Security of Programming Contest Systems

Michal Forišek

forisek@dcs.fmph.uniba.sk,
Department of Informatics,
Faculty of Mathematics, Physics and Informatics,
Comenius University,
Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract. In this paper we examine currently used programming contest systems. We discuss possible reasons why we do not expect any of the currently existing contest systems to be adopted by a major group of different programming contests.
We suggest to approach the design of a contest system as a design of a secure IT system, using known methods from the area of computer security.
The main contribution of this paper lies in a detailed analysis of new, specific threats that arise in the context of programming contests. A secure contest system should be able to handle all the threats mentioned in this article.

1 Introduction

Computer science contests are one of the most successful ways to advertise informatics, both to possible investors and to students deciding their career options. As of today, there are lots of different programming contests with different targer groups – from primary school students to university students, there are even some contests open for everybody.

Some of the most famous worldwide programming contests are the International Olympiad in Informatics (IOI, a science olympiad for secondary school students, [8]), the ACM International Collegiate Programming Contest (ACM ICPC, [1]), TopCoder Algorithm Competition (TopCoder, [14]), and the Internet Problem Solving Contest (IPSC, a yearly programming contest known for using new, original task types, [9]).

Most of the programming contests use some kind of automatic evaluation of the contestants' submissions. Usually the contest organizers (create and) use a software package that provides this functionality, along with some additional functions, such as providing an interface for the contestants, administration functions, etc. This software package is called a *contest system*.

There are some online contest systems, the most famous is the University of Valladolid judge (UVa judge, see [15]). These systems allow people to organize some fixed kind of a contest, usually a contest mimicking the ACM ICPC.

In this article we want to analyse the current situation in the area of contest systems used in various programming contests. The main focus will be on the computer security point of view.

Currently, almost each programming contest uses its own contest system. Still, one has to mention that there are some exceptions to this rule. Probably the most widely used contest system is PC^2 (PC squared, [2]). It is used to organize ACM ICPC contests in multiple regions worldwide. However, in personal communication with contest organizers we often got the response that they only use PC^2 because they lack both an alternative that would suit their needs better, and the resources to develop their own contest system. Moreover, PC^2 is designed for the ACM ICPC only, and thus it is completely unsuitable for any other contest type.

Clearly, the current situation has got many disadvantages to the whole community behind the programming contests. The same work (designing and developing the contest systems), and the same mistakes are repeated over and over again. There is almost no sharing of experience, and almost no reusing of the contest systems happens.

In this article we try to make first formal steps in the direction towards a contest system that could be adopted by organizers of different contests.

In Section 2 we give a brief overview of the internals of a contest system. In Section 3 we discuss the requirements a contest system should fulfill in order to be acceptable for organizers of different contests. In Section 4 we suggest an approach to designing a secure contest system.

Section 5 contains the main contribution of this paper. In this section we give an overview of specific threats to contest systems, we classify the threats into categories, and discuss ways of preventing these threats.

2 Overview of a generic contest system

In this section we give a brief overview of the features common to most contest systems. A reader familiar with some contest system may skip this section.

The main functionality provided by the contest system is *an automated judge*. This is a piece of software responsible for automatical evalution of the contestants' submissions. The canonical way of evaluating a submission consists of *compiling* the source code submitted by the contestant, *executing* the resulting program on a prepared set of test inputs, and checking the program's output for correctness.

When the contestant's program is being run, it is usually isolated in a logical area called a *sandbox*. The goal of the sandbox is to prevent malicious submissions from accessing the restricted areas of the contest system, and to enforce various limits set on the submitted program.

Other functionality necessary for running a contest usually includes *user management* (e.g., authentication, access control), *administration functions* (e.g.,

adding/editing tasks, test data, setting time limits, rejudging submissions), *contest evaluation* (computing the rankings), and a *contest portal* (displaying rankings, handling submissions and clarifications, displaying additional information).

Some of the existing contest systems offer additional functions, such as allowing the users to *backup* their data, or *print* their source codes.

