Facilitating a Battle Between Hackers: Computer Security Outside of the Classroom

Nathan Backman
Buena Vista University
Storm Lake, Iowa, USA
backman@bvu.edu

ABSTRACT

Attack/Defend computer security contests require participants to leverage knowledge obtained from a variety of courses across a computer science curriculum, providing undergraduates with a novel and exciting opportunity to challenge both themselves and their peers. However, there are limited opportunities to participate in such contests and none of them is well suited for novices in computer security.

This paper describes the design of an Attack/Defend security contest that is geared towards undergraduates who have little exposure to computer security and require a more gentle introduction. We provide implementation details of the framework that supports this contest and offer lessons learned over the past three years in growing our contest into an intercollegiate event that is deployed on a cloud infrastructure to support multiple, concurrently operating contest sites which span three timezones.

Keywords

application security; hacking; capture the flag

1. INTRODUCTION

It is difficult to overstate the need for producing security-minded software developers. Addressing security issues late down the software development pipeline as an afterthought may lead to compromised systems and data. Software developers must practice defensive programming and be aware of the risks of the programming languages, libraries, frameworks, and systems they use. Additionally, researchers have learned that a firm grasp in offensive security[11] is an important step in better understanding the challenges inherent to computer security.

The need to further emphasize computer security in undergraduate programs was acknowledged in the ACM/IEEE Computer Science 2013 Curricula[10] through the inclusion of a new Knowledge Area named: *Information Assurance and Security.* Another way to support the development of computer security skills is by encouraging students to participate in extracurricular events and contests that feature these concepts – much like how the ACM International Collegiate Programming Contest[1] provides an opportunity for students to develop programming and algorithmic abilities.

SIGCSE '16, March 2–5, 2016, Memphis, TN, USA
© 2016 ACM. ISBN 978-1-4503-3685-7/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2839509.2844648

1.1 Capture The Flag

Computer security contests are not a new phenomenon and have existed in at least one popular form (Capture the Flag contests) since formalized in 1996 at the annual DEF CON[2] hacker convention. Capture The Flag (CTF) contests provide participants with the opportunity to identify and exploit vulnerabilities hidden within network-based applications (called *services*). Event organizers create custom services for these contests and purposely engineer into them at least one critical vulnerability. Successfully exploiting a vulnerability in a service will give an attacking team the opportunity to gain escalated privileges on the compromised service in order to acquire sensitive data (called a *flag*) stored within the service or on the computer. Teams report captured flags (typically represented as an ASCII string) back to a flag submission interface to redeem points. The flag will be validated for authenticity and teams will be rewarded with points for each flag they submit.

There are two main types of CTF contests: Jeopardy and Attack/Defend. In a Jeopardy contest, teams compete to exploit services hosted on computers that are controlled by the contest administrators. Contestants play only on the offensive and the administrators need only host a single set of services. Teams then compete to exploit as many services as they can as quickly as possible. In an Attack/Defend contest, each team is given control of their *own* computer and all of these computers are hosting an identical set of contest services. This places teams in a position to both defend the services on their own computer and attack the services hosted on the computers belonging to the other teams. To do this, teams will analyze the services (and their source code) to identify vulnerabilities that can lead to flag theft. Upon finding such a vulnerability, a team will fix (patch) that vulnerability in their own service and develop an attack to exploit that vulnerability in the services hosted on all of the other teams' computers. A successful exploit will result in the theft of flags which can be redeemed for points.

Attack-and-defend CTF contests offer a more rich experience by making teams look at application security comprehensively. Being able to identify vulnerabilities, understand the process of developing exploits, and patch these vulnerabilities are all incredibly important in the grand scheme of computer security. Understandably, facilitating an Attack/Defend CTF contest is more challenging that facilitating a Jeopardy CTF contest. The additional challenges include: 1) provisioning a computer (typically a virtual machine) for each team; 2) managing services, flags, and infrastructure support code on all computers; and 3) the need to incentivize teams to keep their services online and functional. Due to these additional development and production costs, Attack/Defend contests are quite a bit more rare than Jeopardy contests.

