Automatic Problem Generation for Capture-the-Flag Competitions

Jonathan Burket
Carnegie Mellon University
jburket@cmu.edu

Peter Chapman
Carnegie Mellon University
peter@cmu.edu

Tim Becker
Carnegie Mellon University
tjbecker@cmu.edu

Christopher Ganas
Carnegie Mellon University
cganas@cmu.edu

David Brumley
Carnegie Mellon University
dbrumley@cmu.edu

Abstract

Computer security games, especially capture-the-flag (CTF) competitions, are growing in popularity. A typical CTF contest presents users with a set of hacking challenges, where correct solutions reveal a text "flag" that can be submitted to a scoring server. In traditional CTF architectures, the problem and the flag are the same across the competition.

In this paper we discuss automatic problem generation (APG), where a given challenge is not fixed, but rather can have many different automatically generated problem instances. APG offers players a unique competition experience and can facilitate deliberate practice where problems vary just enough to make sure a user can replicate the solution idea. APG also allows competition administrators the ability to detect when users submit a copied flag from another user to the scoring server. In 2014 we ran a large-scale CTF competition called PicoCTF, where we measured the prevalence of flag sharing. Our results indicate that about 0.8% of flags submitted to AGP problems were copied, with 14% of teams submitting at least one shared flag. In 68% of flag sharing cases, teams went on to eventually solve the problem on their own.

1 Introduction

Capture-the-flag (CTF) competitions are exploding in popularity. CTFs challenge players with computer security problems from a wide spectrum of technical areas ranging from cryptography to binary analysis. They are used in a variety of roles, including outreach, education, and prize-based contests.

There are multiple different formats of CTF competitions. In this paper we focus on *Jeopardy*-style contests where players are presented a menu of problems with increasing point values. A correct solution to a problem will reveal a *flag*, which is submitted to the score server for points.

CTFs typically present each team with the exact same set of challenges, where each challenge has the same flag. There are several benefits to this approach, including simplified challenge development and easy testing. However, one significant disadvantage is that teams can copy flags from other teams and submit the copied flags for points. The potential damage for leaked flags is exacerbated in longer competitions where there is more time for players to post solutions that other teams can copy.

Automatic problem generation (APG) techniques create new versions of problems, called *problem instances*, that get distributed among teams. APG can ensure that each team receives a different flag for a given problem, mitigating the threat of copied or leaked flags. There are a variety of ways to automatically generate problems, each with different trade-offs. For example, in one type of APG, every team gets the same challenge, but with a unique per-team flag. This type of APG ensures a consistent problem difficulty across all players, but serves only to protect against flag sharing. A more sophisticated APG may mutate problem instances in a way that preserves the essence of the problem but alters the details needed to solve the challenge. This type of APG offers potential educational benefits, such as the ability to create practice problems, but can be difficult to incorporate into a competition.

In this paper we discuss automatic problem generation techniques that we have investigated in research and deployed in a large-scale competition called PicoCTF [5]. PicoCTF is an online capture-the-flag competition designed to excite and educate middle and high school students about computer science and security. In 2014, PicoCTF had approximately 10,000 eligible players competing on 3,000 teams.

While generating problems automatically is not a new idea [1, 3, 12, 15, 17, 19], APG has yet to be widely adopted in capture-the-flag competitions. With a large number of players and significant prize money at stake, CTFs require an APG solution that focuses on scalability, simplicity, and balanced problem difficulty. In order to meet these requirements, PicoCTF uses problem tem-

plates to generate a pool of problem instances with, at minimum, unique flags per instance. Each team is then assigned a problem instance such that two teams are unlikely to receive the same instance. PicoCTF records when a team submits a correct flag for a problem instance they were not assigned, a strong indicator that the team has copied the flag from another source.

In order to gain more insight into flag copying, including the motivation and behavior of players who copy flags, we have performed a variety of measurements in our 2014 competition. We find that flag copying accounted for 0.8% of submissions to automatically generated problems, with approximately 14% of teams submitting at least one copied flag. Encouragingly, in 68% of flag sharing cases, players who attempted to submit a copied flag went on to solve the problem themselves, suggesting that students who copy flags are still interested in solving the problem and moving along in the game.