3 Requirements on contest systems

We will start this section by trying to identify the main requirements on a contest system that could be adopted by a major community of contest organizers. Of course, these requirements can also be read as reasons why none of the currently existing contest systems is universally adopted.

Flexibility. Each programming contest is different, and, naturally, each programming contest has other requirements on the contest system. (As a canonical example consider the part of the system that computes and displays the rankings – it has to reflect the contest's rules.)

A contest system that aims to be universally adopted has to be easy to adjust to different contest rules. The contest system has to be designed with flexibility in mind. One example of a contest system designed to be flexible is CESS2 [11]. The flexibility is achieved through the fact that the parts that may need to be changed are implemented in a simple scripting language, Lua [7].

To achieve flexibility, the contest system should have a modular design, where it is easy to replace some of the modules, and reuse other ones without a change.

Robustness. First of all, a universal contest system has to be robust and reliable. There is not much hope in a contest system that tends to crash on random occassions, or when used in ways not intended by its author.

User-friendliness and support. On a similar note, the system has to be easy to use, and documented well. Note that most of the current contest systems only have the functionality they need to run the contest. They are poorly documented, or not documented at all, and they can be only used by their designers and developers.

A recent example of a contest system that aims to be robust and user-friendly is the Polish SIO.NET system [13].

Trust. The issue of trust (in the security of the contest system) has been neglected in the past. Quite on the contrary, we consider this to be one of the most important issues. We percieve the whole contest system as a secure IT system: The system has to be able to restrict access to some information, protect the integrity of stored results, and so on.

We expect that the issue of trust and security will play a more important role in the future. Especially for major contests the trust in the used contest system is really an important issue, and it really should be addressed by contest system authors.

We give an initial discussion of this matter in the next section.

4 Designing a secure contest system

The standard framework used for designing and evaluating secure IT systems is given by the Common Criteria (CC, [5]). A short overview of the CC is given in [4]. The main reasoning behind this framework is that most secure IT systems share many properties and functional requirements, and that the logical consequence is the potential of the same types of attacks.

The CC provide a framework that, given a set of functional and security requirements, helps the designer to establish a set of protective measures that have to be implemented. In other words, the output from applying the CC framework is: "if you want a system that is this secure, you have to implement these security mechanisms, otherwise it would be vulnerable to such and such types of attacks".

It is a common practice to certify secure systems against the CC framework. The certificate states that the design of a system correctly identifies all necessary functional and security requirements for the given security level.

5 Attacks on contest systems

Computer security can be viewed as a game between the attacker and the defender – both try to think of possible attacks, one tries to employ them, while the other tries to prevent them.

When designing a secure IT system, the framework given e.g. by the Common Criteria, shows us the threats that are inherent in the design of each system having the selected functional and security requirements. However, this does not suffice. We always have to consider the situation at hand, and identify and prevent the threats specific to our system.

For contest systems, there is no known list of such attacks. This paper tries to fill in this gap, thereby establishing a knowledge base in this area. The developers of the current contest systems should check them for vulnerabilities to the attack we describe. More importantly, future contest system designers should design the new contest systems with these attacks in mind. Note that we are aware that many of these attacks were attempted in practice, and we tried almost all of them on test installations of various contest systems.

The most intuitive way of classifying the possible attacks is to determine the time when the attack happens. In this classification we will distinguish three types of attacks: *compilation-time*, *execution-time*, and *contest-time*. (The first two types of attacks occur during the compilation and execution of the contestants' submissions, the third type contains all other attacks.)

In addition, some of the attacks fit into known categories from the area of computer security. We will now give a brief overview of these categories.

The purpose of a *denial of service* (DoS) attack is to block some users of the IT system from accessing and using a particular resource. A common example of a DoS attack is when a computer virus uses infected computers to overload

the inbound or outbound internet connection of some site, thereby denying access to the site's regular visitors. A general overview of DoS attacks, preventive measures, and possible responses is given in [6].

