Lessons Learned from Using an Online Platform to Conduct Large-Scale, Online Controlled Security Experiments with Software Developers

Christian Stransky*, Yasemin Acar†, Duc Cuong Nguyen*, Dominik Wermke†,
Elissa M. Redmiles‡, Doowon Kim‡, Michael Backes*, Simson Garfinkel×+,
Michelle L. Mazurek‡, Sascha Fahl†
*CISPA, Saarland University; †Leibniz University Hannover; ‡University of Maryland
×U.S. Census Bureau; +National Institute of Standards and Technology

Abstract

Security and privacy researchers are increasingly conducting controlled experiments focusing on IT professionals, such as software developers and system administrators. These professionals are typically more difficult to recruit than general end-users. In order to allow for distributed recruitment of IT professionals for security user studies, we designed Developer Observatory, a browser-based virtual laboratory platform that enables controlled programming experiments while retaining most of the observational power of lab studies. The Developer Observatory can be used to conduct large-scale, reliable online programming studies with reasonable external validity. We report on our experiences and lessons learned from two controlled programming experiments (n>200) conducted using Developer Observatory.

1 Introduction

While software developers have specialized technical skills, they are rarely security experts [2, 1]. Indeed, developers frequently make errors for many of the same reasons that end users do: because they are uninformed, because they prioritize other requirements over security, and because tools for achieving security are difficult to use correctly. Consequently, identifying and solving security problems faced by software developers and other professionals promises to have a high impact on the overall security of software ecosystems [3].

Prior work observing or surveying software developers and other IT professionals is often limited by small sample sizes, as dictated by the high effort and cost necessary to recruit these professionals. To remedy the cost and difficulty of recruiting developers for security studies, we developed a browser-based virtual laboratory platform, Developer Observatory, to allow for controlled, geographically distributed experiments. In doing so, we wanted to retain as many as possible of the benefits of a lab study: controlling the programming environment, collecting in-depth data about when developers struggle and the strategies they use to solve problems, and administering exit questionnaires.

In this paper, we describe a framework that allows researchers to conduct online secure-programming studies with remote developer participants. This framework includes mechanisms for randomly assigning participants to conditions, as well as the ability to administer surveys during and after study tasks. Our design also takes into account security (from misbehaving participants), participant privacy, and adaptability for a broad range of similar studies. Here, we report on our experiences instantiating and conducting experiments using our implementation of this framework: Developer Observatory.

Developer Observatory is instantiated using Jupyter Notebooks (for in-browser Python interpretation), Amazon EC2 (to isolate participants from each other and scale with demand), and the Qualtrics online survey platform. Developer Observatory allows for in-browser code editing so that participants can write and test code without installing anything, while also enabling researchers to collect detailed information about participant's activities. We successfully deployed Developer Observatory as part of two large-scale controlled experiments [1, 4], in which 2 396 developers interacted with our system and 563 completed an entire study run.

We make the following concrete contributions:

- We describe requirements for a scalable online secure-programming research tool.
- We develop a framework to meet these requirements and implement this framework in a complete study infrastructure (Developer Observatory), which we publicly release to the research community.
- We report on the benefits, challenges, and lessons learned when we used Developer Observatory for two large-scale studies of software developers.

2 Related work

In order to make empirical studies in security easier to conduct and reproduce, a number of researchers have developed measurement systems and environments. The DETER project, for example, was created to support testing cybersecurity technology at scale, mimicking real-world deployments [5]. Other network-security testbeds support distributed networking research; Siaterlis and Masera provide a survey [11].

Relatively few such measurement environments, however, exist for human subjects research. A number of researchers have proposed guidelines for conducting empirical human-subjects experiments in computer science generally [12, 6], including an emphasis on choosing representative samples. Phonelab supports mobile-computing field studies, including collection of end-user data [9]. The University of Maryland's Build It, Break It, Fix It competition platform allows researchers to observe how participants develop and attack security-relevant systems [10]. The Security Behavior Observatory, recent work by Wash et al. examining the security behavior of undergraduates, and Levecque and Lalonde's work on cybersecurity clinical trials all apply client-side infrastructure to collect field data on security behavior, but do not enable direct, controlled experimentation [7, 8, 13].

Our work is thus distinctively different, offering a scalable tool for conducting online programming experiments with software developers. While we developed Developer Observatory for the purpose of conducting security experiments, its use could extend beyond security into software development and other computer science fields, enabling evaluation of Application Programmer Interfaces (APIs) and/or observation of the learning process of software developers.

3 Requirements

We distill requirements for online secure-programming experiments targeting software developers.

