# Hosting a Hacking Challenge – CTF-style

Lexi Pimenidis

Chair of Computer Science for Communication and Distributed Systems – RWTH Aachen

## 1 Introduction

There are a couple of reasons to create and host a hacking challenge. One of the major reasons for universities and other educational institutes is, to teach defensive and offensive IT-security in a somewhat more realistic environment than the traditional classroom.

Although a hacking challenge is a lot of work to set up, it is a large gain for everyone. The participants, i.e. students, will have new insights and learn how to work under pressure. A successful challenge will raise its host's reputation in the world of IT security and demonstrate his skills to manage and setup complex environments. Experience shows that students very much enjoy this untraditional style of practise and are willing to learn and exercise hard in preparation for such an event. Needless to say that a well organized CTF is fun for everyone.

The main goal of a hacking challenge is to simulate a network under constant attack and let the participants do their best to cope with the situation. Of course, the players will already need to have basic knowledge in defending systems to draw an advantage from these situation. Even more, it is intuitive that hacking challenges are more suitable for the advanced courses, where not only defense skills are taught, but also basic knowledge of offensive approaches to IT security are known.

In a CTF-style hacking challenge participants are usually grouped into teams that are each assigned to a specific server. Recent exercises have shown that teams work most effective, if they are of size five to ten. While smaller teams don't have the man power to keep up with larger teams, too large teams tend to be unproductive because of the large internal communication overhead. The teams' task is to keep the own server's functionality up and its data confidential, while trying to disturb the other teams' services at the same time. The deployed services are most often the same on each server, such that the teams can analyze the services' structure on their own host to gain insights in their functionality. If a vulnerability is found in a service, it can be fixed on the own system and exploited elsewhere.

Scores are assigned for defending the own system and compromising other servers. "CTF" stands for *capture the flag*. *Flags* are small pieces of data, unique random-looking strings, that are stored on each service of every team. The *flags* are stored and retrieved in intervals of several minutes to test, if a team could keep a service up and functional. Defensive scores are assigned for availability of services.

That is, unless some other team was able to compromise a service and could read the stored data. The flags are considered *captured*, if one team can submit the *flags* of another team to a central database. In that case, the capturing team receives offensive points, while the defending team's defensive points are cancelled. The task of submitting flags, retrieving them and keeping track of the scores[1] is done by a piece of software called *gameserver*. The duration of such a challenge can range from several hours to several days.

In the course of this article, I'll describe some preconditions and initial work that has to be done in order to host a CTF-style hacking challenge (or short: CTF). Section 3 is about choosing and creating the services, which is the central part of the challenge. Section 4 will briefly discuss some common pitfalls and contains links to more information.

## 2 Preconditions and Initial Work

This section deals with basic questions on the organizational part of a hacking challenge. While some of them might seem trivial, their importance shouldn't be underestimated. Any challenge that is of non-trivial size can only be accomplished as a major effort of the host, as well as the participants.

Such, the first decision should be to agree on a date, the duration and a place for the event to happen. The location is of minor importance, but a good Internet connection is suggested. Since a CTF can be done "distributed", players can participate over the Internet from any location. If remote participation is allowed and the game lasts only for a few hours, the time frame should be chosen in a way, that the majority of players does neither need to work early in the morning nor very late.

In the following, the focus will be on a distributed CTF. Since the actions during the course of the hacking challenge will be of potential harm to third parties if accidently misdirected, all traffic will has to take place within a closed and secured VPN. To control and log data flows between the teams, all traffic has to be routed over a central VPN-server. An example network layout for two teams is depicted in figure 1.

Hardware and Software requirements differ for each event, but a typical setup can be done with a gameserver and a central router for the VPN. Each team is suggested to deploy a gateway, the server that runs the services and one workstation for each participant.

In a distributed scenario, each team runs its server locally. To ensure that the services initially have an identical setup, the services are deployed as part of a virtual machine. The virtual machine is setup in advance by the host, encrypted and distributed ahead of the exercise. At the start of the exercise the encryption key is published, the images are decrypted and started. Another advantage of working on virtual machines is that the memory layout is identical.

[1] The team with the highest score wins. But in my opinion, winning the contest is a secondary goal only.

