Ten Years of iCTF: The Good, The Bad, and The Ugly

Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupe,
Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili

(vigna,kevinbo,jacopo,adoupe,yanick,invernizzi,dhilung,yans)@cs.ucsb.edu
The SecLab Group
University of California in Santa Barbara

Abstract

Security competitions have become a popular way to foster security education by creating a competitive environment in which participants go beyond the effort usually required in traditional security courses. Live security competitions (also called "Capture The Flag," or CTF competitions) are particularly well-suited to support hands-on experience, as they usually have both an attack and a defense component. Unfortunately, because these competitions put several (possibly many) teams against one another, they are difficult to design, implement, and run. This paper presents a framework that is based on the lessons learned in running, for more than 10 years, the largest educational CTF in the world, called iCTF. The framework's goal is to provide educational institutions and other organizations with the ability to run customizable CTF competitions. The framework is open and leverages the security community for the creation of a corpus of educational security challenges.

1 Introduction

Computer security education has become one of the top priorities within governments and organizations of all kinds. While the most basic requirement is that computer users understand enough about security not to do something that will harm them or their environment (e.g., installing a "codec" that promises to display jumping kittens videos or more questionable material, but instead creates a backdoor in a system that handles sensitive data), there is an increasing demand for "security experts."

Even though security expertise is difficult to quantify and test [4, 6], a basic set of skills that go beyond basic security education is very valuable in the current job market (a simple query on LinkedIn for "Security Expert" returns thousands of positions available, many in Fortune 500 companies). Therefore, it is not surprising that educational institutions are increasing steadily the number of courses focused on security, at both the undergraduate and graduate levels. In addition, most courses on topics such as web development, operating systems, network-based applications, etc., now include a section on security issues.

Teaching computer security is hard for a number of reasons. First of all, security is a fast-moving target. New classes of vulnerabilities are discovered every year, and new protection mechanisms are introduced. For example, it would be infeasible to teach how buffer overflow exploits work today, without mentioning the fact that Address Space Layout Randomization, stack canaries, or No-Execute bits are ubiquitous protection mechanisms, which make this class of attacks more difficult to carry out.

In addition, it is not easy to create environments in which vulnerabilities and protection mechanisms can be reliably tested: they are difficult to set up, maintain, and evaluate, and, therefore, some of the system security classes offered today lack a valid "hands-on" component.

Security competitions provide a way in which students and security practitioners in general can test their skills in a competitive, hands-on environment. Given that security competitions usually span a somewhat limited time frame (from a few hours to a few days), these events are not the framework in which most security skills are acquired. Instead, it is the preparation period that precedes the competition itself that has proven to be extremely valuable in promoting security learning. In a way similar to the training of athletes who prepare for a running sprint, the skills are learned in the months spent training on the track, waiting for that brief moment at the actual competition. Participants in security competitions spend the months preceding the event strategizing about defenses, analyzing previous competitions to understand what is to be expected, preparing and developing tools, testing new approaches, and so on (see for example the results of the questionnaire described in [2], which show that half of the respondents developed *ad hoc* tools in preparation for the competition). This is the period in which most of the learning is performed. The actual execution of the competition adds the pressure to perform, the experience of being under active attack by a capable opponent, and the test of teamwork.

Because of the extra motivation provided by a competitive environment, security competitions have become increasingly popular and bigger in size [1]. The competitions take two main forms: they can be challenge-based or interactive. Even though both forms are often referred to as "Capture The Flag" (CTF) competitions, they are very different, and only an interactive competition can properly be called a CTF competition.

More precisely, challenge-based competition are structured in a way that presents to the participants a number of challenges that address different skills (e.g., reversing binaries, performing forensic analysis on file systems, manipulating network traffic) at different levels of complexity (which are usually associated with different amounts of points when challenges are solved). These challenges are a form of take-home test, and do not include any interactions with other teams. Instead, interactive (or "live") competitions focus precisely on the interaction between teams. Every participant is given the same system (usually a server with a number of network-accessible services) and their task is to identify flaws in their own copy of the server, patch (if possible) their own services without breaking the service's functionality, and use the same knowledge to attack the other participants. As proof of having been able to exploit an opponent's service, an attack involves grabbing a file or other data on the opponent's machine; this piece of data is referred to as a "flag" (and this is where the "Capture The Flag" label comes from). This type of exercise provide opportunities for exercising *both* attack *and* defense skills, and live exercises are therefore very different from challenge-based competition.