The most common attack type is a *privileges escalation* attack, where the attacker exploits some flaw in the design or implementation of the system to access restricted areas, materials, etc.

The only goal of a *destructive attack* is harming the system in some way, and thus disabling some of its functions or security measures.

The term *covert channel* is used to describe a situation where the attacker uses some way not intended by the system's designer to communicate information. Covert channels are usually used to communicate classified information to unauthorized users or areas. The idea of covert channels was introduced in [12].

Forcing a high compilation time

Types: *compilation-time, denial of service*
Description: A short and simple program can still require an unreasonably high compilation time. If the contest system does not enforce a time limit on the compiler, this can be used to block the compiler and thus to endanger, or even completely stop the automated testing process. An example of one such C++ program is given in the Appendix A.1.
Note: Here we would like to note that the IOI adopted a compilation time limit after one contestant submitted a legitimate, non-malicious program that took approximately 90 seconds to compile.
Prevention: Setting a compilation time limit is enough to handle this problem. Using more than one machine to test the contestants' programs in parallel may help to mitigate effects of this attack, but it doesn't necessarily prevent it.

Consuming resources at compilation time

Types: *compilation-time, denial of service*

Due to the language design, some compilers (especially C++ compilers) may sometimes need to make an indefinite lookahead when parsing the source code. (I.e., to understand a command it needs to examine the code that follows, and we can not bound how much code will have to be examined.)

If the compiler is given a large input file, it may need to read it whole into memory. (Some compilers even do this always, even if it is not necessary.) An especially nasty way of achieving this effect in a Unix-like environments is to have the line: `#include "/dev/urandom"` included in your source code. The compiler will read the random garbage into memory until it runs out of memory.

Note that if the operating system can be forced to run out of memory, this can have unpredictable, and even fatal results. For example, any process can be killed if it happens to request more memory when all memory is used up.
Prevention: A way to prevent this from happening is to compile the contestants's submissions inside a limited environment – the compiler will have access

only to a restricted part of the system resources (e.g., memory). Probably the most simple thing to do would be to run the compiler in the same environment you use for *running* the contestants' submissions.

Accessing restricted material

Types: *compilation-time/execution-time, privileges escalation*

The attacker may try to access restricted material, such as the correct outputs for the test data, author's solution, the submissions of other contestants, a log file with the evaluation results, etc. For most of these resources even read access can be abused.

The non-obvious part of this attack is that it can happen during compilation time. If the attacker knows or guesses the right location of the restricted files, he may try to include them into his program. (E.g., imagine a submission consisting of only one line: `#include "../authors-solution.cpp"`)

Prevention: Years of computer security have shown that *security through obscurity* can never work. (If you are not familiar with this topic, read [3] for an unrelated but fascinating example.) The best approach to security is transparency. The design of the system should be public, and it should be such that the contestant's program has absolutely no access to the restricted material. Ideally, when the contestant's submission is being compiled and run, most of the restricted material should not be present on the given machine at all.

Misusing the network

Types: *contest-time, privileges escalation*

If the contestant is allowed to monitor and store network traffic, especially incoming traffic for the automated judge system, he may be able to intercept other contestants' submissions and later submit them as his own.

In on-site competitions, other attacks from this category include communications between the contestants over the network, or using other contestant's password to access his submissions. (We are aware of a past situation when the latter attack was attempted in an international competition.)

Prevention: All communication has to be done in secure ways, e.g., using the HTTPS protocol. In the case of an on-site contest, additional preventive measures are possible, such as configuring the network hardware (switches) to allow only server $\leftrightarrow$ client communication, monitoring the network traffic by the organizers, and logging all contestants' actions (including logins into the system).

Modifying or harming the testing environment

Types: *execution-time, destructive/DoS/other*

The simplest type of a harming attack is to submit a program that tries to delete all it can. A slightly more subtle way of harming the system is generating a huge output (file), thereby forcing the system to run out of disk space.