1.2 Related Work

To our knowledge, and at the time of writing, there are only two regularly occurring Attack/Defend CTF contests that are hosted from within the USA. They are: 1) the previously mentioned DEF CON CTF contest which is hosted by industry professionals in Las Vegas, draws in competitors of the highest caliber, and requires qualification by placing highly in one of a set of other CTF events (which have traditionally been Jeopardy contests); and 2) the iCTF[12] contest produced by academics from the University of California, Santa Barbara which is open only to educational institutions. iCTF is an internet-accessible event that has shown the capacity to host over a hundred teams with thousands of students in a single event. In their most recent event in 2015 [3], teams were required to contribute their own exploitable service to the contest in order to be eligible for participation. This would prove to be a barrier to entry for any participants who are novices in the area of computer security. Finally, there is an additional internet-accessible event hosted in Russia (RuCTFE[4]) that serves hundreds of teams that face a challenging set of services. In addition to these events, there are a small handful of other Attack/Defend CTF contests hosted in Europe and Asia that require a physical presence for participation. A managed list of regular occurring CTF events can be found at the CTFtime website[5].

EDURange[8] is a platform created to quickly deploy lab-based computer security exercises into the cloud with ease. Our CTF framework shares commonalities with this project in how we deploy exploitable services and labs as security exercises. For example, any one of our services could be easily deployed with EDURange to be used as a classroom exercise. With slight modifications, it could be possible to use EDURange to fully deploy an Attack/Defend CTF contest operating on our framework.

1.3 Outline

The remainder of the paper is organized as follows. Section 2 presents the motivation for our contest and describes the expansion of the event over the last three years. Section 3 describes the general format and design of our CTF contest. Section 4 provides an overview of contest services and how they must be structured for inclusion within our framework. Section 5 details the implementation of our framework. And, finally, Section 6 contains our conclusions, reflections, and areas for future work.

2. CONTEST DEVELOPMENT

As advocates for security, we encouraged our students to participate in Attack/Defend CTF contests to gain exposure to the discipline. Coming from a small, private liberal arts college, however, our students do not have a wealth of computer security classes with which to prepare themselves for such contests. In fact, prior to the production of our first CTF event we had never previously offered a course in computer security. This is not unusual for small, liberal arts colleges. Students originating from such colleges tend to have a more introductory skill set for computer security.

The few Attack/Defend CTF contests that exist tend to target and attract teams of an extremely high caliber. The problem sets (contest-provided exploitable services) can therefore be extremely challenging and often simply inaccessible to the security novice. There are not a wealth of these types of contests simply because the development and production costs of facilitating them are incredibly high. Therefore, those that exist tend to be facilitated by security professionals and computer security researchers who cater to advanced participants.

Seeing a need for a beginner's introduction to Attack/Defend CTF contests, we decided to produce such an event for our students. As educators, this prospect appealed to us because it provided an opportunity to integrate a large body of knowledge from our computer science curriculum into the contest-provided services. A collection of services can employ concepts from classes including: introductory programming, algorithms, operating systems, networks, database management systems, web application development, computer architecture, and potentially many more.

The integration of concepts from our curriculum into this extracurricular event provided our students with the unique opportunity to challenge their familiarity with the discipline of computer science as a whole within the context of software security. Students would look forward to subsequent contests as they matured through our computer science program to, loosely, measure themselves and to challenge both themselves and their peers.

2.1 Contest Expansion

We hosted our debut Attack/Defend CTF contest in 2013 on our campus at Buena Vista University (located in Storm Lake, Iowa) and accommodated 12 of our students arranged within 4 teams. Feedback from students was positive and they encouraged us to make this an annual event.

