the scoring system periodically set short identification tokens, termed "flags" for each team.. Teams competed by securing their system and breaking into their competitors' systems to discover flags. The successful compromise of another team's service was demonstrated through the submission of a flag to the scoring system. Teams periodically earned defensive points for each service that retained its flags during the particular period. The team with the most points at the end of the competition won. These UCSB iCTFs in turn inspired other educational hacking competitions, such as CIPHER [6] and RuCTF [4].

Recently, a different type of competition has received a significant amount of attention. In the Pwn2Own hacking challenge [7] participants try to compromise the security of various up-to-date computer devices such as laptops, and smart phones. Whoever successfully compromises a device, wins the device itself as a prize. This competition is solely focused on attack, does not have an educational focus, and does not allow any real interaction amongst the participants who attack a single target in parallel.[3]

[3] Note that this type of design is very similar to early editions of the DefCon CTF, where participants competed in breaking into a shared target system.

Another interesting competition is the Cyber Defense Exercise (CDX) [1,5,8], in which a number of military schools compete in protecting their networks from external attackers. This competition differs from the UCSB iCTF in a number of ways. First, the competition's sole focus is defense. Second, the competition is scored in person by human evaluators who observe the activity of the participants, and score them according to their ability to react to attacks. This evaluation method is subjective and requires a human judge for each team thus yielding it impractical in a large-scale online security competition.

In both 2008 and 2009, we introduced two new designs, which, to the best of our knowledge, were never previously implemented in large-scale educational security exercises. These new designs are the focus of the remainder of the paper.

3 The 2008 iCTF — A Security "Treasure Hunt" Scenario

"It is the early morning and someone is frantically knocking at your door. Shockingly, this person turns out to be a high-profile counter-terrorist agent from a popular television series demanding that you help him. It comes to light that an evil organization known only as 'Softerror.com' has set up an explosive device set to detonate in seven hours. Only you and your small group of elite hackers have the necessary skills to infiltrate the Softerror network, find the bomb, and disarm it before it is too late."

This introductory scenario was given to the teams participating in the 2008 UCSB iCTF, which ran Friday, December 5, from 9am to 5pm, PST. 39 teams, averaging 15 students each, competed from educational institutions spread across several continents. Unlike former iCTFs where each team setup a vulnerable virtual server, in the 2008 iCTF we created 39 identical, yet independent copies of a small network with the topology depicted in Figure 1. The network was allegedly run by a terrorist organization called Softerror.com and was composed of four hosts, each of which belonged to a separate subnetwork and had to be compromised in a specific sequence. The final host contained a virtual bomb that had to be defused to complete the competition. A more detailed description of the four hosts comprising the Softerror.com network is included in Section 3.1.

The creation of 39 replicated networks, required a completely different software and hardware infrastructure from what was used in previous iCTFs. A total of 160 virtual machines needed to be hosted for each of the teams' networks and our test network. Furthermore, machines had to be isolated so that the teams could not interfere with each other. In addition, the network had to be set up in such a way that one network was only made available following the compromise of a previous network. This task was accomplished using a complex routing system that created the illusion of a separate dedicated network for each team. The details of the network setup are described in Section 3.2.

In addition to the new network topology, another significant difference between this competition and classic CTFs was that the teams did not directly attack each other. Instead, this competition modeled a more "real world" situation where the teams had to penetrate a remote network. Given only a single IP

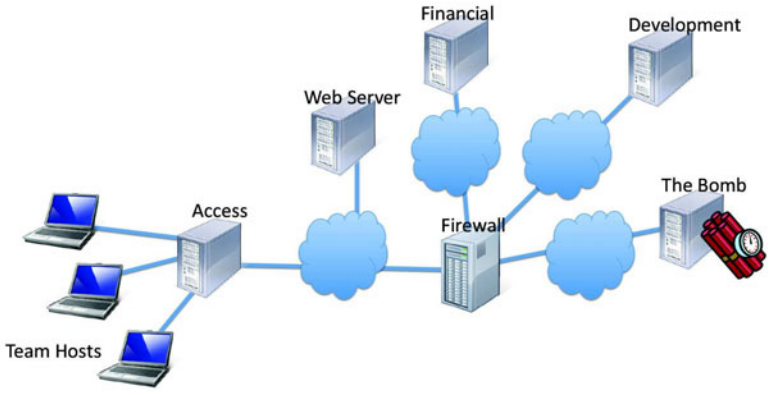

Fig. 1. The 2008 iCTF Network Topology

address, the teams needed to map out the network, discern what services were running, and then exploit these services to gain access to the various servers. Then, they could use the compromised host as a starting point to penetrate deeper into the network and repeat the process, discovering new machines and crafting the corresponding exploits.

Although this competition could have been presented as a "race" where the winner was the team to deactivate the bomb the quickest, we wanted to score the competition at a finer granularity, allowing us to easily monitor the progress of each team. Therefore, we decided to assign a score to each service and award points when the service was compromised. The teams were instructed to submit a report about each compromised service through a web site. The reports were manually examined and the points awarded by a judge. Although we had wanted to automate this process, it was decided that we wanted to know how each team broke into each server. Thus, because teams did not have to directly attack each other, and could make progress immediately after compromising a service, independent of the judge, we predicted to have sufficient time to manually judge each exploit. Unfortunately, as described in Section 3.3, this manual process had its share of problems.