Overall, our contributions are as follows:

- We describe using APG in PicoCTF, a large-scale cybersecurity competition. We also detail how our performance, scalability, and balanced problem difficulty goals shaped our APG design choices.
- We present an evaluation of flag sharing in the PicoCTF 2014 competition based on submissions to automatically generated problems, ultimately finding that flag sharing is relatively infrequent, but occurs across many teams. We also find that flag sharing is not strongly correlated with score, and is most common among teams from the same school.
- We have released the code for PicoCTF as an open-source platform for hosting CTFs, which has already been used in several other competitions[1].

2 Background: PicoCTF

In this section we describe PicoCTF, a yearly security competition first hosted in 2013. In prior work [5] we have discussed the high-level design of the competition in more detail, including the role and impact of PicoCTF's interactive game format.

PicoCTF is an online capture-the-flag (CTF) competition for middle and high school students developed by students at Carnegie Mellon University. A *Jeopardy-style* CTF, PicoCTF presents players with a series of security challenges of increasing difficulty in areas ranging from binary exploitation to forensics. A typical PicoCTF challenge requires students to exploit a vulnerability in a binary program or web application, analyze encrypted messages, or explore mangled images and file systems, with the ultimate goal of uncovering a "flag" that can be submitted to receive points and unlock new challenges.

PicoCTF 2014 had 3185 eligible teams consisting of 9,738 students solving 66 challenges over a 12 day period. Students were eligible to win prizes if their team consisted of five or fewer middle or high school students from the United States. PicoCTF 2014 offered monetary prizes of $1,000-$6,000 to the top ten highest scoring eligible teams.

[1] http://hsctf.com, http://tjctf.org

3 Automatic Problem Generation

Traditionally, capture-the-flag competitions feature a set of fixed challenges, which each have a single solution (flag) that is the same for all teams in the competition. This makes problem deployment and scoring straightforward; all competitors receive the same content, and checking for a correct solution is as simple as checking submissions against a single fixed flag.

If, however, we allow different players to receive different versions of each challenge, we open up exciting new possibilities both for cheating prevention and for an improved educational experience. In the simplest case, having different flags for each version of a given problem ensures that two different players cannot trivially share their answers to that problem. More dramatic changes to individual problem instances offer potential educational benefits by allowing players to solve multiple instances for practice or as part of a second competition playthrough.

Our goal in this research is to explore automatic problem generation in the context of capture-the-flag competitions, where issues such as fairness and scalability are especially critical.

3.1 Example Autogen Problem

Both PicoCTF 2013 and 2014 featured a simple problem based on a Caesar Cipher, in which a plaintext message is encrypted by replacing each letter in the message with the letter n further in the alphabet for some integer n.

The version of this challenge in PicoCTF 2014 expands upon the original with the addition of automatically generated (autogen) problem instances. When creating a new instance, the server first generates a unique flag, appending it to the message "thesecretpassphraseis". The server then picks a random encryption key n from the 26 possible values and encrypts the plaintext. The resulting encrypted message is then given to a team as a challenge.

This automatically generated Ceasar Cipher problem is an example of a *templated* autogen problem, one where the flag and potentially other constants in the problem are randomized. Randomized flags and values can be used in many types of problems, including uniquely generated stack canaries, different field orders for a SQL injection problem, etc.

The changes to the problem can also be more significant. For example, one experimental autogen problem we created produces programs with unique sets of input constraints that need to be satisfied to solve each instance.

We call these *synthesized* autogen problems.

3.2 Templated Autogen Problems

Templated autogen problems are challenges where each instance shares a common template but uses a different flag. Templated problems can also vary other parameters in the problem, as with the encryption key in the Caeser Cipher example. The goal of templated autogen challenges is to detect and prevent flag-based cheating.

Flag Copying Prevention and Detection With autogen problems, different competitors do not necessarily see the same version of a given challenge and therefore cannot reliably share flags. Additionally, if competitors are unaware that different players have different problem versions, then it is also possible to use autogen problems to *detect* flag copying. If a team submits a flag that is a solution to a different problem instance than the one they received, then it is likely that the team collaborated with a different team. We discuss how we used these detection mechanisms for PicoCTF 2014 in Section 4.