In 2014 we invited Whitworth University (located in Spokane, Washington) to participate with us remotely. The event included 39 students distributed across 10 teams. To accommodate both universities, our event was hosted on the Amazon Elastic Compute Cloud[6]. Web cams, microphones, and projectors were situated in the two separate contest sites to maintain a live audio/visual connection between the two universities throughout the duration of the contest. This created a "portal" between the two contest sites which allowed for an exciting environment and enabled friendly banter, taunts, and interactions between teams across sites. Again, feedback was positive and Whitworth University was receptive to being included in future events.

In 2015 we expanded yet again by inviting Taylor University (located in Upland, Indiana) to compete as a third remote site. We also invited students from nearby Northwestern College (Orange City, IA) to come compete at the Buena Vista University site. Ultimately there were 4 universities represented situated across 3 contest sites each in a different timezone and all sites again connected via web cams. Collectively, there were 70 student participants distributed across 14 teams. This event featured a bonus opportunity which allowed teams to "hack" into a black-box room accessible via a web interface in which lights, cameras, and USB missile launchers could be utilized to meet team-specific objectives. Again, feedback was positive and each university expressed an interest to return for future events in spite of a couple of hiccups that we faced in the administration of this event.

It is important to note that the other schools we invited to participate are also small, primarily undergraduate institutions without a wealth of security classes. At most, these institutions offered only an introductory computer security class (and this was often not the case). In fact only one of the participating institution offered a security class at the time of their first involvement in our contest and two universities have begun offering an introductory computer security course since their first involvement.

3. CONTEST FORMAT

This section presents the structure and format of our contest with notes on how the event has changed in successive iterations.

3.1 Team Composition

Some readers may have noticed that our participant count has not risen proportionally to our team count over the past three years. Due to the nature of the Attack/Defend contest, as the team count increases so does the work required in delivering exploits to other teams. This extra work can be circumvented through the automation of exploits but, frankly, many of our teams have not mastered the art of automation – especially the beginners who we are trying to attract. In an attempt to keep the team count from exploding, our most recent event has allowed for teams to participate with 4 to 6 students. This number of students per team seems manageable, not requiring extreme measures for coordination.

From the outset, this contest was designed to be a learning experience instead of a quick exhibition of skills like you might see in the ACM ICPC. With this in mind, each team was *required* to include a student that had not yet completed their first year of study in computer science. An initial and seemingly intuitive reaction would be that first-year participants would be completely out of their league in such a contest. Actually, and without exception, each first-year participant has told us this – that they were completely in over their head. However, when asked if they would recommend that future first-year students also participate, they, again without exception, recommended that they should.

We have found that including first-year students is a great way to get them connected with more senior students while building camaraderie within the major. These senior students have usually done a great job in finding ways for the first-year students to participate while offering insights along the way. It might be suggested that the addition of first-year students to a team may slow them down. While this is certainly possible, we again emphasize that our objective was to facilitate a learning experience rather than just an exhibition of skills. Also, due to the length of the contest, additional person-hours of support can be considered invaluable to assist in some tasks which are difficult to automate – launching exploits and submitting flags, for example.

3.2 Round-based Format

Our CTF contests takes place over a series of 1-hour rounds. At the beginning of each round, the contest framework will deliver new flags to each service on each computer. For example, if there are 4 teams and each team is hosting 6 contest services, 24 flags will be distributed at the beginning of the round with each computer receiving 6 flags (one for each service).

A team will only be rewarded points for capturing a specific flag once. Therefore, in a single round, with t teams and f flags, a team will be able to capture at most $(t-1)f$ flags per round. The fact that new flags are distributed at the beginning of subsequent rounds is very important. It allows teams to launch their exploits yet again to capture the newly distributed flags. The expectation of being hit with a fresh wave of exploits at the beginning of each round encourages teams to keep searching for ways to patch their services. Similarly the round-based format encourages teams to search quickly for vulnerabilities to develop exploits before the flags for the current round are replaced with new ones.