Further differentiating from former CTFs, this competition had the notion of a "mole," which was a fictitious character who provided valuable clues. In order to gain access to clues, teams spent points earned from compromising services. Thus each team was faced with the choice to use points to potentially speed up service compromise. At the end of the event, the overall winner was the team who had the most points and defused the bomb.

3.1 Vulnerable Applications

The competition was divided into three major stages. The stages were arranged by increasing difficulty. The first stage was a web server with relatively simple vulnerabilities, which successfully compromising led to two possible paths: a

team could attack either the financial server or the development server. Exploiting either of these servers, opened access to the final stage, the bomb challenge. The network partitioning was accomplished by setting up a firewall within the second stage machines that blocked traffic to the bomb server. Thus, in order to gain access to the final stage, the teams had to gain root access on one of the second stage machines and disable the firewall.

The Web Server. The web server was the first stage of the "Softerror network" on which a simple PHP-based web site ran. Successful compromise required each team to discover a command injection vulnerability. While a simple code inspection of the site would reveal the vulnerability, teams were not given the source code. However, another vulnerability was introduced that allowed the reading of an arbitrary file. This vulnerability could be discovered by inspecting the normal operation of the site and noticing that a user-generated link actually contained a base64 encoded file system path. By exploiting this vulnerability, a team could then download the source PHP code and discover the command injection vulnerability.

The Financial Server. The Financial server hosted the Softerror "financial" information. This server was set up as a series of four stages that had to be completed in order. The primary focus of the financial stages was passwords and password management. The first stage required breaking a weak password given its hash, which could be easily accomplished by performing a web search of the hash. The other three stages contained various examples of poor password management. Stage 2 was a service with weak authentication, stage 3 stored the passwords in a plain-text file that was readable from a vulnerable application, and stage 4 contained an SQL injection vulnerability that could be used to reveal the final password.

The Development Server. The development server challenge required binary reversal. The service itself was accessible on a remote port and contained a format string vulnerability allowing arbitrary code execution. In reality, remote exploits can be very complicated to successfully exploit, as they often require detailed understanding of the target machine's underlying memory layout. To simplify matters, the application leaked a significant amount of information, thus easing the process of developing an exploit. Our intent was to make exploiting this vulnerability approximately as hard as breaking into the financial server.

The Bomb Host. The bomb was implemented as an Elf x86 "firmware" binary that could be downloaded, modified, and then uploaded back to the server. The bomb was also a binary reversing challenge, as no source code was made available. In order to successfully complete this challenge, a disarm function of the library had to be written by the teams. After writing, uploading, and successfully calling the function, the firmware deactivated the bomb thus completing the competition for the team.

3.2 Infrastructure

The most prominent difference between the 2008 iCTF compared to the classic CTF is the network topology. Normally, each competitor is responsible for

providing hardware to host the vulnerable virtual server. With this competition, a motivating goal was to mimic a real-world remote network penetration, including reconnaissance and multi-step attacks. To facilitate our goal, we decided to host all the machines ourselves, guaranteeing a level playing field and a network topology that we directly controlled. Even though the network given to each team consisted of only four machines, the number of teams required us to host 160 virtual machines.

It was readily apparent that we did not have enough physical hardware to support the required number of virtual machines if we were to use "heavy weight" virtualization platforms such as VMware or Xen. Although there are many reasons why we may have wanted to use these suites, the amount of hardware resources they require per guest made them unsuitable. In order to meet our virtual machine requirements within our hardware budget, we chose OpenVZ, a modified Linux kernel that provides a mechanism to elegantly solve this issue.

The advantage of using OpenVZ is that it allows for a very lightweight style of virtualization in a Unix environment. Instead of virtualizing an entire hardware stack, OpenVZ is a kernel modification that creates a sort of "super" root user on the host machine. This super user can then spawn guest machines that have their own operating environment, including the notion of a regular root user. This concept of kernel-level virtualization has existed for quite some time, especially on the BSD family of operating systems, where it is known as a "jail." One downside to this kernel-based approach is that the range of guest operating systems is very limited. Only Unix-like operating systems with a similarly patched kernel can be used as guests. The benefit of taking this approach is that the kernel itself is now shared among all the guests thus making resource sharing much simpler and very efficient. Once we decided on using OpenVZ, scaling our limited resources to the required number of machines turned out to be fairly simple.

Our operating system of choice was Ubuntu 8.10 with the aforementioned OpenVZ modified kernel. Each guest was created using the default OpenVZ Ubuntu template. We had six physical machines based on a Core2 quad core CPU with 4 Gigabytes of RAM and 1 Terabyte of disk space each. With some basic testing, it was determined that we could host a little over 40 guests on each individual machine using the default configuration settings. Although OpenVZ has a plethora of parameters that can be tweaked and adjusted to provide even greater scalability, we hit our target number with the default settings. Of those machines, four of them were selected to be our server "stacks." That is, on one machine we brought up 40 OpenVZ guests responsible for running identical copies of the "web server" service. Likewise, one machine was dedicated to each of the other three servers. Although this setup has merit based just on its simplicity, the primary reason for hosting identical services on the same physical machine is that it is trivial to write directly to the file system of OpenVZ guests from the host OS. In addition, OpenVZ supports executing commands from the host OS as a root user on the guest. With these two features, keeping all identical guests on the same machine greatly simplified management, as updating configuration files and sharing resources is made vastly simpler.