Note that templated autogen problems help prevent *flag* sharing. Since the core problem is still ultimately the same, however, they do not prevent teams from sharing their *method* for solving a given challenge.

Identifying the Source of Flag Disclosures With more than 10,000 users competing in PicoCTF 2014, it was difficult to ensure that no flags were publicly disclosed before the end of the 12 day competition. Indeed, on multiple occasions, flags for earlier problems in the competition were leaked either on public message boards or video streaming sites.

When competitors leaked flags from autogen problems, however, they were effectively revealing their identity, since each team (with high probability) received a different set of flags. The ability to identify the source of a disclosure allowed us to, in most situations, contact the player and request that they remove the flag, as well as other exposed flags for problems that were not automatically generated. Most large-scale flag disclosures for PicoCTF 2014 were either accidental or a misunderstanding of the competition rules. Participants we contacted were therefore quite responsive and quick to remove the offending flags. In more extreme cases, we used this information to lock the account of the team responsible for the disclosure.

3.3 Synthesized Autogen Problems

Developing the cybersecurity skills associated with capture-the-flag competitions requires practice. At any given time, however, there are a limited number of competitions running, most of which take down their challenges once the competition is over. Further, once a player completes a given set of problems, there is generally little advantage to playing through the competition again. The limited availability of new challenges can make it difficult to gain the practice necessary to improve one's cybersecurity skills.

In the past few years, substantial work has gone into automated generation of practice problems for algebra [19], word problems [15], and even formal logic [1]. Similarly, we expect that APG can be used to generate CTF problems that help students practice security-related skills. However, such autogen problems require more than just flag or constant randomization to be useful for practice.

We have begun developing *synthesized* autogen challenges that change the underlying problem more substantially. We have prototyped a generic reverse engineering challenge where users must provide an input that satisfies sets of constraints of varying complexities, a return-oriented programming challenge where students match gadgets with their function, and a steganography problem where files are hidden in various types of images. Since these problems are more complex, it is harder to ensure that the different instances generated are of similar difficulty (see Section 3.4). For this reason, we did not include these challenges in PicoCTF 2014. Nevertheless, in future competitions we plan to place a greater emphasis on challenges that can be solved multiple times to gain more practice with the relevant skill.

In our experience, it is easier to develop synthesized autogen problems for novice players than it is for more experienced competitors. A good high-value CTF challenge will typically feature something unique: a rare vulnerability, a surprising combination of vulnerabilities that need to be exploited in sequence, or a new twist on a classic problem. In many of these cases, the challenge is discovering the location of the weaknesses in the system and the steps required to exploit those weaknesses. Problems where the principal challenge is to discover the unique twist are fundamentally difficult to duplicate for practice, since the element of surprise feeds into the difficulty of the problem.

PicoCTF problems, however, generally place a much stronger emphasis on *executing* attacks, rather than on discovering what exploits to use. The PicoCTF 2014 challenge `Overflow 1`, for example, explicitly tells the player to perform a standard buffer overflow attack. While such a problem would be trivial to an experienced player, it poses a significant challenge to a student new to cybersecurity. Learning how to use the debugger, how to determine buffer lengths in binaries, and how to construct functional exploits are all skills that require practice. For problems like this where *executing* the attack is the challenge, we expect to be able to generate multiple problem instances where each instance is of comparable difficulty to the original problem. Thus, while synthesized autogen problems may not add much replay or practice value to

competitions for advanced players, we expect it to have a meaningful impact in a competition such as PicoCTF.

3.4 Challenges with Automating Problem Development for CTFs

Automatically generated problems introduce a number of new challenges, particularly for CTFs.

Consistent Difficulty While the primary goal of PicoCTF is education, it is still ultimately a competition, which means that fairness is critical. If each team receives different problem instances, then there is a risk that some problem instances will be significantly easier or harder than others, giving some teams a competitive advantage.

We experienced such an issue with an autogen problem in PicoCTF 2014. The `ZOR` challenge required students to decrypt a message encrypted using a one byte XOR scheme and a randomly selected key from a set of 256 possible values. We observed that for each instance of the problem (of which 100 were generated), the ratio of incorrect to correct submissions was roughly equivalent (on the order of 10:1). For one instance, however, there were more than twice as many correct solutions as incorrect ones. This outlier instance had been created with the randomly selected key "0", causing the encrypted message to be identical to the plaintext. Thus, for 1 in 100 teams, solving the `ZOR` challenge was as simple as copying the value of the flag from the message.