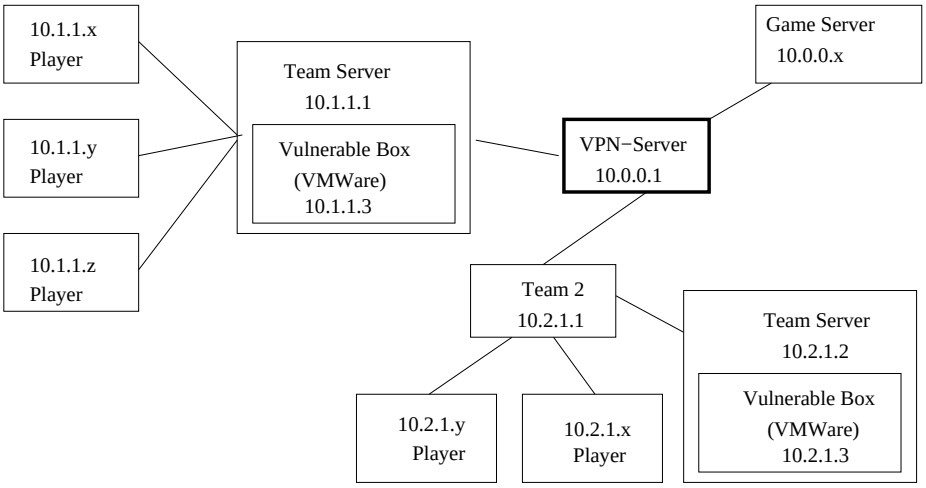


**Fig. 1.** An example network layout, using a central VPN-node

Although the term "hacking challenge" implies a certain absence of regulated behavior, it is commonly accepted that even during this sort of event some basic etiquette has to be kept. Since CTFs are quite new, there is yet no elaborated set of rules. Instead, every host chooses to adapt rules from former exercises and creates new ones. Still, some common basics exist: the most important rule is that destructive behavior, be it due to unsophisticated denial of service attacks, or wiping essential system files on other teams' servers, is forbidden. The same applies to intentional support of other teams. Since a CTF usually allows a larger number of teams, it's common that multiple teams from e.g. a single university take part. Thus it has to be avoided that cooperation of these teams lead to unfair advantages. Automated tools that work around security issues, like e.g. stack protection mechanisms, without fixing the cause of a vulnerability are banned from most challenges. Finally, it is strictly forbidden for the participants to filter requests in order to block queries from the other teams and accept queries from the gameserver.

Of course, it is difficult, if not impossible, to enforce the above rules, if the teams are distributed over different continents. Thus a team usually has to assign a person that is not actively playing as a local referee. Additionally all traffic is logged to allow in-depth investigations of possible incidents later on[2]. To make filtering of queries more difficult, the traffic is sometimes anonymized on the IP-layer. But then, care has to be taken to keep a good QoS of the network: it has happened that the central router was not able to keep up with the participants and the network broke down for large periods of the exercise.

[2] The traffic logs can also be used as training data for for forensic classes.

Having understood the impact and organized all of the issues in this section, the main work of hosting a hacking challenge starts: choosing and creating vulnerable services.

## 3 Vulnerable Services

The most difficult part in hosting a CTF is to choose or create a set of vulnerable services. Unfortunately it is also the most crucial part. If the weaknesses are too difficult to find and exploit, then the participants will soon get tired of looking for them and loose interest. As such it is important to have some vulnerabilities that are obvious, simple to fix, and trivial to exploit. After these bugs are inserted, the host can start thinking about hiding the next security issues deeper in the code. In some recent CTFs it happened that there were some services left without analysis by any team because of the degree of difficulty. This should be avoided because it is an unnecessary waste of the organizer's and participants' time and resources. Additionally, it can be regularly seen that some teams can't fix even rather simple bugs, such that (unfortunately!) there is likely in any CTF a chance to exploit other teams' services even hours after the start of the exercise.

An easy way to provide some services, is a setup consisting of ordinary out-of-the-box software, like e.g. Apache or Samba. To make exploitation feasible, older versions are preferred – it makes no sense asking teams to come up with zero-day exploits. An excellent choice of software can also be found in some freeware collections of e.g. webbased guestbooks, where the work of unskilled programmers is distributed. On the other hand, there is a huge drawback with this approach, since there are often some exploits for this kind of software in the wild that can be found with Google or in similar databases. This raises the risk that players will not analyze the code themselves or look for innovative methods to defend their servers; instead they tend to compile the latest version and look for existing exploits in the WWW. Especially the latter is definitely something that should not be encouraged by CTFs.

The second best thing to do, is to take existing software and deliberately insert vulnerabilities to make exploitation easier. But even well hidden bugs can be trivially detected by comparing the modified program with the original source or binary. To avoid this, all appearances of a program's real name and version have to be changed. A task that can get very fast tedious and if some clues remain, will be in vain.

A solution that circumvents all of this problems and is commonly adapted throughout major CTFs is, to write custom services from scratch. To make services exploreable and exploitable by other teams and the gameserver, they are in some way connected to the network and listen for input. Note that *custom* does not necessarily refer to custom network protocols, but rather to the implementation. Although widely known standards like POP3, HTTP and SMTP are preferred, sometimes arbitrary new protocols can be encountered, as well as cross-overs.