With the configuration of individual machines done, it was decided that they should be arranged around a single "firewall" host. This host had eight physical Ethernet ports, making it ideal to act as both a central location to record traffic and also help enforce routing rules. One requirement was that each team should see an isolated view of their network. To accomplish this, a basic iptables-based firewall was used to restrict traffic based on the IP address and virtual Ethernet device of the OpenVZ guest. That is, traffic coming from a virtual Ethernet device associated with one team would only be forwarded to the IP ranges also associated with that team. Furthermore, these firewall rules ran on the host machine, which was inaccessible from the guest machines. This setup gave us the unique ability to have both mandatory iptables rules to enforce game rules and iptables rules on the guest itself to simulate an internal firewall that could be modified. Thus we could ensure the order in which each stage had to be broken. Moreover, even if teams were to spoof traffic, they could not interfere with each other.

3.3 Overview of the Live Exercise

The competition took place on December 5th, 2008, from 9am to 5pm, PST. Even though we did not disclose the nature of this competition until the day of the event, we were able to get the network up and running beforehand by distributing a test virtual machine to test network connectivity, much like we would have done had we run a more traditional CTF. Although we had not stress-tested our OpenVZ-based network against the oncoming onslaught of connections, the network was stable. The largest connectivity issues came from a few teams that accidentally wrecked one of their OpenVZ guests. We had explicitly set it up so that all traffic had to go through the initial web server and the teams had to be very careful not to sever this connection. Even though we were explicit about being careful due to this very problem, we did attempt to help teams that had not heeded this warning. However, we also had to exercise great care while helping as a misconfigured firewall or an improper command could easily have taken the entire competition offline.

Another issue we overlooked was the name of our fictitious terrorist group. Clearly, the name Sof*terror* was supposed to invoke the idea of an evil software group doing evil software things. We had not considered that there might actually be a Soft*error* group that would supply friendly services such as consulting and development to their entirely benign customer base. Unfortunately, we did not think to check for similar names until after the competition began. Thus, when we received an email from a team asking if they were supposed to attack the real domain www.softerror.com, we were quite surprised. However, since the traffic of the whole competition was confined to a VPN with non-routable IP addresses, no actual attacks were carried out against external targets.

Initially we had hoped to augment the process of scoring exploits through the use of automated tools. Unfortunately, these tools were untested by the time the competition started and issues quickly developed. We were then faced with the decision of trying to hot fix them as the competition progressed or take a manual approach. With the competition running along, it was decided that we

should take the latter approach and that the teams would have to submit their exploits by email to be manually judged. When decided, the number of emails started to grow very rapidly as teams raced to submit their exploits. Given that we had about ten people involved as organizers, we were quickly overwhelmed with the torrent of emails and the competition suffered because of it. While we attempted to maintain fairness by giving emails that detailed successful breaks for a particular level to the same judge, the response time could be fairly long, which prompted teams to send additional emails, exacerbating the problem. Moreover, trying to work through all the issues related to scoring took us away from dealing with other issues that came up, such as the occasional connection issues mentioned above.

At the end of the competition, with minutes to spare, Team ENOFLAG managed to upload a modified version of the bomb firmware that successfully deactivated their bomb. Embroiled with the controversy surrounding our scoring procedures, we were at least relived that we could unequivocally decide the winner of the 2008 iCTF.

4 The 2009 iCTF — A Botnet Attack Scenario

The theme for the 2009 iCTF was "Know Thy Enemy!" For this competition, we decided to mimic the world of malware and design a competition that incorporated many features unique to the physiology of modern botnets. In this iCTF edition, each team played the role of an evil botmaster, competing against other botmasters for the control of a large number of simulated users. The 2009 iCTF was the largest security competition to date, with 56 teams representing more than 800 participants.

Scripts simulated users that were to be compromised and controlled by the participants. Each simulated user followed a cyclical pattern: First the user visited a bank (called *Robabank*, a pun on the real Rabobank) using a browser, and logged in using her credentials. The bank set a cookie in the user's browser to authenticate further requests. Then the user visited a news site (called *PayPerNews*) and randomly extracted a word from the content of the news. Teams could publish news on this site by paying with money from their bank account.

The word chosen by the user was then submitted to a search engine called *Goollable*. Goollable routinely crawled each team's editable webpage. One of the links returned by the search engine was chosen by the simulated user using a Pareto distribution that gave higher probability to the top links. The user directed the browser to this page, controlled by one of the teams, and was possibly compromised by a drive-by-download attack. Finally, the user returned to the bank web site and checked the balance of her account. Even though the user script was identical for all 1,024 users, the users browsed with different browsers each having multiple versions with unique vulnerabilities.

