Toward a Field Study on the Impact of Hacking Competitions on Secure Development

Daniel Votipka and Michelle L. Mazurek  
University of Maryland  
dvotipka,mmazurek@cs.umd.edu

Hongyi Hu and Bryan Eastes  
Dropbox, Inc.  
hhu, bryane@dropbox.com

ABSTRACT
The ability to find and fix vulnerabilities is critical to producing secure software. Previous research has shown that the main difference between experts who specialize in finding security flaws and general software practitioners (i.e., developers and testers) is that experts have been exposed to more security issues. To bridge this experience gap, computer security competitions, called Capture-the-Flags (CTF), have been carried out both in the academic and corporate setting. Using a mixed-methods approach, we examine in a field setting whether CTF competitions improve participants’ ability to identify security weaknesses and write more secure code. Our initial results indicate that CTFs have a positive effect on security thinking, encourage communication with the security team, and reduce overconfidence in participants’ ability to handle complex security problems.

1. INTRODUCTION
Secure development is the cornerstone of cybersecurity: If developers fail to correctly implement a security protocol, all guarantees provided by the protocol are void. Even if an organization correctly establishes its network defenses, all their effort is wasted if their software is vulnerable.

Significant advances have been made in the automation of vulnerability discovery tasks, but human intelligence is still required to identify the most complex flaws [3][5][11][21][25][27][29][30]. Because of this, companies often rely on multiple levels of code review to manually identify vulnerabilities; this can include review within the development team for a given component, review by a dedicated security team, and even hiring outside experts (e.g., penetration testing, bug bounties) to examine code. Within-team developers typically work on the targeted code base and are well positioned to fix any identified vulnerabilities, but may lack security expertise [16][23]. Security teams, in contrast, must concern themselves with all areas of a codebase, meaning they may not have time to review everything, or may not understand some nuances of a particular segment [23][24]. In addition, the later in the process that a vulnerability is identified, the more difficult and expensive it can be to fix [7][18][22][28][31][35].

This situation suggests that it is critical to arm developers with the ability to identify and fix vulnerabilities — or better, avoid them in the first place. One commonly suggested approach is to have security champions within software teams to identify vulnerabilities, disseminate knowledge to their teammates, and escalate issues to the dedicated security team [17][20]. This leads to a key question: how can companies obtain security champions? Hiring existing security experts is one obvious possibility, but such experts are relatively scarce and may not want to be embedded within a product team [10][11].

An alternative is to train existing developers to become security champions. How to conduct this training effectively remains an open question [9][26]. Our prior research suggests that exposure to a broader variety of vulnerabilities is a critical component of developing vulnerability-finding expertise [33]. One common way that experts report gaining this experience is through hacking competitions known as Capture-the-Flags (CTFs). While public CTFs have long been sponsored by companies for recruitment, many organizations have increasingly begun to hold internal CTFs to train their own employees [8].

Unfortunately, little is known about the efficacy of internal CTFs in improving secure-development outcomes. Prior work has investigated the educational value of CTFs in academic settings [12][13][15][19][32][34], finding that CTFs provide valuable, immediate feedback to learners. To our knowledge, however, there has been no investigation of the effects of CTFs on secure development practices in a real-world setting: are CTF participants able to translate what they learn from the experience into more secure product development?

In this paper, we take a first step toward addressing this question. We describe a pilot mixed-methods study, carried out in conjunction with an internal CTF at Dropbox, an enterprise software company with over 1800 employees supporting a 23 MLOC codebase. Our study investigates not only whether CTF participants perform better in vulnerability-finding and fixing, but also whether they are more likely to recognize potential problems during development. We found several trends which seem to indicate that CTF participants perform better in vulnerability-finding and fixing, but also whether they are more likely to recognize potential problems during development. We found several trends which seem to indicate that CTFs have a positive influence on the way participants consider security. Specifically, we observed that CTF participants considered less-intuitive classes of vulnerabilities and were more comfortable reaching out to the security team for help. Interestingly, we also observed that the CTF appeared to reduce potentially harmful overconfidence in participants’ ability to produce secure code. These pilot results will inform a future full-scale study in the same environment.