Fortunately, this was a low value problem and this issue therefore had fairly little impact on the competition. It does, however, highlight the importance of giving careful consideration to consistent problem difficulty when designing and testing autogen problems.

Bug Prevention In a traditional CTF competition, problems are relatively easy to test. One can walk through the challenge pretending to be a competitor, then verify that the uncovered flag is accepted for points. This style of testing becomes much more difficult with autogen problems. Testing a large number of problem instances by hand is generally not feasible. Yet, it may be the case that a bug is present in only a handful of the problem instances, such that testing a few example instances does not detect the issue.

We had such a bug in the PicoCTF 2014 challenge `Internet Inspection`. In order to allow for input and formatting differences, most PicoCTF challenges strip out extra spaces and formatting code from user submissions. Figure 1 shows code from the `Internet Inspection` problem used to remove spaces and `<td>` tags from user input. The author of this code, however, misunderstood the behavior of the Python `strip` function, which strips off any individual *characters* provided in the input string. Thus, any flag submission ending in a 't' or a 'd' would be cut off prematurely. Since flags were generated at random,

```
def grade(autogen, key):
    return flag == key.strip().strip('<td>')
    .strip('</td>').lower()
```

Figure 1: Buggy Grader for `Internet Inspection`

this issue affected approximately one in thirteen problem instances.

In our testing before the competition, we solved several instances of each problem, but did not run into this issue due its relative rarity. Fortunately we were able to quickly patch the problem once the issue was discovered about 30 minutes into the competition. Nevertheless, this sort of issue suggests that manual testing alone is insufficient for automatically generated challenges.

Scalability and Deployment Capture-the-flag competitions are often run on a handful of small servers with a limited budget, despite frequently having hundreds or thousands of participants. This means that overhead and scalability are particularly important considerations when adding new features, such as autogen problems.

In our original prototype, problems were generated on-demand. Whenever a player unlocked a new autogen challenge, the server would immediately create a new problem instance before returning the new list of challenges to the user. This worked well for simple challenges and had the advantage that every team received a unique instance. However, as the number of concurrent users solving challenges increased and with more complex autogen problems that could not be created instantly, this problem generation strategy quickly became a bottleneck.

Deployment of autogen problem also presents new challenges. Since different users receive different problem instances, the actual problem content (messages, web pages, program binaries, etc.) needs to be tied in some way to user accounts such that teams only receive the content for their particular problem instances.

3.5 Autogen Implementation in PicoCTF

Our implementation to support automatically generated problems was primarily designed to be simple, scalable, and general-purpose. Rather than generating problem instances on demand, as in our prototype, we instead create a pool with a fixed number of instances n in advance. This has the disadvantage that some teams will receive the same problem instance, but this likelihood can be tuned by changing the value of n. If two teams are *unlikely* to receive the same challenge instance, then this still achieves most of the benefits of having completely unique instances.

By generating problem instances in advance rather than on demand, we gain a number of performance and deployment advantages. Whenever a new team starts the competition or unlocks a new autogen problem, they are

assigned an *instance number* corresponding to one of the existing problem instances in the pool. They then immediately have access to the challenge and all of the relevant content. If necessary, the number of problem instances can always be expanded by simply increasing the size of the instance pool. In addition, specific instances can be revoked or teams can be remapped to new instances in cases where flags for specific instances are disclosed.

When creating an autogen challenge, problem developers define two scripts: a *generator* and a *grader*. Given an instance number and a random seed, the *generator* produces the appropriate problem text and any necessary resource files for that problem instance. Files produced by the generator are automatically deployed by the web server and served to the correct users based on the user's session information.

Graders are responsible for evaluating user submissions to judge whether they are correct for a given problem instance. Most autogen problems in PicoCTF 2014 use deterministic flags based on a hash of the instance number and a per-problem secret. That way, both the generator and grader have access to the value of the correct flag without needing to read from either the database or the file system.