However, the attack does not always have to be destructive. We will give a concrete example of one attack that modified the testing environment in a clever way.

In the attacked system the contestant's program was compiled and run on several test inputs. After each run, the output of the contestant's program was compared to the correct output (using a file compare tool `fc`). If and only if for all outputs `fc` reported no differences, the submission was accepted.

The attacker's program did not produce any output. Instead, it created a new binary called `fc`. After the contestant's program finished, the automated judge software called this binary instead of the original `fc`. The only things this binary did was to copy the correct output to the contestant's output file, report that no differences were encountered, and delete itself. Note that at this moment the system was back in its original state, and all following submits were tested normally.

Another, more simple example of this type of attack is pre-computing some information and keeping it in a temporary file. In the next runs, if the temporary file is found, its contents is used to save running time.

Prevention: Even in the sandbox environment the designer of the contest system has to be very careful about what should the contestant's program be allowed to do. Almost anything that is not necessary for a correct program can be misused to harm the system.

Circumventing the time measurement

Types: *execution-time, other*

One of the more subtle attack types is trying to circumvent the automated judge's time measurement to get more computation time.

The simplest possibility is that if the system only measures the time of the process it starts, the running program can fork a child process (which now has unlimited time) and sleep until the child process finishes.

There are more subtle approaches, that can succeed even if the imposed limits do not allow the solution to fork. For example, in the currently used Linux kernels (versions 2.4 and 2.6) the process time measurement is not precise enough. A solution that always runs for a short period of time and then sleeps for a carefully determined period of time may be able to use an arbitrary amount of processor time without this showing in the process information given by the kernel. (We were able to implement this type of attack on a machine running a vanilla 2.4 series Linux kernel.)

A possible way to block the testing facility is to let the program wait for an unavailable system resource. Such a program doesn't consume processor time, and it can run for an arbitrarily long time without exceeding the processor time limit.

Prevention: In the Slovak contest system the testing machine uses a patched Linux kernel to allow exact processor time measurement for processes. The patch can be found at [10]. In general, the designer of the system has to be aware of the

internals of the platform he decides to use, and be sure to measure the process time correctly.

In addition, we suggest limiting the contestants' programs to one thread (disallowing `fork()`s, `clone()`s, and analogous system calls on other operating systems). A good preventive measure is to have a (sufficiently large) real time limit in addition to the processor time limit.

Exploiting covert channels

Types: *contest-time, covert channels*

The contest system often informs the contestant about the success of his submission. This information often depends on the submitted program's behavior on the (secret) test data. (ACM ICPC-like contests are a canonical example of this situation.)

The information given to the contestant can be used to get information on the secret test data. The more information is given to the contestant, the more bits of secret information can be retrieved using a single submission. The most common things to use are the termination report (finished successfully, aborted, runtime error, etc.), and the information on the time and memory consumed by the program. An outline of a simple program that sends messages using this covert channel is given in Appendix A.2

Note that the information obtained in this way can be used to write an incorrect heuristic program that correctly solves the given set of test data. This attack tends to be very successful for tasks where the answer is binary (YES or NO).

Prevention: For some contest types there is no way to prevent this type of attack. The only thing that can be done is to mitigate its success. This can be done by careful choice of how the automated judge's response should look like, and by using good test data.

Misusing additional services

Types: *contest-time, denial of service*

Often the additional services, such as source code printing or backup facility, are not as secured as the rest of the system, and it is possible to misuse them.

The most common form of an attack is a denial of service attack on one of the services. A simple example: a contestant submits a 500-page text file that uses up all the available paper in the printer.

Sometimes the attack may even harm other parts of the system. E.g., the attacker may try to backup inappropriate amounts of data, thus forcing the contest system to run out of disk space.

Prevention: We advice setting a hard limit on the number of printed pages, and the size of the files a contestant is allowed to backup. (Similar bounds should apply for other additional services.)