3.3 Contest Length

We have struggled to determine an appropriate length for our CTF contests. As the event is intended to be a learning experience, we want students to have plenty of time to investigate the services, research types of vulnerabilities, and develop exploits.

Our first contest had a length of 24 hours and was designed such that students could come and go as they pleased – the expectation was that students would take time off to eat and sleep. Well, our undergrads are stubborn and refused to heed suggestions to try to automate their exploits over the night. Instead, they were thrilled with the option of staying up through the night (or at least taking turns sleeping) to collect flags. In the following year, the contest kept its 24-hour time span but the contest was suspended for 8 hours at night to force students to take time off and sleep.

In our most recent contest, the invitation of a neighboring university to compete locally at our site resulted in shortening the contest. The new time format allowed for 4 hours of competition followed by a 1-hour break and then another 4 hours of competition. This format seemed to help to keep participants focused and adequately rejuvenated throughout the contest.

3.4 Scoring

Teams can score points in one of three ways.

1. **Capturing Flags**: By exploiting a service, a team will have an opportunity to steal a hidden flag. The flag can be redeemed for points through a flag submission interface.

2. **Keeping Flags**: At the end of a round, teams will be rewarded with defense points for keeping flags secure from other teams. For example, if a team is hosting a particular service that was exploited by only 2 of 10 other teams in a round, then they would receive defense points for each of the 8 teams that did *not* steal that flag in the round. A team that was exploited by all 10 other teams on a particular service would receive no defense points for that service.

3. **Maintaining Service Uptime**: In order to prevent teams from simply shutting down their services to prevent flag theft, we reward teams with points for service

uptime. Determining whether a service is up and functional is addressed in Section 4.4. All services on all team computers are checked to see whether they are functioning appropriately every 3 to 5 minutes over the duration of the contest. If a service is found to be functioning appropriately during an uptime check then the team hosting that service will receive uptime points.

Flag capture points and uptime points parallel the scoring mechanics of other Attack/Defense contests. We added points for keeping flags to reward teams that have done well patching services. We observed that some teams were able to patch many more services than they were able to exploit but they were not rewarded for their defensive capabilities. We now try to balance this dynamic by rewarding offensive as well as defensive successes.

4. CONTEST SERVICES

Services are developed for the contest with a couple of things in mind. We want to feature a broad range of programming languages, frameworks, and technologies and have students draw from their knowledge of a variety of concepts covered in the computer science curriculum. In the past, we have produced services that have been written using languages including: Python, Java, C, C++, Javascript, PHP, SQL, and Bash. Types of services we have created include: chat clients, social media sites, meme generators, file sharing programs, games, web servers, key-value data stores, and various other types of web applications and remote administrative tools.

When creating services for our first event, some of the services were really quite simple in size and scope. Such simple services did not leave a lot of room for creativity when it came to searching for vulnerabilities. That is, there were relatively few possible attack vectors worth investigating. Expanding on the size and scope of the services will ultimately require teams to do more investigative work to uncover vulnerabilities.

A recent example of a service we created is: HackerChat. This service was written in C and contained both client-side and server-side binaries. Clients would establish a TCP connection with a server and be presented with an IRC-like chat room and could chat with other clients connected to the server. This service could be exploited via buffer overflow. Overflowing the buffer containing the user's away message could overwrite the path the server maintained to a message-of-the-day file. Replacing this path with the path to the flag file could then enable the attacker to fetch the flag by prompting the server to print the contents of (what it believed to be) the message of the day file.

4.1 Vulnerabilities

Often, services are designed with a couple of vulnerabilities in mind. For example a content management system might be designed with a SQL injection vulnerability in mind or a C application might be designed to be vulnerable to a format string exploit. We typically try to include a wide range of vulnerability types across the set of services. Also, each service *should* ideally have more than just one vulnerability which can lead to flag theft. This can lead teams to wonder if they have found and patched all of the vulnerabilities within their services.