Unfortunately, designing and running interactive competitions is much harder than running challenge-based competition, and it is not surprising to see that, of the top 50 competitions listed on the CTFTime.org web site [1], only 8 are interactive. The challenges in designing, implementing, and running a live security competition (or "proper" CTF) are many. First of all, it is necessary to have the resources to support the creation of the game network (hardware, software, and human resources) and the skills necessary to monitor its usage. Then, one has to make sure that the game does not get "out of hand" in a number of ways: a team might perform a denial-of-service (voluntarily or inadvertently), a flaw in the game administration infrastructure might bring the whole competition down, and, if the competition is short-lived, it might make months of work completely useless (as the vulnerable services have been disclosed and a new competition will require a completely new setup to be fair to all participants). Finally, scaling these competitions can be a daunting task. Small competitions with a dozen of teams might be relatively easy to manage, but if a competition has more than a hundred teams and thousands of participants, it becomes very difficult to provide a seamless experience.

Year	Theme	Teams
2003	Open Source Windows	7
2004	UN Voting System	15
2004	Bass Tard Corporation	9
2005	Spam Museum	22
2006	Hillbilly Bank	25
2007	Copyright Mafia	36
2008	Softerror.com Terrorist Network	39
2009	Rise of the Botnet	57
2010	Mission Awareness	73
2011	Money Laundering	89
2012	SCADA Defense	92
2013	Nuclear Cyberwar	123

Table 1: The UCSB iCTF throughout the years.

At UCSB, we have been the first institution to run distributed educational CTF competitions, which we called iCTF (from "International CTF") [10]. We started in 2003 and have since designed, implemented, and run 11 security competitions (see Table 1), and, to this day, the iCTF has been consistently the world's largest interactive CTF focused on computer security education. In more than ten years of competitions, we have experimented with different designs, scoring systems, combinations of challenge-based and interactive competitions, and ways to collect interesting datasets that might support research into security education in particular, and system security in general [5, 7, 3, 2, 9, 8].

The success and the overwhelming enthusiasm with which iCTF participants responded to the event prompted many requests from education institutions and other organizations for a way to reproduce the iCTF concept on a smaller scale, maybe within a class or at a company training. We released a first version of a simplified form of iCTF software in 2010, but the capability of the software were very limited and required substantial skills to be configured and run.

However, our latest design for the iCTF, made us realize that there was an opportunity to create a framework (described in detail in the rest of this paper) that could be leveraged to easily create custom live security competitions, solving some of the challenges associated with the creation and execution of this kind of exercises automatically.

One key aspect of the framework is to allow third-parties to provide services and scoring components, so

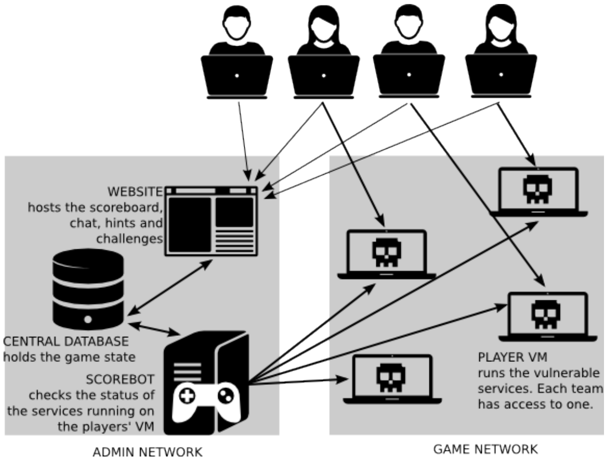
WEBSITE
hosts the scoreboard, chat, hints and challenges
CENTRAL DATABASE
holds the game state
SCOREBOT
checks the status of the services running on the players' VM
PLAYER VM
runs the vulnerable services. Each team has access to one.
ADMIN NETWORK
GAME NETWORK

Figure 1: System architecture overview.

that the community can contribute to a repository of instances of vulnerable software. As a result, competitions can be tailored to specific skill-set levels or types of vulnerabilities.

In the rest of this paper, we describe our iCTF framework and how it supports the creation of customizable live security competitions, hoping that, with the help of the security community, it will be possible to make this type of exercise available in a large set of educational settings.