Note that we are not talking about limiting the *size* of the file submitted for printing. The issue of printing is more tricky than it may seem at the first sight.

Using the PostScript language one can create documents shorter than 1 kB that contain millions of pages, or take arbitrarily long to evaluate. One such example is given in the Appendix A.3. We would like to stress that also many print filters (like `a2ps`) are vulnerable to this attack.

Exploiting bugs in the operating system

Types: *execution-time/contest-time, privileges escalation*

This type of attack should be partially covered by the general security requirements for secure IT systems. However, there are specific issues worth mentioning.

If the contestant is able to access the machine where the submissions are tested, he may attempt to scan this machine for known vulnerabilities, and try to exploit them. This is the general attack type.

The specific issue in this case is that the contestant's program is executed on the testing machine with the privileges of some local user. This is always considered to be a higher security risk than an outside access to the machine's services. The contestant can try to run arbitrary code on this machine, including an *exploit* – a short program that uses some bug in the operating system or one of its services to gain (usually super-user) privileges. Successful exploiting of the machine that does the testing may lead to the exposure of secret testing data or even let the contestant harm the system.

Prevention: It is advised to isolate the parts of the system the contestants should not have access to. The contestants should only be allowed to communicate with a computer that works as an interface (a gateway) to the whole contest system.

The contestants' submissions have to be executed in a secure sandbox environment that prevents them from accessing unnecessary parts of the operating system.

Obfuscation

Obfuscation is not really an attack type, it is just a tool to help other types of attacks to succeed. For example, many of the contest systems we encountered simply used `grep` (a tool to search for a string in text) to check whether the program included any forbidden system calls.

Prevention: The method described above is **not adequate** to prevent the program from making the forbidden system calls, or other forbidden attacks. For several contest systems we were able to successfully circumvent such protection.

The best way to prevent obfuscated attacks from succeeding is, of course, checking for the real threats, not for their common symptoms. E.g., if you want to check whether a program uses the `fork()` system call, do not look for it in the source code. Instead, intercept and check all the system calls the program makes during its runtime.

6 Conclusions and further research

Clearly, the authors of the existing contest systems should check them for vulnerability to the attacks described in this article, and fix found vulnerabilities. Any further discussion of known attacks would surely be interesting and helpful.

As a goal for future contest system authors, we would suggest to design the contest system as a secure IT system, and to specifically address all known case-specific vulnerabilities in its design. The resulting design, including sufficient rationale on addressing the vulnerabilities, should be made available to the system's end users. In other words, a contest system that aims to be universally adopted has to contain an explicit claim about the security functions it provides, and a sufficient overview on how these functions are achieved.

This paper can serve as a reference for such claims.

7 Acknowledgements

The author would like to thank all reviewers for their insightful comments that helped to improve the quality of this paper.

This work was supported by the Comenius University Grant Programme under the number UK /397/2006.

A Code snippets illustrating some of the attacks

A.1 Forcing a high compilation time

```
#include <map>
using namespace std;

typedef map<int,int> M1;
typedef map<M1,M1> M2;
typedef map<M2,M2> M3;
typedef map<M3,M3> M4;
typedef map<M4,M4> M5;
typedef map<M5,M5> M6;
typedef map<M6,M6> M7;
typedef map<M7,M7> M8;

int main() { M8 tmp; }
```

A.2 Exploiting covert channels

The following function is able to send 7 bits of information in the automated judge's response, if the response contains the termination type and the memory used (rounded to megabytes).

```
void sendInformation(int n) {
  malloc(1024 * (n&31)); // send 5 bits in the memory size
  n >>= 5;
  if (n==0) exit(0); // terminate with a wrong answer
  if (n==1) assert(0); // abort
  if (n==2) while(1); // exceed the time limit
  if (n==3) { int a=3*n; a/=9-a; } // division by zero
}
```