For the contest administrator, it can be exciting to see a team integrate a patch and thereby block a currently exploited vulnerability only to be exploited at a later time on that same service through a previously unknown vulnerability. We do not want students to rest easy just because they have found and patched a single vulnerability within a service. The potential for multiple vulnerabilities should keep teams on their toes and searching for extra vulnerabilities in the event that some of their previously developed exploits become blocked.

4.2 Flag Locations and Service Permissions

Each service must be written so that the vulnerabilities built into it can lead to the theft of its flag. All flags are stored in known locations. For example, if we have Services A, B, and C, then the following set of files would exist on each team's computer: `{/etc/flags/A, /etc/flags/B, /etc/flags/C}`. File permissions are set such that only the services to be exploited will have read access to their flag file. It is only by exploiting a service that you can leverage its permissions to gain read access to its flag file. Also, teams will never gain write-access to flag files – not even their own. This prevents teams from corrupting flags so that they fail validation when other teams capture and submit them.

We also allow flags to be stored in different ways as is appropriate for each particular service. For example, a content management system may store its flag in its back-end database under a field applicable to the site administrator. Then, perhaps via privilege escalation to administrator status or perhaps via SQL injection, an attacker may be able to fetch the flag from the database. In the event that a flag will be stored in such a service-specific way, then the location of that flag would be reported in the documentation for that service. Also, teams hosting services would have write access to these flags but we disincentivize teams from modifying their own flags with the threat of lost uptime points for corrupting service functionality.

In summary, teams should not be required to hunt for the flag – the locations of the flags will always be known – instead they must hunt for vulnerabilities that can let them gain access to a flag.

4.3 Minimizing Damage

We must be careful to restrict the kinds of havoc that an attacking team can inflict on the teams and services they exploit. For example, if an exploit allows one to execute arbitrary system commands on an opponents computer (such that `cat /etc/flags/A` would enable flag theft) then we want to prevent the attacking team from damaging or deleting other important service files. An exploit should not enable one team to impede the progress of another. Also, an exploit for one service should not be able to fetch flags from other service. Each service should be self-contained and vulnerable only to problems in its own code.

In our most recent contest, one team abused an exploit to delete all files relating to the attacked service on their opponents' VMs. This was ultimately a failure of the contest administrator and degraded the learning environment as well as the morale of the attacked teams. Their patches, progress, and exploits were repeatedly deleted. There is only so much an administrator can do to fix problems during the contest so every precaution must be taken to identify and remedy these issues prior to the start of an event.

4.4 Support Scripts

To integrate a service into the contest framework, a service author must also produce a series of scripts to support administrative functionality. These scripts are:

- **service-deploy**: A deployment script is used to install, configure, and start the service on a team's computer. Depending on the complexity of the service, this script may be as simple as copying files over to a working directory and executing a binary or involve something more complicated such as creating and initializing a MySQL database.
- **service-delete**: A service deletion script is used to remove all traces of the service from the computer. This will include deleting files, database, and terminating any running processes relevant to the service.
- **service-round-update**: This script must be written in order to facilitate any service's requirement to place a flag in a non-standard location. For example, this script may place the newly issued flag in a database if that would be appropriate for the service.
- **service-uptime-check**: This script is perhaps the most complicated of all. It reports back to the scoring system whether or not a running instance of the service is functioning appropriately. The definition of "functioning" here is ambiguous and left to the service writer to specify. Such a script will emulate standard and expected client behavior that the service is meant to facilitate. Should the service not respond in a way that the service author considers appropriate (for example by eliminating login functionality to prevent any form of privilege escalation – even when legitimate) then they will not receive uptime points. This script will report back one of three values: 1) **Online**: meaning that the service is deemed functional; 2) **Offline**: meaning that the service could not be contacted; and 3) **Malformed**: meaning that the service is active and responding but that it is not correctly facilitating legitimate user actions. This service status is displayed in a publicly available location for all teams to see after every uptime check. Teams can then take corrective measures when they notice that their service is not cooperating and they can also see which of the other teams' services are online and ready to be attacked.