2 The Platform

Although every year we introduce a novel theme and new game mechanics for the iCTF, the underlying platform that lets us run a CTF game remains the same. In our game, every team is assigned a virtual machine (VM), which is usually hosted by the teams themselves[1]. These VMs run vulnerable programs that are accessible over the network. The players' task is to keep these programs online and functional at all times and, if possible, patch them so that other teams cannot take advantage of the incorporated vulnerabilities. Shutting these services down is not an option, as their activity is vital for the players to participate in the game. The status of these services is constantly tested by a scorebot, which exercises the programs on a regular basis to ensure that the players are keeping the services up, and that their core functionality has not been crippled by an incautious patch.

Each service contains a "flag," a unique string that the competing teams have to steal so that they can demonstrate the successful exploitation of a service. This flag is also updated from time to time by the scorebot. We try to ensure that the players cannot tell when the scorebot is interacting with their services, because they could use this information to fingerprint the scorebot and make their services appear to be online just to the scorebot. This is achieved by masking the scorebot's network connections so that they appear to be coming from random, constantly-changing teams.

Upon successful exploitation, a team must submit the stolen flag to our website within a given period of time to receive points for it. Note that this website acts as a central point of information: it shows the current standings on the scoreboard, lets players chat among one another and with the organizers, and it is used to disclose hints and additional challenges.

The actual state of the game is kept secure in a central database (Section 2.2), that is protected from all direct access by the players.

2.1 Services Design

A service is a program that, from a high-level point of view, has a stated benign purpose (e.g., it implements a web forum) and contains some flags: secrets that are not supposed to be revealed to unauthorized users during normal operation (e.g., private messages in the web forum).

The goal of the opponent teams is to steal a flag and submit it to the organizers. Services are written with a variety of deliberately incorporated vulnerabilities, so that stealing flags is indeed possible, unless the defending team is able to patch all vulnerabilities successfully. Each vulnerability should be verified by the organizers to be actually exploitable in practice, to prevent unnecessary frustration for the players.

Generally speaking, a service listens on a given port, so that external programs can interact with it and invoke certain functions. This functionality should cover at least the following high-level categories: a *setflag* functionality that allows one to set the currently active flag; a *getflag* functionality that allows the benign and authorized retrieval of a flag; some additional benign functionality that represents the normal behavior of the service.

For instance, a service might implement a simple social network, where it is possible to create a user (with a given username and password) and to set a status. In this example, the user's status is the secret flag, and the status can only be read if the user is logged in by providing his username and password. Here, the *setflag* functionality would take as input the username, the password, and a status to be set, and it would then login and update the status of the user; the getflag functionality would take as input the same username and password, and retrieve the user's status after logging in. Note that the password is only known to the organizer and acts as an authentication token. On the other hand, to steal a flag, the attacker needs to discover and exploit a vulnerability in the service in order to read the status of a user whose password is not known.

[1] In the past two years, we decided to host the VMs at UCSB instead, to lower the setup burden for the teams.

One key point that requires attention when implementing an interactive security exercise is how such "secrets" are being stored within the service. In our implementation, we take into account the following considerations:

- The flag should change over time, so that the scoring system can distinguish between distinct attacks depending on various circumstances (as they will submit different secrets).

- The defending team should not be able to make a service unexploitable (in the sense that proper exploitation is not observable by the organizers) by simply changing the value of the flag. In other words, the defending team should be forced to protect a service by patching or neutralizing vulnerabilities, not by tampering with the flag or scorebot.

- The defending team should not know which is the currently active flag, so that naïve detection techniques, like searching for the flag value in network traces, do not work or would lead to false positives, and, in turn, decrease their score.

To this end, we designed a novel mechanism to store and retrieve flags. First, a new flag is periodically set and retrieved by our scoring infrastructure so that, at any point in time, each service has one and only one active flag. The scorebot does so by using the setflag and getflag functionality.

Second, the setflag and getflag functionality should blend with benign service interactions and not stick out in an obvious way. Not only the setflag and getflag should resemble benign interactions, but benign interactions should resemble setflag and getflag functionalities as well. In fact, the setflag and getflag functionalities can be used for benign interactions by simply setting or retrieving inactive or incorrect flags.