Since our primary emphasis was on templated autogen problems, we implemented a template system for introducing generated constants and flag values into existing files, such as programs or web pages. In most cases, this is as simple as performing a search and replace, swapping out variables in the template file with the per-instance value determined by the generator script. This templating system works particularly well for problems with simple, static deployment, such as basic cryptography, web-based problems, and problems that provide users with downloadable files.

In contrast, problems that feature interactive, exploitable services are currently deployed by hand, as they require more per-instance setup and are often served from a different machine. For future competitions, we plan to expand our autogen tools to better automate deployment of these more challenging problem types.

All of our code used both for autogen problems and to run the PicoCTF competition is open-source and can be downloaded at `https://github.com/picoCTF/picoCTF-Platform-2`.

4 Flag Sharing in PicoCTF 2014

By including numerous automatically generated challenges in PicoCTF 2014, we were able both to evaluate our technique at scale and to gather data on the prevalence of *flag sharing* in the competition. Students competing in PicoCTF are encouraged to play on teams of up to five students and to discuss the challenges with their teammates. All members on the same team see the same problems, and a correct submission from any team member counts as a correct submission for the whole team. There is no penalty or limit for incorrect solutions. While students may look up information from outside sources, the sharing of flags, the final solutions that are submitted for points, is against the rules of the competition.

By analyzing submissions to automatically generated problems, we were able to gain insight to how frequently flags were shared across teams in the 2014 competition. For the sake of our analysis, we say that flag sharing has occurred if a team submits a flag for an autogen problem that is *incorrect* for that team's instance but *would be correct* for some other team's instance. For example, all of the flags for the PicoCTF 2014 challenge `Substitution` are the names of Disney songs. If a team receives a problem instance for which the answer is "Let it Go", but submits the flag "A Whole New World", a solution to a different problem instance, then this is labeled flag sharing. Note that we do not detect cases where teams share a flag and also happen to have the same problem instance.

Autogen Problems in PicoCTF 2014 PicoCTF 2014 featured 10 different templated autogen challenges. As shown in Table 1, the problems ranged from extremely simple (solved by nearly every team in the competition) to far more challenging (solved by only 1% of participating teams). Over the course of the 12 day competition, eligible players submitted a total of 127,412 flags to these challenges, 16,604 of which were correct. We created 100 instances of each autogen problem, with instances assigned to teams randomly. As discussed in Section 3.3, PicoCTF 2014 did not contain any synthesized autogen problems.

4.1 Flag Sharing Statistics

Among the 127,412 submitted flags, 1081 (0.84%) were labeled as "shared flags". Removing multiple incorrect attempts by the same team on a given problem yields 530 cases of a team attempting to submit an incorrect flag that would be correct for a different problem instance. Attempts were widely spread among teams, with 460 (out of 3185) having at least one flag sharing attempt. Flag sharing occurred across all score levels, though not among any team that won prize money.

Interestingly, as shown in Table 1, the number of shared flag cases varies dramatically across autogen problems and is not obviously correlated with problem difficulty. We suspect that the frequency of flag sharing reflects the difficulty of sharing a given flag versus the benefit of not having to solve a given challenge. Consider the problem `No Comment`. To solve this simple challenge, players must look through the source of a web page to find a hidden flag. The randomly generated flag is extremely long ("flag_0d326ecce064e8153e743f1d927d3c15313f66ec"

Problem Name	Category	Score	Teams Solved	Shared Flag Cases	Eventual Success Rate
Tyrannosaurus Hex	Miscellaneous	10	3210	79	98.7%
No Comment	Web	20	2952	3	100%
Caesar	Cryptography	20	2684	7	85.7%
Internet Inspection	Web	30	2704	17	82.4%
Javascrypt	Web	40	1719	124	75.0%
Substitution	Cryptography	50	1677	165	69.1%
ZOR	Cryptography	50	554	36	38.9%
Basic ASM	Reversing	60	887	43	58.1%
Repeated XOR	Cryptography	70	182	56	25.0%
Block	Cryptography	130	35	0	-

Table 1: Automatically Generated Problems in PicoCTF 2014

in one instance), making it nearly impossible to share verbally with another team. Since the problem itself is relatively easy and sharing the flag is difficult, it is not surprising that only three flag sharing attempts were detected for this problem.