The service-deploy and service-delete scripts are mainly used before the contest starts and after the contest ends. However, they also have a very important purpose during the contest. It is often the case that a team will somehow irreparably damage their own service – perhaps by deleting files, large portions of code, or making a change that they cannot undo. In such a scenario, their service status will likely go to "Offline" or "Malformed" and the team will quickly fall behind due to a loss of uptime points. We therefore provide a mechanism by which these scripts can be used to reset a service to a factory-default state.

5. CONTEST FRAMEWORK

We created our contest framework with the goal of facilitating a number of Linux virtual machines (VMs) to represent both the team computers and the administrative computer. In this model, we required only a single VM to be used for administrative purposes and one VM would be launched per team in the contest. Figure 1 provides a diagram of this architecture and we will elaborate on it in the following sections.

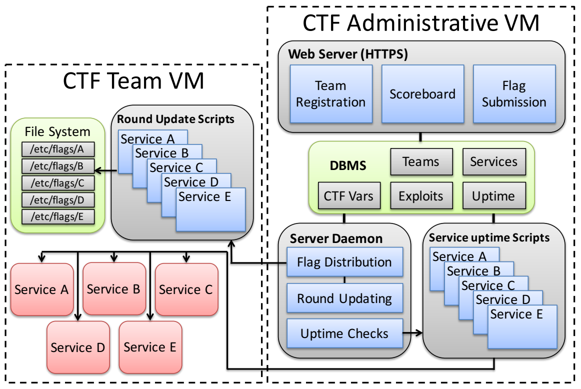

Figure 1: Architecture of the contest framework

5.1 Administrative VM

The administrative VM is responsible for hosting the contest website and for managing all updates to the state of the contest. The contest state is stored in a relational database and keeps track of team information (team name, password, VM IP address, etc.), service information (current flag, current uptime status, team ID, etc.), and a complete history of all scoring events (captures, defense points, uptime points). The database is relied upon heavily by both the contest website and the daemon that supports regular and ongoing administrative tasks.

The daemon triggers the beginning and end of the contest. Between these two points, the daemon had two additional tasks: 1) initiating new rounds; and 2) enforcing uptime checks. New rounds occur every hour on the hour. Once triggered it generates a new flag for every single service identified in the database and it then distributes those flags to the team VMs where those VMs would then locally run round-update scripts to integrate those flags. The uptime checks occur randomly once every three-to-five minutes throughout the duration of the contest. This variability was integrated to prevent teams from keeping their services off (and therefore immune to attack) except for when uptime checks were expected. The result of an uptime check is be immediately pushed to the database and assessed for points.

5.2 Team VMs

Team VMs are all spawned from the same image and contain all of the contest software. Prior to the start of the contest, we execute the service-deploy script to install and launch each of the services. With all services running, the team VM is only responsible for periodically executing round update scripts when commanded to do so by the administrative VM. This script integrates the new flags provided by the administrative VM. Occasionally these VMs would also be responsible for triggering the service-delete and service-deploy scripts when necessary to reset a service.

Teams otherwise have control over their team VM to investigate services, patch services, launch exploits, automate exploits, and do any additional development necessary. Team members connect to a VM its public, internet-accessible IP over ssh. All team members can then collectively work on the team's VM.

5.3 Contestant Web Interface

Teams interact with the game through a website hosted on the administrative VM. The website provides a basic scoreboard in which team standings are presented as well as a color-coded grid that depicts the uptime status of every service in the competition. Teams can quickly glance at this grid to determine when it was necessary to repair a service. We also graphed contest statistics over time to show flag accumulation as well as progression of defenses.

A nice and engaging feature of the website is that we designed a push-based mechanism to audibly inform teams immediately and verbally when flag captures occurred. On a flag capture, the speakers in the room would play a message akin to: "The Red Team has exploited Service B on the Green Team's server." This small feature was surprisingly effective at instantly changing the mood of the entire room!