Third, from a conceptual point of view, the service stores not only one flag, but multiple flags. We stress that, at any point in time, only one of these flags is actually considered active by the scorebot. The defending team does not know which flag is active at the moment and it must protect all of them. To achieve this goal, each service conceptually stores a list of (`flag_id`, `token`, `flag`) tuples. In our previous social network example, `flag_id` corresponds to the username, `token` to his/her password, and `flag` to the status. Clearly, the implementation and storage details can vary depending on the service: a service might use an in-memory data structure, another might rely on the filesystem, and a third might use an external database.

2.1.1 Implementation Details

Some components must be developed to integrate a service into our infrastructure.

First, a set of scripts must be developed to exercise the various functionalities of the service. In particular, the setflag script takes as input the `flag_id`, a `token`, and a `flag`, and is in charge of installing a new `flag` into the service. The getflag script takes as input the `flag_id` and its `token` and should retrieve the corresponding `flag`, so that the scoring infrastructure can determine whether the service is properly functioning or not. Note that the getflag script can retrieve the `flag` associated to a given `flag_id` because it has access to its associated `token`, *not* by exploiting a vulnerability.

Other scripts exercise other service-specific benign functionality, so that the setflag and getflag scripts do not stick out in any obvious way, and to ensure that the service is fully operational and has not been tampered with (e.g., because of a careless patching process).

Finally, exploit scripts play an important role in the game. These scripts should be developed by the attacker teams (and, following best practice, by the organizers, to test the service's exploitability). Conceptually, an exploit script needs to steal the `flag` associated with a given `flag_id`, without knowing the associated `token`. Note that the `flag_id` specifies which of the many flags of the service the script must retrieve. For this reason, the only input the exploit script takes is the `flag_id`, and it must return the `flag`.

2.1.2 Integration with the Infrastructure

To ease the integration and testing of a given service, a set of information must be provided through a JSON file. In our current setup, this file is used to provide the following information to the infrastructure:

- Information about the author;
- File paths of the scripts (setflag, getflag, . . .);
- TCP or UDP port used;
- Service ID;
- Service name;
- Service description;
- `flag_id` description (i.e., what component of the service represents the `flag_id`, and, to some degree, how it is being used by the service; e.g., the username, in our social network example).

The last four items are also communicated to the participating teams, so that they know how an exploit script for a service should utilize the `flag_id`, and what should be returned as the flag.

2.1.3 Deployment of Services

The final step in designing iCTF services is deciding how they are deployed on the teams' VMs, both during the game and for testing.

Service-specific installation scripts can prove problematic, especially when services require certain non-default utilities or services to be present on the VM. Therefore, we decided to use the Debian package format, which is well-tested, has native dependency-handling, and can automatically start the service. In this way, once the package is successfully installed, the service can be trusted to be running and properly configured.

As a side effect, building Debian package encourages standardizing the compilation and deployment procedure: this is a useful feature in case organizers need to urgently modify and re-deploy a service due to issues discovered while the game is in progress (like an unintended denial-of-service vulnerability that was only discovered during the game).

As much as possible, the packages also standardize each service's permissions within the VM. It is typically undesirable to run all services with the same user and with files accessible across services: with such a setup, a single service would potentially allow obtaining flags for all other services. In educational settings, it is often preferred for each service to be independent, thus rewarding the ability to attack a range of different services (as opposed to leveraging a single type of exploit). Similarly, one might prefer limiting the amount of damage a single exploit can cause, for instance by prohibiting services from modifying themselves: this makes it considerably harder (or even impossible) for attackers to install backdoored versions. In our default setup, each service runs as a dedicated user and has access only to the files on which it needs to operate.

2.2 The Central Database

The central database is responsible for enforcing the rules of the game and for keeping track of its state. Essentially, the central database is, as the name suggests, the central component of the entire system. Every other component will pull information about the state of the game from the central database, and will notify the central database as to the changes of the state.

The central database is technically composed of two pieces: an HTTP-based RESTful API, with which the other components of the system interact, and the actual database, which stores all the data associated with the competition. We settled on this design because it allows for the decoupling of the underling database storage engine from the other components in the system. One year we experimented with having every component access the underlying database directly, however this approach had many drawbacks: changes to the database schema broke other people's code, the database schema was designed in an *ad hoc*, as-needed basis, and it was difficult to cache or otherwise limit the number of queries each part of the system performed. Thus, we believe that a better design decision in this case is to decouple the underlying storage engine and database schema from the way the other components of the system interact with the central database.