The goal of the participants was to lure a user to a web site under the participant's control, perform a drive-by-download attack against the user's browser, and take complete control of the user. Once the user was compromised, a team

had to do two things: i) transfer the money from the user's Robabank account to their own account thus accumulating "money points," and ii) establish a connection from the user to a remote host, called the *Mothership*, on which to send information identifying the compromising team. A team gained "botnet points" by performing this action. This last step was introduced to generate traffic patterns resembling the interaction of bots with Command-and-Control (C&C) hosts in real botnets.

Solving side challenges offered teams a third way to gather points. Challenges varied in type (e.g., binary reversing, trivia, forensics) and difficulty. Teams were awarded "leetness points" for solving a challenge.[4]

At the end of the game, the final score was determined by calculating the percentage of each team with respect to each type of points and computing a weighted sum of the percentages, where botnet points had a weight that was twice the weight of leetness points and money points. More precisely, given the maximum money point value across all teams, M, the maximum botnet point value, B, the maximum leetness point value, L, and the score in each of these categories for a specific team, m, b, and l, the total score for a team was computed as $25m/M + 50b/B + 25l/L$. Note that during the game, teams could exchange leetness points and botnet points into money points using the *Madoffunds* web site. The exchange rates varied dynamically throughout the competition.

This rather complex system of inter-operating components was a central aspect of the 2009 iCTF. That is, instead of just concentrating on single services or single aspects of the game, the participants were forced to understand the *system as a whole*. Even though this aspect generated some frustration with the participants, who were used to the straightforward designs of previous competitions, the purpose of the complexity was to educate the students on understanding security as a property of complex systems and not just as a property of single components.

We expected the teams to first solve a few challenges in order to gain leetness points. These points would then be converted into money points using the *Madoffunds* site and used to pay for the publishing of news on the *PayPerNews* web site. At the same time, a team had to set up a web page with content "related" to the published news. The idea was that a user would eventually choose a term in a news item published by a team, whose web site would score "high" in association with that term. This scheme is similar to the Search Engine Optimization (SEO) techniques used by Internet criminals to deliver drive-by-download attacks. Once the user was lured to visit the team's web site, the team had to fingerprint the user's browser and deliver an attack that would allow the team to take control of the user. The first team to compromise a user could transfer all the user's money to their account. Additionally, all teams that compromised a user could gather botnet points by setting up a bot that connected to the *Mothership* host.

The *Robabank*, *Madoffunds*, *PayPerNews*, *Goollable*, and *Mothership* sites had no (intended) vulnerabilities. The only vulnerable software components were the

[4] A discussion of some iCTF09 challenges can be found at `http://www.cs.ucsb.edu/~bboe/r/ictf09`

browsers used by the simulated users. In the following, we provide more details about the search engine behavior and the browsers whose vulnerabilities had to be exploited.

4.1 The Crawler and Search Engine

A crucial goal of each team was driving the vulnerable browsers to their web server. In order to do so, teams needed to perform SEO to boost their search results for desired keywords thus driving traffic to their web server. Each team was allowed to host a single web page accessible by the root path. A sequential web crawler visited the teams' web pages once a minute in random order. In order to be indexed, web servers needed to respond to requests within one second, and responses over 10KB were ignored.

Once a page was crawled, keywords were extracted from *title*, *h1* and *p* HTML tags, and a base score was assigned to each keyword based on the number of times the particular keyword appeared in the text relative to the total number of keywords. For instance, in the text, "the quick brown fox jumps over the lazy ground hog" there are a total of ten keywords with *the* appearing twice thus having a density of 0.2. All other keywords have a density of 0.1.

To prevent teams from using naïve techniques, such as having a page with only a single keyword or alternatively containing every word in the dictionary, only keywords with densities between 0.01 and 0.03 were assigned base scores. However, keywords appearing in either the *title* or *h1* tags were guaranteed a base score of at least 0.008. In addition to the base score, a bonus multiplier was applied to keywords appearing in the *title* or *h1* tags according to a linearly decreasing function that favored sooner appearing keywords. For example, in the title "iCTF 2009 was Super Awesome!" the keyword *iCTF* would receive a 0.3 fraction of the *title* multiplier and the remaining words would respectively receive a 0.25, 0.2, 0.15, and 0.1 fraction of the multiplier.

On the other end of this system was the *Goollable* search site. *Goollable* was accessible both by the simulated users and to each team through a standard HTML interface. The simulated users performed single keyword searches to determine which team's web server to visit and the teams accessed Goollable to see their relative search ranking for particular keywords. When designing this system, it was our hope that teams would reverse-engineer the scoring function in attempt to achieve the maximum possible score for desired keywords. Our hope, however, fell short and we decided to release the crawler and search engine source code midway through the competition.[5]

4.2 The Vulnerable Browsers

The overall goal of a team was to compromise as many users as possible. Users had to be lured to a web site under the control of the attacker that would

[5] The crawler and search engine source code is available at `http://www.cs.ucsb.edu/~bboe/public/ictf09/search_engine.tar.gz`

deliver a drive-by-download attack, in a way similar to what happens with attack "campaigns" in the wild. In the following, we describe the characteristics of the various browsers that were used by the simulated users in the competition.