In addition to reporting on the contest state, the website also featured a flag submission interface and a mechanism to ask for clarifications from the contest administrators. As the contest grew, it quickly became clear that disseminating important information and updates via web cam (aside from the opening briefing) was fraught with problems. We therefore implemented this clarification system to provide lasting records of posed questions similar to the system provided in the PC^2[7] programming contest platform.

6. CONCLUSIONS AND FUTURE WORK

We developed an Attack/Defend CTF contest and supporting framework to provide a more gentle introduction to the discipline for undergraduates who have had little exposure to computer security. Growing from 12 contestants, to 39, and then 70 has confirmed that there are other institutions that can benefit from this program and who are also interested in finding a way to encourage their undergraduates to seriously consider computer security.

To be frank, we have received a lot more interest from universities wishing to participate than we can currently handle under the umbrella of a single contest. Indeed, we face logistical challenges not faced by programs such as iCTF because of the fact that our demographic (our undergraduate teams) is less capable of scaling. If we were to permit the inclusion of well over 100 teams (as iCTF has done)[9] then our undergraduate teams would be burdened with carrying the logistical weight of exploiting a great deal more teams than they are likely capable of. Scaling in this way would shift our desired emphasis away from investigating vulnerabilities and developing exploits towards the menial work of applying developed exploits, recording and submitting flags, and attempting to automate scripts to do the same.

As we prepare for our next CTF contest in 2016, we plan to not introduce any additional sites or universities although we do expect the number of participants to increase due to growing exposure at those respective universities. Increasing the scale of our event beyond 16 teams or so will require a fundamental change to how the contest must be organized – and we flirted with that limit at our last contest. Before continued expansion we are considering implementing something akin to round-robin stages for the first half of the contest from which we can identify high performers and low performers that can be grouped together in subsequent stages. This would limit the number of targets each team must attack while grouping competitors of a similar skill level.

7. ACKNOWLEDGMENTS

We would like to thank our participating universities for their assistance and support in beta testing our framework and contest as we have expanded. This work was also supported by two Amazon AWS in Education Grants that funded deployment onto cloud services to support intercollegiate, cloud-based contests.

8. REFERENCES

[1] ACM International Collegiate Programming Contest, http://icpc.baylor.edu.

[2] DEF CON® Hacking Conference, A History of Capture The Flag at DEF CON, https://www.defcon.org/html/links/dc-ctf-history.html.

[3] The UCSB iCTF Competition, http://ictf.cs.ucsb.edu.

[4] RuCTFE, Online international challenge of information security, https://ructf.org/e.

[5] CTFtime.org: All about CTF (Capture the Flag), https://ctftime.org.

[6] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, Amazon.com, Inc.

[7] CSUS Programming Contest Control (PC^2), http://www.ecs.csus.edu/pc2/, California State University, Sacramento.

[8] S. Boesen, R. Weiss, J. Sullivan, M. E. Locasto, J. Mache, and E. Nilsen. Edurange: Meeting the pedagogical challenges of student participation in cybertraining environments. In *7th Workshop on Cyber Security Experimentation and Test (CSET 14)*, San Diego, CA, Aug. 2014. USENIX Association.

[9] N. Childers, B. Boe, L. Cavallaro, L. Cavedon, M. Cova, M. Egele, and G. Vigna. Organizing large scale hacking competitions. In *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'10, pages 132–152, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA, 2013.

[11] M. Mink and F. C. Freiling. Is attack better than defense?: Teaching information security the right way. In *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development*, InfoSecCD '06, pages 44–48, New York, NY, USA, 2006. ACM.

[12] G. Vigna, K. Borgolte, J. Corbetta, A. Doupé, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili. Ten years of ictf: The good, the bad, and the ugly. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, Aug. 2014. USENIX Association.