2. DROPBOX’S CAPTURE-THE-FLAG
CTFs are attack-oriented competitions where teams work to capture a “flag” — some secret — by exploiting a vulnerability in a small target program. Dropbox uses a jeopardy-style competition, in which challenges are created in several different categories. Pedagogically, this allows participants to switch categories if they get stuck, and ascending problem difficulty within each category also gives participants a
after they committed a change to Dropbox’s codebase. To
For this sub-study, we sent participants a 10-minute survey
ingistered short surveys over time, support better participant
recall than asking participants to remember past events in
in which participants complete a series of randomly admin-
We used the experience-sampling method (ESM) to inves-
tigate when and how participants considered security when
3.1 Diary study
We used the experience-sampling method (ESM) to investi-
gate when and how participants considered security when
making changes to the Dropbox codebase. ESM studies, in
which participants complete a series of randomly admin-
istered short surveys over time, support better participant
recall than asking participants to remember past events in
one cumulative survey [6].
For this sub-study, we sent participants a 10-minute survey
after they committed a change to Dropbox’s codebase. To
avoid fatigue, we only sent invitations in response to their
first two commits each week. Each survey asked participants
to answer the following questions regarding their commit:
• What potential problems were considered (i.e., secur-
ity, functionality, or performance).
• Why did you consider these problems (e.g., standard
practice, recommended by a teammate).
• How did you or did you not resolve the issue (e.g., refer-
cenced company documents, consulted security team).

3.2 Knowledge assessment
The knowledge assessment sub-study was structured as a
 quasi-controlled experiment. Participants were given one
hour to compete a two-part secure development test. In Part
1, participants were asked to write a secure function that
passed untrusted input to a command-line utility. In Part
2, participants were asked to find and fix four vulnerabilities
that we deliberately added to an isolated, non-production
branch of the Dropbox codebase. These vulnerabilities —
two CSRF bugs, one XSS injection bug, and one authoriza-
tion bug — were similar to real issues previously identified
by the Dropbox security team.

The knowledge assessments were scored by Dropbox secu-
ritv experts. In Part 1, participants were awarded one point
for writing functional code that was not vulnerable to com-
mand injection. In Part 2, participants were given one point
for each correctly identified vulnerability and an additional
point for correctly fixing it, for a maximum of 8 points. Par-
ticipants were awarded half a point for identifying where the
vulnerability existed or providing a general idea for fixing it.
We administered two versions of the knowledge assessment:
one prior to the CTF and one six weeks afterward. This
allowed us to measure the withinsubjects effect of the CTF,
while accounting for medium-term knowledge retention.
Finally, because of the deception used in the diary study,
we did not initially tell knowledge assessment participants
about UMD’s involvement, to avoid exposing the true pur-
pose of the diary study. Knowledge assessment data was
not shared with UMD researchers until participants were
informed and gave consent to share.

3.3 Security metrics
We also collected secure-development behavioral metrics from
Dropbox for all participants during the study.

To avoid biasing responses, we used deception: we initially
told participants the study’s purpose was to understand gen-
eral software development practices, and we asked distractor
questions about functionality and performance bugs as well
as security. At the end of the six week period, we debriefed
participants regarding the true nature of the study and gave
them the option to have their responses deleted. Only one
participant elected to leave the study after the debrief.

3. METHODOLOGY
To understand how experience finding and exploiting secu-
ritv issues in a CTF affects secure development behaviors, we
developed a mixed-methods pilot study around Dropbox’s
CTF. The study, conducted from October 2017 to February
2018, focused on two research questions:
RQ1: Does participation in a CTF improve par-
ticipants’ recognition of security issues?

RQ2: Does participation in a CTF improve par-
ticipants’ ability to prevent security issues?

Recognition of a security issue can occur during the design
phase, while writing code, or after code has already been
written (i.e., identifying a vulnerability during code review
or in deployed code). Even if developers are not able to
fix the issue, any improvement in recognition is still useful,
as many large companies have internal security teams that
can be consulted to help solve these problems. Preventing
security issues can include writing secure code, developing
test plans to ensure the code is protected against a variety of
attacks, and communicating with security experts to design
acceptable solutions.

Due to the complexity involved in answering these questions
and our quasi-experimental setting, we piloted a wide variety
of both constructed and field measures, to maximize both
internal and external validity, described below. Figure 1
gives an overview of study events. This study was approved
by UMD’s institutional review board.

3.1 Diary study
We used the experience-sampling method (ESM) to investi-
gate when and how participants considered security when
making changes to the Dropbox codebase. ESM studies,
in which participants complete a series of randomly admin-
istered short surveys over time, support better participant
recall than asking participants to remember past events in
one cumulative survey [6].