Operla. As the name suggests, *Operla* was a browser written in Perl that relies on `libwwwperl` to perform the necessary HTTP communication. Operla supports a minimal form of the HTML `object` tag, and introduces a so-called *Remote-Cookie-Store*. Three versions of Operla were deployed incrementally during the competition each of which contained unique vulnerabilities.

The first version of the Operla implemented a *Remote-Cookie-Store*. This feature was designed to upload a copy of the users' cookies to a remote location. While the attacker could choose an arbitrary URL to upload to, Operla would perform this action only if an associated security header contained the correct password. Operla stored an MD5-sum of the password, thus by determining the plain text password associated with the MD5-sum, an attacker could trigger the upload of the cookies. Since the MD5-sum was stored without a salt, searching for this value on the web was enough to retrieve the required password.

The second version of Operla contained a vulnerability that mimics the real-world vulnerability of the Sina DLoader ActiveX component [9]. This component allowed an attacker to download and install an arbitrary file from the Internet on the victims' computers. Operla, by incorrectly validating the parameters for HTML object tags, suffered from a similar vulnerability.

The final version of Operla contained a remote code execution vulnerability. To exploit the vulnerability an attacker had to perform the following steps. First, two HTTP headers needed to be sent back to the browser. One contained the code that should be executed upon successful exploitation and the other contained an arbitrary URL. To this response, Operla created a challenge string consisting of ten random characters and transmitted them in a request to the arbitrary URL. Second, the attacker needed to respond to the request with a JPEG image consisting of a visual representation of the challenge string and containing the MD5-sum of the challenge string in the image's EXIF header. If the image was configured properly Operla executed the attacker-provided code.

Jecko. Jecko was a vulnerable browser written in the Java language. During the competition, we provided the teams with three versions of Jecko, each containing a different vulnerability. All versions were distributed in bytecode format, which was trivial to convert to source code by using a Java decompiler, such as JAD.

The first vulnerability was a command injection in the code that handles the HTML `applet` tag. When Jecko parses an `applet` tag, it retrieves the code base specified by the `code` attribute, saves it on the local disk, and executes it by spawning a system shell, which, in turn, invokes the Java interpreter using a restrictive security policy. In addition, Jecko allows pages to specify a custom `mx` attribute to set the maximum memory available to the "applet" program. The value, however, is used unsanitized in the shell invocation. Students had to identify the command injection vulnerability in Jecko's source code, and attack it by injecting arbitrary shell commands, which would then be executed with the privileges of the user running the browser.

The second vulnerability consisted of exposing untrusted pages to insecure plugins. In Jecko, plugins are implemented as C libraries that are loaded by the Jecko through the JNI framework. Pages can instantiate plugins by using the `object` tag. Jecko was provided with two plugins, one of which exposed a function that allows pages to download the resource at a given URL and execute it. This vulnerability mimics several real-world vulnerabilities, such as CVE-2008-2463. Exploiting this vulnerability required the attacker to understand Jecko's plugin mechanism and reverse-engineer the provided binary plugins.

Finally, the last vulnerability consisted of an authentication flaw in Jecko's upgrade system. When Jecko parses a page containing the custom `XBUGPROTECTION` tag, it assumes it is visiting a site that provides updates for the browser. Then, the URL specified by this tag should contain a serialized Java object that specifies the commands required for the updates. These commands are encrypted with a key transmitted as part of a custom HTTP header. The attacker had to reverse-engineer this upgrade mechanism and discover that the update commands are not signed. To launch an actual exploit, teams had to create serialized versions of the update object and configure their pages to respond with the appropriate custom HTTP header.

Erbrawser. Erbrawser was a browser written in the Erlang programming language and was composed of a main module that implemented the user interface, performed queries via the standard Erlang `http` library, and parsed the HTML responses via the *mochiweb* toolkit. The vulnerability for this browser was designed to be simple to detect and exploit. The main challenge was becoming familiar with this functional language and producing the correct code to inject into Erbrawser. Erbrawser was deployed in two versions containing a similar vulnerability, although the second version of the browser was more difficult to exploit. The source code for both browsers was given to the teams thus making it somewhat simple to discover the exploit in the first version.

Erbrawser introduced the `<script type="text/erlangscript">` tag that executed its content in the Erlang interpreter without sanitization or sandboxing. This feature contained a number of vulnerabilities that allowed the execution of arbitrary third-party code inside a user process. The first version of Erbrawser allowed the execution of any Erlang commands, thus the easiest way to exploit was to execute shell commands through `os::cmd()`. The second version of Erbrawser forbid the execution of `os::cmd()` and similar functions. Nonetheless, having direct access to Erlang's interpreter gave the attacker the ability to use the browser as a bot by spawning an Erlang thread inside the browser. This thread could then continuously read and submit flags to the *Mothership*.