After a number of years of experimentation with different design approaches, we settled on the components of the system pulling information from the central database. We experimented with designing an iCTF where the central database would push important game state-changes to other components of the system, however, when there were problems in this pushing aspect, the central database system had to be restarted. Whereas, in a pull-oriented architecture, each component is responsible for deciding how often to pull from the central database and for taking action based on changes in the state of the game.

After running the iCTF competition for more than a decade, we have converged on (a few) best practices. One is that, for keeping track of any player point or score totals (or any changing value over time), it is much better to have an append-only table, where each separate action and its effects on the score are recorded separately. It also aids in debugging (and, possibly, rescoring) if each entry in this table has a description of why the score was changed and which component changed the score.

Moreover, it is critically important for *every* game action to be recorded in the central database. This not only empowers debugging, which often happens during the competition, but also eases the burden of post-competition analysis. In order to accomplish this properly, each component of the system must log all events within the central database.

2.3 The Scorebot

The scorebot infrastructure is responsible for monitoring the status of the services. It achieves this by interacting with the services using their *setflag* and *getflag* functionality. The scorebot periodically (the period or the *tick* duration is selected by the organizer) generates a new active flag tuple (`flag_id`, `token`, `flag`). These values are used to invoke the *setflag* and *getflag* scripts associated with each service, so that the respective service's functionalities are exercised and the active flag is set and retrieved.

In addition to setting up the active flags, the scorebot is also responsible for generating benign traffic to the services. The scorebot does this in two ways: first, it executes *setflag* and *getflag* scripts randomly with dummy `flag_id`, `token`, and `flag` values; second, it executes a set of *benign* scripts that are (optionally) provided by the service writer, so that all the service's benign functionality is properly exercised, and that one can be assured that patching the vulnerabilities in a service did not affect its intended behavior.

The values returned by these scripts (including the *benign* scripts) are used to determine the status of the service. A service is considered "up" if all executed scripts return *success*. It is considered "down" if one (or more) script fails to connect to the service. If a script fails because a service does not communicate properly, e.g., by sending improper data such as an invalid flag, the service is considered "non-functional".

Benign scripts should be independent from each other and should be executed in any order without affecting the state of the game. However, the execution order of the *setflag* and *getflag* scripts, which set and retrieve the active flag, must be preserved. In fact, if the *getflag* script would be executed before the corresponding *setflag* script completes its operation, it would fail, and the service would be incorrectly considered as "non-functional". While the scorebot independently executes these scripts with random delays, it maintains the order of execution when needed by utilizing locks. While the introduction of these random delays make it possible to exercise the service in benign ways between the flag updates and retrievals, it also prevents the scorebot from being fingerprinted through simple traffic analysis.

2.4 Putting it Together

One aspect that has become quite clear to us over the many iCTFs that we have organized is that, in addition to having quality components to a CTF design, these components need to function well together. In this section, we will describe how the different parts of the system fit together into an end-to-end CTF experience, both from the organizers' and the players' perspectives.

In this context, a CTF organizer is an entity that wishes to host a CTF competition for a certain amount of players. Our design for the system allows organizers to host CTFs without our involvement, going as far as drawing from our repository of services if they do not want to develop their own.

2.4.1 Choosing the Services

An important feature of our CTF framework is the presence of a repository of ready-to-use services. However, using these services comes with a serious drawback: often, solutions to previously-seen services ("write-ups") become available on the Internet once a service is used in a CTF. In most cases, to ensure a fair game, original services should be developed for each CTF. These services would have to be created to conform to our service specification and uploaded to our system. In cases where absolute fairness is not of great concern (e.g., a company educational exercise where the instructors can rely on a "do not peek" policy), the organizers can save a great deal of time by pre-selecting some of the available services.

2.4.2 Generating the Infrastructure

After services are created or chosen, our system proceeds to generate the virtual machines required to host the game. These virtual machines are distributed as VirtualBox Appliances, allowing us to leverage VirtualBox features such as *internal networking*, which is VirtualBox's simple Software-Defined Networking implementation.