On the other hand, the challenge `Substitution` requires students to solve a substitution cipher to uncover the lyrics of a Disney song, the name of which serves as the flag for the problem. In this case, the problem is more challenging, and sharing the value of the flag is trivial. It may even be possible to overhear another team discussing the problem and infer the value of the flag without the cooperation of the other team. When the song title is revealed by decrypting the message, it contains no spaces ("letitgo", for example), and thus most students (88.5%) submit the song title with no spaces. In flag sharing cases, however, submissions were four times more likely to initially contain the song title *with* spaces ("let it go"), suggesting the solutions were often shared via word of mouth.

4.2 Sources of Shared Flags

In order to better understand how shared flags were being used in the competition, we attempted to manually determine the origin of the shared flag for each of the 530 cases of flag sharing. We ultimately observed that most instances (75%) could be classified as either an attempt to submit a publicly disclosed flag or an attempt to submit a solution that worked for a different team at the same school. We were unable to determine the source of the copied flag in the remaining 25% of cases.

Disclosed Flags Given the length and scale of the PicoCTF, it was difficult to ensure that flags were not leaked before the end of the competition. By looking at submissions to autogen problems, however, we were able to determine both when a leak had occurred and how much of an impact it was having on the competition. Since autogen problems are randomly assigned to teams, we observed that the number of submissions (both correct and incorrect) was similar for different instances. Cases

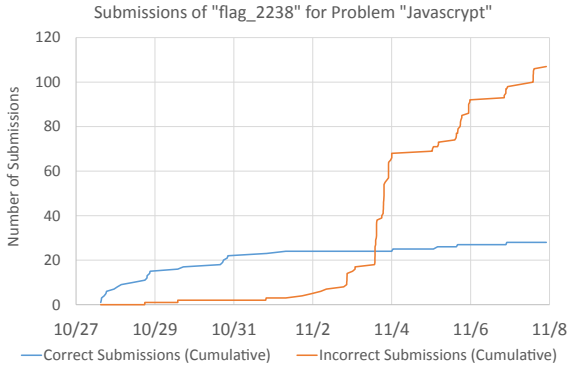

Figure 2: Submission Patterns Before and After a Flag Disclosure

of shared flag submissions were also distributed fairly evenly among instances, in terms of which instance was the source of the original flag. If, however, a large number of submissions labeled as attempts to use a shared flag all used a flag from one instance, this almost always indicated that a public flag disclosure had occurred.

On November 1 (five days into the competition), one player began posting videos of his solutions to some of the lower-valued challenges on YouTube. Among these solutions was the flag "flag_2238" for the `Javascrypt` problem. As shown in Figure 2, the frequency of both correct and incorrect submissions of the flag "flag_2238" followed a fairly clear pattern until November 1, at which point the number of attempts to use "flag_2238" on the wrong instance skyrocketed.

The other major flag disclosure we were able to detect with autogen problems was for the problem `Repeated XOR`. Several days into the competition, a student asked for assistance solving the problem on the question-answer forum Stack Overflow. Several other users replied, eventually posting the decrypted message and corresponding flag. Of the 56 cases where shared flags were submitted for `Repeated XOR`, 47 were attempts to submit this flag. Only 13% of teams who submitted this flag were able to

eventually solve the challenge themselves before the end of the competition.

Collaboration Between School Teams Flag sharing among teams from the same school accounted for 71% of the cases for which we were able to identify a likely flag source. We classified an invalid autogen submission as a *shared flag from the same school* if the same flag was accepted earlier (or up to ten minutes later) as a correct answer by a different team from the same school.

Unlike with publicly disclosed flags, which were submitted at arbitrary times after the disclosure, there were clear patterns in the timing of shared flag submissions among teams from the same school. As detailed in Table 2, a third of all shared flags were submitted within 20 minutes of the original, with 13% of submissions occurring within the same minute. Another third of the flags were submitted fairly evenly throughout a one day period and a final third were spread out over the rest of the competition.

This submission pattern suggests that flag sharing between students from the school took several different forms. Both from submission patterns and from discussions with teachers, we know that many classrooms play through PicoCTF as a classroom activity, with multiple teams from the same school competing at the same time. It is therefore not surprising to see that in some cases teams in close proximity shared solutions, especially if the teams were playing for more for fun than for the sake of competition.