Pyrefox. Pyrefox was a browser written in Python. The browser had two versions each with a vulnerability. The first version of Pyrefox used the *path* attribute of a cookie to determine the name of the file in which to store the cookie's contents. Therefore, one could use a path traversal attack to overwrite or create any file accessible to the browser's user. The second version of Pyrefox fixed that vulnerability but introduced a code injection vulnerability. This vulnerability was associated with the fictitious scripting language, called JavaScrapt,

supported by the Pyrefox browser. The JavaScrapt language allowed a modification to the page by specifying an XML Path-like expression to a new string. However, the path and the string were passed unsanitized to an `eval()` statement, allowing for the execution of arbitrary Python code.

Crefox. Crefox was a browser written in the C programming language requiring `libcurl` and `htmlcxx` for HTTP communications and HTML parsing respectively. Crefox came in three versions, all with vulnerabilities that eventually allowed an attacker to execute arbitrary code supplied as part of the pages retrieved by the browser. We voluntarily leaked all the versions of Crefox source code throughout the competition to allow the teams to focus on exploiting the vulnerabilties rather than reversing the binaries.

The first version of Crefox had a NULL pointer de-reference vulnerability in addition to a `mmap` call that mapped the downloaded HTML page at the virtual memory address 0. Thus, triggering the vulnerable code path required the attacker to serve a page starting with the special string `USESAFEPRINTFUNCTIONA`, followed by the code the attacker wanted to execute. When the special string appeared, Crefox attempted to call a nonexistent function thus executing the attacker's code. The second version of the browser fixed the previous vulnerability, but introduced a format string vulnerability triggered when the downloaded page was about to be printed. To make this vulnerability difficult to detect, the `printf` function name was encoded and subsequently retrieved at run-time.

Finally, the third version of Crefox fixed the previous vulnerability, but introduced two new ones, namely a plain stack-based buffer overflow, and a non-control data buffer overflow leading to a command injection. Ironically, the former was not intended at all, but was the result of a typo that switched the destination buffer of a string copy routine from a global variable to one residing on the stack. To trigger the command injection vulnerability, the attacker had to embed a new HTML `ictf` tag with a `code` attribute containing a specific pattern followed by the shell command to inject.

4.3 Overview of the Live Exercise

The 2009 iCTF took place on December 4, 2009 between 8 am and 5 pm, PST. The participating teams connected to the competition VPN during the preceding week. The teams received an encrypted archive that contained a presentation describing the complex setup of this competition. The presentation was supposed to include audio that described the various steps of the competition, but unfortunately the audio was not included, due to technical problems. This generated some confusion in the initial phase of the game. We eventually gave teams the audio portion of the instructions and the game could start.

The first problem we ran into was the poor performance of the entire system. Various components of the infrastructure were only lightly tested and under the full weight of hundreds of participants, they slowed to a crawl. We traced this issue to the fact that most components were not multi-threaded and therefore they could not respond to the volume of requests being made. We solved the

issue by adding some standard threading code and while this operation easy to do, it still required some time, causing further delays to the game.

In addition to the scoring infrastructure woes, we also had to deal with problems with the simulated users. The aforementioned threading issues affected the simulated user scripts causing them to misbehave and not visit teams' websites as often as they should have. There were also some browser specific issues, for example the Java based Jecko browser would periodically run out of memory causing our simulated user processes to terminate unexpectedly. This issue went mostly unresolved and as a consequence there were not many opportunities for teams to exploit this particular browser.

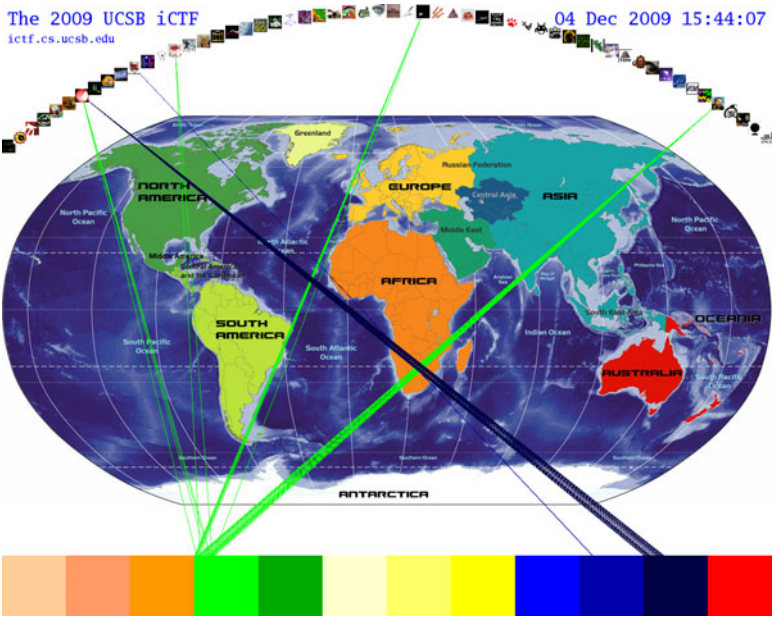

Fig. 2. The 2009 iCTF Scoreboard. Each pixel on the bottom of the screen represents one of the 1,024 users, each colored according to the browser they used. A line from a user to a team indicates that the user was "owned" by the team.