Figure 1: Survey procedure. Diary study components are
shown above and knowledge assessment components are be-
low the center timeline. Participants could take part in one
or both of the sub-studies.

Challenges were based on the Dropbox codebase, focusing
on vulnerabilities previously seen in that code. This allowed
participants to focus on learning security concepts, rather
than spending effort learning to use unfamiliar programming
languages or APIs. This avoids a common problem with
public CTFs [13,33].

Finally, because of the deception used in the diary study,
we did not initially tell knowledge assessment participants
about UMD’s involvement, to avoid exposing the true pur-
pose of the diary study. Knowledge assessment data was
not shared with UMD researchers until participants were
informed and gave consent to share.

3.3 Security metrics
We also collected secure-development behavioral metrics from
Dropbox for all participants during the study.

Before any commit is added to the Dropbox codebase, a
series of security-specific static analyses are performed to
check for common security anti-patterns and/or changes to
security-sensitive code. A match will trigger a blocking re-
view by an engineer on the Dropbox product security team.
We collect the number of these security checks that par-
ticipant commits fail during the six weeks after the CTF.
While these analyses are simplistic, matches on security antipatterns provide a lower-bound approximation for the number of basic vulnerabilities introduced by each participant.

Additionally, we monitored the number of times that participants contacted the security team (via well-known, official channels) during the six weeks after the CTF. Our goal was to understand whether the CTF helped participants know when they should contact security experts, even if they did not know the exact security implications.

3.4 Recruitment
For the knowledge assessment, we invited all Dropbox CTF participants (the CTF was advertised to all Dropbox employees). For the diary study, we invited all employees who actively contribute to the Dropbox codebase, whether or not they had participated in the CTF. All recruitment messages were sent by Dropbox managers, in order to provide legitimacy and demonstrate leadership approval. To show appreciation for their time, knowledge assessment participants were given Dropbox CTF t-shirts and diary-study participants were entered in a raffle for five Dropbox jackets. Participants were explicitly permitted to complete all study tasks during work hours.

3.5 Limitations
For this pilot study, we employed a variety of constructed and field metrics with different benefits and drawbacks. For example, our field metrics do not account for the fact that participants working in very different areas of the codebase may encounter security-relevant issues with very different frequencies. While each metric is itself insufficient, we believe that taken together they provide a reasonable range of internal and external validity.

The sample size for our pilot was very small, and all results we report should be considered preliminary at best. We plan to use these observations, and our experiences conducting this study, to inform the design of a larger follow-up.

4. PRELIMINARY RESULTS
We next describe the results of our pilot; due to limited sample size, we do not draw statistical conclusions, but instead highlight trends to be explored in the follow-on work.

4.1 Diary study
A total of 28 Dropbox employees participated in the diary study, 12 of whom also competed in the CTF. During the six-week study, 169 diary responses were submitted.

Overall, security was rarely considered when making functional codebase changes. Participants reported considering security in only 17 of 124 such changes. However, CTF participants were slightly more likely to report considering security (19% of changes) than non-CTF participants (13%).

We also observed differences in CTF participants’ approach to secure development, as follows.

Logic-related vulnerabilities were considered by all participants. Figure 2(a) shows the types of vulnerabilities participants reported considering, among commits when they considered security at all. Both groups considered logic flaws most often (57% CTF, 60% non-CTF), and authorization bugs were also fairly common (42%, 20%). Local file disclosure, which also relates to system logic, was second-most-common among non-CTF participants (30%). We hypothesize that these types of logic-based vulnerabilities are most natural for participants to consider without training.

CTF participants considered vulnerabilities seen in the CTF. Conversely, CTF participants occasionally reported considering generic vulnerabilities (i.e., not specific to the implemented functionality) demonstrated in CTF problems, like SQL injection, CSRF, and XSS (14% each), while the non-CTF participants did not.

CTF participants adopted an adversarial perspective. Figure 2(b) shows the distribution of reasons our participants considered security. Non-CTF participants considered security mostly because it was a “standard practice” (80%) or a teammate recommended it (40%). CTF participants similar cited standard practice frequently (71%). However, CTF participants were more likely than non-CTF participants to cite adversarial reasons, such as use of sensitive data (42% vs. 20%) or considering what a hacker might do (20% vs. 10%). We hypothesize that because the CTF is inherently an offensive exercise, it may teach participants to think more like an adversary.

No participants thought about security based on a tool’s output. Several participants had commits automat-
CTF participants were less confident. All participants were confident in their actions, but feel more comfortable making low-risk judgment calls.