A.3 Misusing additional services

A small PostScript file with a huge/infinite number of pages.

```
%!PS
/Times-Roman findfont 100 scalefont setfont
/page 0 def
{
  /page page 1 add def    % increment the page counter
  page 4 gt { exit } if   % try omitting this line!
  100 550 moveto
  page 20 string cvs show % print the page number
  showpage                % ship out the page
} loop
```

References

1. ACM International Collegiate Programming Contest. `http://icpc.baylor.edu/` [accessed Mar 2006]
2. Ashoo, S.E., Boudreau, T., Lane, D.A.: Programming Contest Control System PC^2. `http://www.ecs.csus.edu/pc2/` [accessed Mar 2006]
3. Blaze, M.: Cryptology and Physical Security: Rights Amplification in Master-Keyed Mechanical Locks. IEEE Security and Privacy, 2003. `http://www.crypto.com/papers/mk.pdf` [accessed Mar 2006]
4. *Common Criteria – An Introduction.* `http://www.commoncriteriaportal.org/public/files/ccintroduction.pdf` [accessed Mar 2006]
5. *Common Criteria for Information Technology Security Evaluation.* Version 2.3, 2005. ISO-IEC standard 15408. `http://www.commoncriteriaportal.org/` [accessed Mar 2006]
6. CERT Coordination Center: Denial of Service Attacks. 1997. `http://www.cert.org/tech_tips/denial_of_service.html` [accessed Mar 2006]
7. Ierusalimschy, R., de Figueiredo, L.H., Celes, W.: Lua – an extensible extension language. Software: Practice & Experience **26** (6/1996) 635-652, `http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/lua.ps.gz` [accessed Mar 2006]
8. International Olympiad in Informatics. `http://ioinformatics.org/` [accessed Mar 2006]

9. Internet Problem Solving Contest. http://ipsc.ksp.sk/ [accessed Mar 2006]
10. Košinár P., Koutný V., Kralovič R.: A linux kernel patch to have exact processor time measurement
http://people.ksp.sk/~misof/ioi/timepatch-2.4.25 [accessed Mar 2006]
11. Koutný V.: Testovací systém pre programátorské sútaže
(in Slovak, English title: A testing system for programming contests).
Master Thesis at Comenius University, Bratislava, 2004.
http://people.ksp.sk/~vlado/cessd2/CESSd2.tar.gz [accessed Mar 2006]
12. Lampson, B.W.: A Note on the Confinement Problem.
Communications of the ACM **16** (10/1973) 613-615.
13. Michalski, M. et al.: SIO.NET Plug&Play Contest System. Presented at Perspectives on Computer Science Competitions for (High School) Students, 2005.
http://bwinf.de/competition-workshop/RevisedPapers/16_Michalski+_rev.pdf
[accessed Mar 2006]
14. TopCoder Algorithm Competitions.
http://www.topcoder.com/tc?module=Static&d1=help&d2=index
[accessed Mar 2006]
15. University of Valladolid Online Contest System. http://acm.uva.es/contest/
[accessed Mar 2006]

B Paper Summary

In this paper we examine currently used programming contest systems. We discuss possible reasons why we do not expect any of the currently existing contest systems to be adopted by a major group of different programming contests.

The main contribution of this paper lies in a detailed analysis of new, specific threats that arise in the context of programming contests. A secure contest system should be able to handle all the threats mentioned in this article. We give a detailed description of each of the presented attack types, and discuss possible ways of preventing the attack.

A list of the attacks discussed in the article:

- Forcing a high compilation time
- Consuming resources at compilation time
- Accessing restricted material
- Misusing the network
- Modifying or harming the testing environment
- Circumventing the time measurement
- Exploiting covert channels
- Misusing additional services
- Exploiting bugs in the operating system
- Obfuscation

We suggest to approach the design of a contest system as a design of a secure IT system, using known methods from the area of computer security. From our point of view the four most important requirements for a contest system that could be adopted by a majority of contest organizers are: flexibility, robustness, user-friendliness, and, most importantly, trust. The end users require a convincing claim about the security aspects of the contest system.