While we were sorting out these issues, a team managed to discover a flaw in the banking system. The goal of the banking system was to have people withdraw money from our simulated users and add it to their own accounts. One enterprising team discovered a flaw in our validation routines, in that we allowed for negative amounts. In effect, this allowed them to drain the accounts of other teams by simply making a transfer request from their own account to the victims account, with a negative amount. They were gracious enough to alert us and let us patch the flaw, instead of wrecking havoc throughout the scoring system.

Even though the competition had a slow beginning, the teams started to figure out how to exploit the competition's complex system of interconnected components and eventually users were compromised and made part of a botnet. Figure 2 shows the graphic format that was used, during the competition, to show which users were "owned" by which team.

By the end of the competition, the team CInsects managed to capture a significant portion of the simulated users that visited their site and took first place.

4.4 Feedback

After the 2009 competition was completed, a poll was given out to the point of contact for each team. Each person was asked to rank various aspects of this competition. Of the 56 participants we received 35 responses.

The most basic question we wanted to answer was in regards to the size of the competition. Every year we see more and more teams participating, however we did not ask about how many individuals each team had. This year we wanted to get a more accurate number on how many people were participating. From our respondents we learned that, on average, each team was composed of 15 players. Although it is unfortunate that we cannot give an exact number because we did not get feedback from every team, we can provide a very reasonable estimate that this competition involved well over 800 individuals from all over the world, supporting our claim of running the largest security competitions.

Table 1. Botnet Gameplay Feedback

Category	No Response	Awful	Satisfactory	Awesome
Playability	2	4	23	6
Score System	1	8	20	6
Novelty	1	2	12	20
Network Setup	1	4	16	14
Challenges	1	4	15	15

From these responses as outlined in Table 1 and the feedback received, it is clear that while most teams enjoyed the competition there were numerous aspects that needed improvement. The most common complaint was that the mechanics of this competition were not entirely clear at the start and it took teams a significant amount of time just to understand what they were being asked to accomplish. However, the overwhelming majority of teams agreed that it was a novel competition and echoed our sentiments in that creating a new competition leveled the playing field. It is also interesting to note that not many people complained about the network connectivity, even though we were having issues with the simulated users connecting to teams early on in the competition. We suspect that our network issues may have been masked by the fact that the teams were taking a significant amount of time to understand the competition.

Table 2. Botnet Team Feedback

Category	Yes	No
First Time Playing	12	23
Developed Tools	18	17
Gained Skill	33	2
Educational	33	2

Table 3. Network Trace Characteristics

Year	Duration (hh:mm)	Data Size (MB)	Packets (10^6)	Data Rate (bytes/s)
2003	3:19	2,215	6.96	188,941
2004	0:54	258	2.78	121,680
2005	7:27	12,239	30.14	1,427,060
2007	6:45	37,495	92.57	2,065,115
2008	41:29	5,631	34.11	60,179
2009	17:51	13,052	40.58	550,408

By the time the teams were actually ready to receive traffic, we had resolved the network issues.

One of the primary goals for running these security competitions is that we want participants to go beyond the normal course work and explore new and original ways to accomplish these tasks. In order to check if this goal was being met, we asked how many teams had developed tools specifically for this competition. As displayed in Table 2, half of our respondents stated they had worked on tools specifically for this competition, giving us empirical evidence that these competitions are working as intended. Furthermore, we also collected evidence supporting the continuing popularity of these competitions, as more than two thirds of our respondents were veterans who had played in previous CTFs.

5 Lessons Learned

Throughout the seven years (and eight editions) of the iCTF we learned (the hard way) a number of lessons.

An important lesson that we learned very early is that the scoring system is the most critical component of the competition. A scoring system must be *fair* and *objective*. Given the size of the competitions we ran, this means that it also needs to be *automated*, that is, it cannot rely on human input. Even though we learned this lesson many years ago, our scoring system failed catastrophically during the 2008 competition. In this case, the lack of testing forced us to switch to manual evaluation, with very unsatisfactory results. Fortunately, because of its design, the competition had a clear winner, which was determined in an objective way. However, we cannot say the same about the ranking of the rest of the teams.

A second lesson learned is that the critical events should be *repeatable*. That is, all the events that cause a change in the score of a team should be logged, so that if a bug is found in the scoring mechanisms, it is possible to handle the failure and restore the correct scores of all teams. The 2009 scoring system did not include a manual component, but suffered from a number of glitches, mostly due to erroneous database programming. Fortunately, all transactions were logged and, therefore, it was possible to troubleshoot the problems and restore the correct scores.

A third lesson learned about the scoring procedure is that a scoring system should be *easily understood* by everyone involved. This helps because, on one hand, the participants will understand what is strategically important and what is not, and, in addition, they can identify errors in the scoring process and help the organizers fix them. On the other hand, if the system is well-understood by the organizers, it is easy for them to fix problems on-the-fly, which is often necessary, given the time-critical nature of the competition.

As previously mentioned, our scoring system, as well as network traces can be found at `http://ictf.cs.ucsb.edu/`. Table 3 summarizes the network traces captured for the past iCTFs.

While these lessons described above have been learned throughout all the editions of the iCTF, there are several lessons that are specific to the 2008 and 2009 editions.