CTF participants better understand security issues and feel more confident in the face of complicated security problems. Prior work suggests that it can be very easy for people to overestimate the security of their code [1]; considering security, than to find and fix security bugs (48%). Injection flaws were the most difficult to find and fix (mean score 0.46/2), followed by authorization bugs (0.61/2) and then CSRF (average score 2.22/4).

CTF participants reviewed system documentation more often. Figure 2(c) shows actions taken by participants to prevent or remedy vulnerabilities. Non-CTF participants mostly reached out to their teammates for help (50%), relied on prior experience (50%), or contacted Dropbox experts (30%). CTF participants primarily reported reaching out to experts at Dropbox (29%), reviewing external documentation (29%), or relying on previous experience (29%). We suspect this is because the CTF exposed participants to the Dropbox security team and may have incentivized them to explore system documentation to address unfamiliar problems.

Interestingly, only one respondent, a CTF participant, said they did not address the security problem. One of the authors inspected the associated commit and saw that the vulnerability only had a minor security impact. It is likely the participant felt a later audit was sufficient. This could indicate that CTF participants better understand security issues and feel more comfortable making low-risk judgment calls.

All participants were confident in their actions, but CTF participants were less confident. When asked how confident they were that the remedial action taken was sufficient, all participants replied either “Very confident” (76%) or “Confident” (24%). Interestingly, a slightly higher portion of Non-CTF participants reported being “Very confident” (80% to 71%). As we discuss later, a small reduction in confidence in the face of complicated security problems can be considered a beneficial result.

**4.2 Knowledge Assessment**

We had 20 participants in the knowledge assessment sub-study, with seven completing both the pre- and post-CTF assessments. The median change in overall score was 1 (mean 1.36). Figure 3 shows each participants’ change in assessment scores compared to their CTF scores. Interestingly, all our participants who scored above the mean CTF score (blue vertical line) had an above-mean (red horizontal line) improvement in their assessment score. The one exception to this trend earned the maximum score on both assessments.

Participants generally found it easier to write secure code (88% of all responses were correct), at least when primed to consider security, than to find and fix security bugs (48%). Injection flaws were the most difficult to find and fix (mean score 0.46/2), followed by authorization bugs (0.61/2) and then CSRF (average score 2.22/4).

**4.3 Security Metrics**

Among the 35 individuals who completed either of our sub-studies, we collected 30 flagged, potentially vulnerable commits and six distinct communications with the security team.

Non-CTF participants commits were flagged slightly more often. Among Non-CTF participants, two of 17 submitted at least one piece of code flagged as potentially vulnerable, while only one of 18 CTF participants’ code was flagged. This could indicate that the CTF had an impact on participants’ ability to write secure code. However, due to the low number of events, further investigation of the associated commit is required to determine whether this was a false positive.

CTF participants flagged potential security problems more often. Four of the six instances where participants reached out to the security team were initiated by CTF participants, and all were examples of pointing out a potential vulnerability requiring expert review. In contrast, the two communications initiated by non-CTF participants were a request to review automatically flagged code and a request for help setting up an ssh connection respectively.

5. DISCUSSION

Despite small samples that prevent firm conclusions, the pilot study revealed several positive trends that can be evaluated in follow-up work. The CTF seems to have some positive effects in improving security thinking, exposing participants to less-intuitive classes of vulnerabilities (such as injection bugs), and helping participants be more comfortable with the security team and with system documentation. (Prior work suggests that good documentation can improve security outcomes [2].) Perhaps most importantly, the CTF seemed to reduce potential overconfidence in the face of complex security issues. Prior work suggests that it can be very easy for people to overestimate the security of their code [1]; from the perspective of the Dropbox security team, getting participants to stop, think about security, and ask for help when needed would in itself be an important achievement.

To validate these trends, we will need to recruit a larger sample in conjunction with the next Dropbox CTF. As with many field studies, some of the logistical complexities became apparent only in retrospect. Based on our pilot, we have identified several potential changes to the study which will help us maximize recruitment and obtain meaningful results in a future iteration. One example of this can be found in the security metrics collected. While code matching the anti-patterns we tracked could introduce catastrophic vulnerabilities, in practice, the false-positive rate was high enough that failing these checks was not likely to actually indicate insecure coding practices.

6. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback; the University of Maryland IRB for their prompt review and management of our ethics review; and Jessica Chang for contributing to recruiting and organizing the study.
7. REFERENCES