### 3.1 Cookbook for Custom Services

Designing a custom service for a CTF is probably more kind of an art than real programming due to the didactical aspects that have to be integrated. Before starting to code, the programmer should have a clear idea on the type of the service, i.e. he has to decide upon the network protocols, the kind of vulnerabilities that will be included and the programming languages used.

Unless the participants are supposed to learn a new programming language under pressure, it is recommended to stick with widely known languages. Most CTFs have at least some parts that make use of PHP, Perl, C, and SQL. A reverse engineering challenge is always included, too. Commonly used are also Bash and Python, to a lesser extend Java and C++. There weren't yet services written in Basic, Pascal, or Mono, although the latter might be an interesting choice for upcoming events. Whatever the choice is, the creator of a service needs to have a very good understanding of the programming language's structure and possible vulnerabilities.

The service should be designed, not only with the interfaces in mind, but also must have a natural way of working with flags, i.e. storing and retrieving confidential data. This can be accomplished by extending a network protocol to include new commands for the gameserver, or install backdoors in the code that can not be exploited by the players. But in general it is recommended to stick with the normal behavior of a service, e.g. if the service is a mail-server, the flags should be stored by SMTP and retrieved by POP3, instead of connecting to the server with a SSH-shell and leaving a flag in an undocumented location.

Up to now, not a single line of code was needed to be written, but if the above issues have been integrated into an overall design, coding can start. It is a recommendation to first code a flawless version of the service that is well documented and understood. For a CTF that lasts six hours, a service should have between 500 and 2500 lines of code. The complexity should not exceed the level than can be accomplished by a single average participant within the given duration of the exercise.

When the clean version is finished, the time has come to spin-off the vulnerable version. The main reason for this being late in the process is, that only this way, the programmer of the service can be sure to control the way, the service will get exploited. Otherwise there might be some ways to break it, that weren't foreseen. As already discussed in Section 3, there should always be some easy to spot vulnerabilities included. To raise the degree of difficulty, atypical vulnerabilities can be inserted, where appropriate, e.g. shell code injection in C programs.

The main purpose of each planted vulnerability is to give "unauthorized" access to the stored flags. Depending on the rules of the specific exercise, it should be avoided that this access can be used to destroy flags. Otherwise a team might start collecting flags and destroy them afterwards, such that the other teams can't collect them anymore. In any case it must be avoided that a vulnerability in one service can be used to read the flags from another service. If so, the fact should be taken into account for the scoring. The worst thing to happen is that

a team easily reaches administration privileges on another team server, thus being capable to read all flags at will and destroy whatever they like. Thus the operation system on the servers should be as secure as possible, separating the vulnerable services from each other and protecting the basic functionality.

The last few items on the check list for a custom service include writing a module for the gameserver, such that the flags can be set and retrieved for the scoring system. Some testing in the final environment is obligatory, while example exploits for the intentional vulnerabilities are optional.

## 4 Common pitfalls

This section will, in no specific order, touch further issues that should be kept in mind.

If the contest is done remotely and the services are distributed in an image of a virtual machine, care has to be taken that the image will boot correctly and that all participants, possibly being from other countries, can work on it. Such, the timezone should be set to UTC, the keyboard layout set to US, and automatic hard disk-checks at startup should be disabled.

In general, the most likely cause for a drop-out of the exercise are hardware failures, directly followed by software failures of untested setups. Organizers should always have some spare hardware during the contest, which is ready configured for hot-swapping. The same applies to a lesser degree for each team's router and server, whereas workstations are not critical.

Depending on the amount of advertisement that has been done for the service (if any), there will be some teams that are incapable of connecting to the VPN or didn't understood the rules properly. Prepare a policy how to deal with these in advance and firmly stick with it during the CTF. The same applies to violation of given rules. Bear in mind that, whatever any rules might be, they are fair as long as they treat all participants and all of them equal. Good experiences have been made, to publish detailed rules in advance, such that there are no misunderstandings.

Finally, all side channel attacks on flags and the scoring system have to be avoided. That is, e.g. flags should under no circumstances be computable or follow some scheme that can be broken.

More information can be found on the web-pages of those who host CTFs on a regular basis:

**CTF UCSB** http://www.cs.ucsb.edu/ vigna/CTF/
**CIPHER** http://www-i4.informatik.rwth-aachen.de/~ lexi/cipher/
**op3n** http://www.ito.tu-darmstadt.de/edu/ctf/da.op3n(2005)/
**Italian CTF** http://idea.sec.dico.unimi.it/ctf/index.it.html