3.1 Validity requirements

Validity requirements are designed to ensure that a study successfully meets its scientific goals for measuring useful information about participant behavior.

Internal validity The framework should prevent duplicate participation to the extent possible. The framework should support randomly assigning participants to conditions (with or without weighting or blocking). The framework should also support showing tasks to participants one at a time, and should allow researchers to specify a randomized or rotated order for tasks. We also want the framework to promote a study environment that is as similar as possible across all participants: e.g. using the same software versions and pre-installed libraries.

Ecological validity. In any laboratory study, it is important to maximize ecological validity, or the ways in which the study resembles real-life situations, to the extent possible. The framework should promote ecological validity by allowing participants to use their own equipment and software setup whenever possible, and to participate from their own home or work environments.

Data collection. The framework should support automated collection of as much data as possible. This starts with, but is not limited to, collecting the code participants write and self-reported survey questions about their demographics and their experience with the study. For example, we would also like to capture when code is copy-and-pasted (and potentially ask the participant where it came from), how many times the participant tests their code, and whether it produces execution errors. We would also like to capture timestamps for all relevant events. The data collection mechanism should be extensible to support varying needs across studies.

3.2 Interaction requirements

We want the framework to be easy to use for both participants and researchers.

Participants. We do not want interaction difficulties to drive away otherwise-willing participants. Participants should be able to participate with the equipment and software they have, without having to install libraries, virtual environments, or programming languages. We want each participant's experience to be seamless, from clicking a link in a recruitment advertisement through completing all programming and survey tasks, without having to take explicit actions like entering an identification (ID) value, creating an account, or stopping to download or install. Participants who give up on a task should (where reasonable within the study design) be able to skip it, provide information about why, and continue to the next task, rather than quitting the study entirely.

Researchers. The framework should make it easy for researchers to develop a new study; this means it should be modular and extensible, and in particular that items that are expected to change from study to study should be isolated from longer-term infrastructure components. It should be easy for researchers to monitor an ongoing study, in order to manage the rate of participation and to quickly identify and correct any problems that occur. The collected study data should be easy to export, and data collected in different parts of the study should be easily linkable via a consistent participant pseudonym. Finally, the infrastructure should be deployable within most researchers' budgets.

3.3 Technical requirements

We define five technical requirements, as follows.

Reliability. Participants, who are recruited by email or web advertisement, may follow the provided link and begin participation at any time. As such, it is critical that the study infrastructure is robust and reliable, without requiring researchers to constantly monitor that it is operating properly while the study is running.

Scalability. The scale of desired participation will vary based on requirements of individual studies, but participant counts in the order of hundreds of participants seem likely to be needed. While it may not be necessary for all participants to run at one time, the infrastructure must expand to allow sufficient simultaneous participants, and data storage and retrieval must be designed for the total participant load.

Participant Isolation. Participants must not be able to see or alter others' code or survey responses. Further, participants should not be able to access information about conditions to which they have not been assigned (to protect the integrity of the experimental design).

Security. More generally, participants should not be able to break or exploit the study infrastructure. Specifically, participants should not be able to access the underlying operating system to perform operations outside the scope of the study, should not be able to disrupt the study infrastructure, and should not be able to, e.g., use the study infrastructure as a platform to send spam or cause other harm. If a participant does disrupt the infrastructure (for example by writing code that creates an infinite loop), this disruption should affect only that participant and not extend to other participants. This requirement is complicated by the fact that participants will by definition be writing and executing code (possibly under a privileged user account).

Participant privacy. Participant privacy is crucial, both as an ethical matter and as a practical reassurance to participants that their cooperation is low-risk. The framework should allow either anonymous or pseudonymous participation. In the pseudonymous case, participants should be identified by a coded ID which is stored separately from the email address (or other contact data) used for invitations. When desired, researchers can map secondary information (such as the channel via which a participant was recruited) to the coded ID.

4 Developer Observatory design

To meet these requirements, we designed a framework (cf. Figure 1) with five main logical components (many of which can be co-located): a landing page, a VM manager, a task server with an online code editor, a database, and a survey tool.

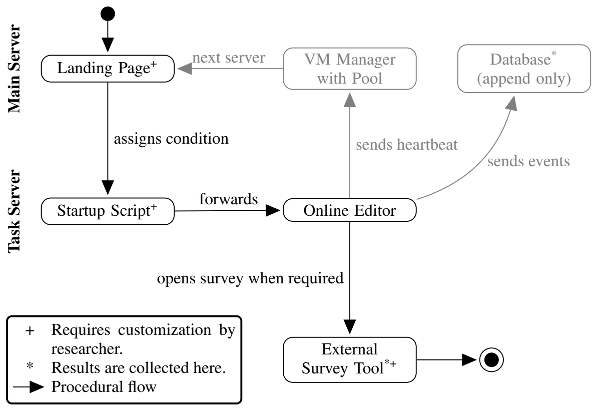


Figure 1: The different components of the Developer Observatory. The black boxes and arrows illustrate the navigation path of a participant (note that the grey components do not).