Cases where students submitted shared flags significantly later than the original are harder to reason about with solution data alone. It may be the that students discussed problems with other teams after being stuck on a problem for a significant amount of time. From discussions with teachers, we know that some cases of non-immediate flag sharing occurred because of overlap between team members. In one situation, groups of students played PicoCTF 2014 as part of a classroom activity. Then, once class was over, a subset of the most interested students formed a new team to keep playing after school. In this case, the instances of flag sharing were actually students attempting to reuse flags they themselves had obtained on different teams.

4.3 Discussion

The prevalence of flag sharing in PicoCTF highlights the difficulty of preventing flag sharing in an extended, online-only competition. There is still good reason to be optimistic about the integrity of the competition, however. Despite the large number of teams that shared flags, the number of actual attempts to submit copied flags was relatively small. Increasing the number of autogen problems should further reduce the effectiveness of flag sharing.

We were surprised to see that, in addition to low-scoring teams, high-scoring teams also shared flags. We had originally thought that competitive players would not share flags, as anyone sharing flags would likely be unable to reach the higher levels of play through problem unlocking. In PicoCTF 2014, we did not award prize money to any teams that shared flags, however we did allow teams suspected of flag sharing to remain on the scoreboard. For future PicoCTF competitions, we plan to make our policy about shared flags more explicit and then treat cases of shared flag submissions among higher scoring teams far more seriously.

Many of the factors we found to be associated with flag sharing are relatively unique to PicoCTF. The length of PicoCTF (12 days) made limiting online flag disclosures especially challenging. In addition, the fact that multiple teams frequently came from the same school seemed to encourage these teams to share flags. Finally, since most PicoCTF competitors are novices (70% cited no "hacking" experience in our 2013 survey [5]), we suspect some may not take flag sharing as seriously as more experienced players or may be less familiar with proper CTF etiquette. Thus, while competitions similar to PicoCTF may experience the same sorts of flag sharing behavior, we suspect that flag sharing in other competitions is concentrated among fewer teams.

5 Related Work

The idea of giving different tests to different students has long been a technique to prevent cheating for in-person multiple choice tests [4, 8–11]. While different test versions are traditionally generated by permuting the order of problems and possible answers for each question, there have been instances where test-givers have used small problem differences to detect cases where students are collaborating [4]. These techniques have largely been successful both in preventing and deterring cheating on multiple-choice tests [6, 11].

With the growth of online education, and in particular massive open online courses (MOOCs), there have recently been efforts to develop cheating prevention techniques for *online* tests. Many of the methods that have been proposed rely similar randomization strategies similar to those used with paper-and-pencil tests [2, 7, 14, 16, 18]. Tools for supplementing in-person classes with digital evaluations, such as learning management systems (LMSs), often offer support for either problem-level or test-level randomization [18].

Khan Academy features a problem framework called *khan-exercises* [12] that allows problem writers to create autogen problems based on templates. These templates have similar format to templates we used for PicoCTF problems, however while the primary goal of templated autogen problems in PicoCTF is to detect and prevent cheating, Khan Exercises focuses more on generating

Time Between Submissions	Number of Cases	Cumulative Percentage
<10 Minutes After	74	33%
<1 Hour	18	41%
<1 Day	67	70%
<2 Days	18	78%
Rest of Competition	49	100%

Table 2: Time Between Original Flag Submission and Shared Flag Submission for Teams from the Same School

large numbers of practice problems.

Within the CTF community, generating custom problem instances has not yet taken off. Besides PicoCTF, we are only aware of several challenges in the Embedded Security CTF [13] that are automatically generated.

Finally, while autogen challenges in PicoCTF require a fairly detailed specification of how to generate problem instances, recent work has gone into more advanced types of problem synthesis, where new questions can be generated based on examples or a desired difficulty level [1, 3, 15, 17, 19].