The organizer must decide on several options:

VM Hosting. For scalability reasons, the organizer may leave the hosting of the teams' VMs to the teams themselves. When this option is chosen, our system generates a VPN concentrator VM for the organizer and VPN client VMs for the teams, which also act as routers for them.

Infrastructure Distribution. Again, for scalability reasons, our system can generate either a single VM containing all the infrastructure components, or single VMs for each piece.

Network Topology. By default, our system uses VirtualBox's internal networking feature, which automatically creates several LANs, depending on the first two chosen settings. However, VirtualBox's internal network requires all VMs to be on the same physical host, which once again brings up scalability concerns. To address this issue, our system can be configured to generate VMs with bridged networks, so that they can be hosted on different physical hosts.

The easiest way to host a CTF using our system is for an organizer to use internal networking, run a single infrastructure VM, and host all of the teams' VMs. This setup, spun up on a single physical host, is the most *turnkey* solution: once booted, no configuration is necessary, and the game is ready to be started from the infrastructure's administrative interface. The obvious downside to this setup is scalability. Once a game grows to include more than a dozen teams, hosting all of the virtual machines on one physical host becomes impractical and can be a source of rather unpleasant problems for organizers and players.

2.4.3 Game On!

If the teams must host their own VMs, the organizer must distribute them before the game. In most cases, the organizer will want to encrypt these VM, and release decryption keys at the start of the CTF, so that it is not possible to extract and analyze the services before the actual competition has started. The CTF can be started (and, later, stopped) using the administrative web interface.

3 Conclusions

The iCTF competition has demonstrated that it is possible to create interesting security exercises that involve more than a hundred teams and thousands of students. This paper presents a framework whose goal is to make the basic concepts behind the iCTF available, on a smaller scale, to educators across the world. The framework allows for the creation of custom competitions that are tailored to the skill-set of the participants, and, in addition, supports contributions from the security community, and, in the future, can be leveraged to create datasets to support security research. Parts of the framework have been used as the basis for running various editions of the iCTF. The iCTF framework will be made available for download at `http://ictf.cs.ucsb.edu`. This release includes a number of vulnerable services that can be immediately used to create live security exercises.

Acknowledgments

This work was supported by the National Science Foundation, through grants CNS-0820907, CNS-0716753, and CNS-0939188, and by the ARO through MURI grant W911NF-09-1-0553. We want to especially thank Carl Landwehr, Jeremy Epstein, and Karl Levitt at the National Science Foundation for their support to cyber-competitions.

References

[1] CTF Time. `https://ctftime.org`, 2014.

[2] Childers, N., Boe, B., Cavallaro, L., Cavedon, L., Cova, M., Egele, M., and Vigna, G. Organizing Large Scale Hacking Competitions. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)* (Bonn, Germany, July 2010).

[3] Doupe, A., Egele, M., Caillat, B., Stringhini, G., Yakin, G., Zand, A., Cavedon, L., and Vigna, G. Hit 'em Where it Hurts: A Live Security Exercise on Cyber Situational Awareness. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (Orlando, FL, December 2011).

[4] (ISC)^2. Cissp. https://www.isc2.org/CISSP, 2014.

[5] Shoshitaishvili, Y., Invernizzi, L., Doupe, A., and Vigna, G. Do You Feel Lucky? A Large-Scale Analysis of Risk-Rewards Trade-Offs in Cyber Security. *ACM Symposium on Applied Computing* (March 2014).

[6] Timmay. Why You Should Not Get a CISSP. DEFCON 20, July 2012.

[7] Vamvoudakis, K., Hespanha, J., Kemmerer, R., and Vigna, G. Formulating Cyber-Security as Convex Optimization Problems. In *Control of Cyber-Physical Systems*, vol. 449 of *Lecture Notes in Control and Information Sciences*. Springer, July 2013, pp. 85–100.

[8] Vigna, G. Teaching Hands-On Network Security: Testbeds and Live Exercises. *Journal of Information Warfare 3*, 2 (February 2003), 8–25.

[9] Vigna, G. Teaching Network Security Through Live Exercises. In *Proceedings of the Third Annual World Conference on Information Security Education (WISE)* (Monterey, CA, June 2003), C. Irvine and H. Armstrong, Eds., Kluwer Academic Publishers, pp. 3–18.

[10] Vigna, G. The UCSB iCTF. http://ictf.cs.ucsb.edu, 2014.