A first lesson is that attack-only competitions like the 2008 and 2009 iCTFs are valuable. When the iCTF was first conceived, it was to be the final test for students finishing a graduate-level computer security course. The goal was to provide a one-of-a-kind hands-on learning experience for teams of novice security experts. Since then, several annual CTFs have emerged in addition to the iCTF, and there are quite a few teams that regularly participate in these competitions. As such, these teams have gained significant experience and are quite organized. Even though the goal of the competition is to foster the development of novel tools and approaches, the fact that some teams come very prepared can make the competition too hard for newer, inexperienced teams. By having an attack-only competition, where the teams compete side-by-side and not directly against each other, it is possible to shield newcomers from "veteran" teams. Of course, a major drawback of attack-only competitions is that the defense skills of the students are not tested.

New participants were also aided by the new nature of the competition. In fact, by changing radically the style of the competition, we managed to somewhat level the playing field. Although the winning teams were still experienced groups, in both the 2008 and 2009 editions, teams of first-time competitors placed quite high in the ranking. This was possible because we intentionally did not disclose in advance to the teams the nature of these new competitions. In both cases, many "veteran" teams expected a standard CTF and were surprised to learn that this was not the case. Of course, it is hard to keep surprising teams, as designing new competitions requires a substantial amount of work. However, it is arguable that this type of competition is inherently easier for novice teams to participate in.

Another important lesson is that too much novelty can hurt the overall competition. Although the 2008 competition was a radical departure from the traditional iCTF, the task was fairly straightforward: break into a network. With the 2009 iCTF, the competition structure was much more complicated. Not only did teams have to reverse-engineer the browser software they also had to perform Search Engine Optimization to get users to visit their sites. Moreover, once they understood how to capture users, to score points they had to figure out

how the banking system worked, as well as how the botnet *Mothership* could be used to generate more points. In total, there were three different kinds of points, with a fairly complex relationship between them, which many participants found sometimes confusing.

A final thought about these kind of competitions is: *are they worth the effort?* Preparing all the editions of the iCTF and, in particular, the two completely new iCTF structures of 2008 and 2009 took an enormous amount of time and resources. Therefore, it is understandable to wonder what are the benefits. We think that the fact that after the iCTF was introduced many other similar events started surfacing shows that there is interest and perceived value for these events. We really do think that competition fosters group work and creative thinking, as witnessed by the feedback we gathered for the 2009 iCTF, and we think that live exercises are a useful tool to support the security education of students. Also, these types of competitions help the organizing team, because they provide visibility and publicity to their institution. For example, this year a number of the applications to the graduate program of the Department of Computer Science at UCSB mentioned the iCTF.

6 Conclusions

Security competitions and challenges are a powerful educational tool to teach hands-on security. However, the design, implementation, and execution of complex, large-scale competitions require a substantial amount of effort. In this paper we described two novel designs that were implemented as large-scale security live educational exercises. These exercises involved several hundred students from dozens of educational institutions spread across the world. The information that we provided about the software/hardware infrastructure supporting the competitions, as well as the lessons learned in running these events can be useful for educators who want to create similar competitions.

Acknowledgements

This work has been supported by the National Science Foundation, under grants CCR-0524853, CCR-0716753, CCR-0820907, and by the ARO under grant W911NF-09-1-0553.

References

1. Augustine, T., Dodge, R.: Cyber Defense Exercise: Meeting Learning Objectives thru Competition. In: Proceedings of the Colloquium for Information Systems Security Education, CISSE (2006)
2. ComputerMajors.com: Computer Science Degrees: Starting Salaries (June 2009), `http://www.computermajors.com/starting-salaries-for-computer-science-grads`

3. Cowan, C., Arnold, S., Beattie, S., Wright, C., Viega, J.: Defcon Capture the Flag: defending vulnerable code from intense attack. In: Proceedings of the DARPA Information Survivability Conference and Exposition (April 2003)
4. Group, T.H.: The ructf challenge (2009), http://www.ructf.org
5. Mullins, B., Lacey, T., Mills, R., Trechter, J., Bass, S.: How the Cyber Defense Exercise Shaped an Information-Assurance Curriculum. IEEE Security & Privacy 5(5) (2007)
6. Pimenidis, L.: Cipher: capture the flag (2008), http://www.cipher-ctf.org/
7. Pwn2own 2009 at cansecwest (March 2009), http://dvlabs.tippingpoint.com/blog/2009/02/25/pwn2own-2009
8. Schepens, W., Ragsdale, D., Surdu, J.: The Cyber Defense Exercise: An Evaluation of the Effectiveness of Information Assurance Education. Black Hat Federal (2003)
9. SecurityFocus: Sina DLoader Class ActiveX Control 'DonwloadAndInstall' Method Arbitrary File Download Vulnerability, http://www.securityfocus.com/bid/30223/info
10. Vigna, G.: Teaching Hands-On Network Security: Testbeds and Live Exercises. Journal of Information Warfare 3(2), 8–25 (2003)
11. Vigna, G.: Teaching Network Security Through Live Exercises. In: Irvine, C., Armstrong, H. (eds.) Proceedings of the Third Annual World Conference on Information Security Education (WISE 3), June 2003, pp. 3–18. Kluwer Academic Publishers, Monterey (2003)