6 Conclusion

Automated problem generation has great potential to improve both the effectiveness and integrity of computer security competitions. With basic forms of APG, we were able to prevent numerous cases of flag sharing and gain a better understanding of inter-team collaboration over the course of the competition. Generating problem instances automatically in a competitive setting also comes with new challenges, particularly with regards to scalability, deployment, and competition fairness. In future PicoCTF competitions, we plan to expand our APG efforts, both by increasing the percentage of problems generated automatically and by experimenting with methods to improve the replay value of generated challenges.

Acknowledgments We would like to thank Roberto Valle, Maxime Serrano, and Collin Petty for their work developing autogen tools and problems. We would also like to thank each of our sponsors for making PicoCTF possible: Trend Micro, Boeing, Qualcomm, the Entertainment Technology Center, CloudShark, and the National Security Agency. This material is based upon work supported by the National Science Foundation under Grant No. 1419362 and by a Graduate Research Fellowship under Grant No. 0946825.

References

[1] Ahmed, U. Z., Gulwani, S., and Karkare, A. Automatically Generating Problems and Solutions for Natural Deduction. In *International Joint Conference on Artificial Intelligence* (2013).

[2] Anderman, E. M., and Murdock, T. B. *Psychology of Academic Cheating*. Elsevier Academic Press, 2007.

[3] Andersen, E., Gulwani, S., and Popovic, Z. A Trace-based Framework for Analyzing and Synthesizing Educational Progressions. *SIGCHI Conference on Human Factors in Computing Systems* (2013), 773–782.

[4] Anderson, W. A. Cheating Control on Exams. *Journal of Agronomic Education 10* (1981).

[5] Chapman, P., Burket, J., and Brumley, D. PicoCTF : A Game-Based Computer Security Competition for High School Students. In *USENIX Summit on Gaming, Games, and Gamification in Security Education* (2014).

[6] Cizek, G. J. *Cheating on Tests: How To Do It, Detect It, and Prevent It*. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 1999.

[7] Cluskey, G. R., Ehlen, C. R., and Raiborn, M. H. Thwarting Online Exam Cheating Without Proctor Supervision. *Journal of Academic and Business Ethics 4* (2011).

[8] Cook, L. L., and Eignor, D. R. An NCME Instructional Module on IRT Equating Methods. *ITEMS • Instructional Topics in Educational Measurement*, 1984 (1991), 37–45.

[9] Harpp, D. N., and Hogan, J. J. Crime in the Classroom: Detection and Prevention of Cheating on Multiple-Choice Exams. *Journal of Chemical Education 70* (1993), 306.

[10] Houston, J. P. Alternate Test Forms as a Means of Reducing Multiple-Choice Answer Copying in the Classroom. *Journal of Educational Psychology 75*, 4 (1983), 572–575.

[11] Kerkvliet, J., and Sigmund, C. L. Can We Control Cheating in the Classroom? *The Journal of Economic Education 30*, 4 (1999), 331–343.

[12] Khan Academy. The Khan Academy's Exercise Framework. http://khan-exercises.readthedocs.org/, 2011.

[13] Matasano Security and Square, Inc. Embedded Security CTF. https://microcorruption.com, 2014.

[14] Meyer, J., and Zhu, S. Fair and Equitable Measurement of Student Learning in MOOCs: An Introduction to Item Response Theory, Scale Linking, and Score Equating. *Research & Practice in Assessment 8* (2013), 26–39.

[15] Polozov, O., O'Rourke, E., Smith, A. M., Zettlemoyer, L., Gulwani, S., and Popović, Z. Personalized Mathematical Word Problem Generation. *International Joint Conference on Artificial Intelligence* (2015).

[16] Rowe, N. C. Cheating in Online Student Assessment : Beyond Plagiarism. *Online Journal of Distance Learning Administration 7* (2004).

[17] Sadigh, D., Seshia, S., and Gupta, M. Automating Exercise Generation: A Step Towards Meeting the MOOC Challenge for Embedded Systems. *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education* (2012).

[18] Sewell, J. P., Frith, K. H., and Colvin, M. M. Online Assessment Strategies : A Primer. *MERLOT Journal of Online Learning and Teaching 6*, 1 (2010), 297–305.

[19] Singh, R., Gulwani, S., and Rajamani, S. Automatically Generating Algebra Problems. *AAAI Conference on Artificial Intelligence* (2012), 1620–1627.