A link in an invitation email or advertisement directs participant to the study's landing page. Here, information about the study is offered: The researchers and their institutions are identified so that participants can contact them in case of questions, the purpose of the study is explained in as much detail as possible without priming the participants and introducing bias, and a downloadable consent form is presented that details how information obtained in the study is obtained, processed, stored and published in an ethical and privacy-preserving way.

After consenting to the study, participants are assigned a condition and a VM instance. Conditions can be assigned round-robin, or with weighted probabilities (e.g., to account for conditions with higher dropout rates). The task order can be randomized when appropriate.

The VM manager creates instances of the task server and provides the next available instance for the correct condition to the participant. It is also responsible for maintaining a small queue of ready VMs, so that participants do not have to wait for a new instance to boot up, and for shutting down VMs that are no longer considered active according to the study policy.

Participants are then forwarded to the task server, which hosts an online code editor. The editor supports writing and executing source code, as well as showing results and errors to the participant. Task instructions are provided inline, so that participants can read the instructions and work on their code in the same interface. Participants are also able to test their code and view output, as well as recover from infinite loops and kernel crashes. Once participants are ready to move on, they can indicate whether they are completing or skipping the task. We expected that offering the option to skip a task would decrease the number of participants who quit the study in frustration because they could not solve a particular task.

The task server has several important features that address the requirements described in Section 3. These include data collection features, such as detection of copy-paste events, test runs, task completions, and the collection of snapshots of the code at each important event. The task server also enables popover messages and/or short pop-up surveys triggered by particular events; for example, a copy-and-paste event can trigger a reminder to document the source of any pasted code.

Once a participant has completed all programming tasks, they can be forwarded to an exit survey. This exit survey might include questions about the tasks they worked on, their demographics and background, and any other relevant information. Information from the programming tasks and the exit survey is linked via a participant's unique pseudonym ID so that participant's code can be imported to help the participant remember their work while answering.

5 Implementation

In this section, we discuss our implementation of the Developer Observatory. Our implementation relies on Amazon EC2, which provides reliability and scalability. We use one EC2 instance to host the landing page, VM manager, and database. The VM manager creates and assigns participants to EC2 *micro* instances: these are cheap instances with limited resources, but they are sufficient to run the online code editor. The database is stored on a separate PostgreSQL server.

The VM manager keeps a small pool of already-prepared instances for assignment to new participants. A VM instance requires about one minute to boot, so without this pool, participants would be forced to wait unacceptably before beginning the study. As one instance in the pool is assigned to a participant, a new instance is started in order to maintain the pool size. For example, in our first study using Developer Observatory [2], we found we needed a pool of 10 instances to keep up with demand. For a subsequent, smaller study, we used an initial pool size of five, which we reduced to three as participation slowed. This generally proved sufficient to prevent potential participants from waiting. [4]

To manage outstanding instances, we use a heartbeat signal. An assigned instance will send a heartbeat to the VM manager every 60 seconds, starting when that instance is assigned to a participant, as long as the browser window is open and the online editor is operational. An assigned instance automatically shuts itself down after a participant finishes all tasks; the VM manager can also apply a shutdown policy to kill the instance if it has been open too long. In our cryptographic API study, we experimented with a few shutdown policies before finding that killing an instance after four hours with no heartbeat provided a good tradeoff between wasting instances and allowing participants to complete tasks in their own time. We applied this policy for our recruitment study as well. We note that Amazon limited us to a maximum of 50 concurrent instances, so managing shutdown became important during times of high demand.

As our online code editor, we used Jupyter Notebook 1.0, an in-browser editor for writing and executing Python code.[1] Each programming task consists of two Jupyter *cells*: one *markdown* cell that provides instructions for the task, and one *code* cell that provides skeleton code for the task itself or to test it. Participants then add and edit code directly in the code cell and run their code with the run button. A screenshot from the cryptographic API study is shown in Figure 2, left.

When a participant is assigned to an instance, the VM manager provides the assigned condition to the instance as a parameter. The appropriate task file(s) for the assigned condition are then copied into place, with a generic name, so that the filename does not provide extra information about the study goals and conditions.

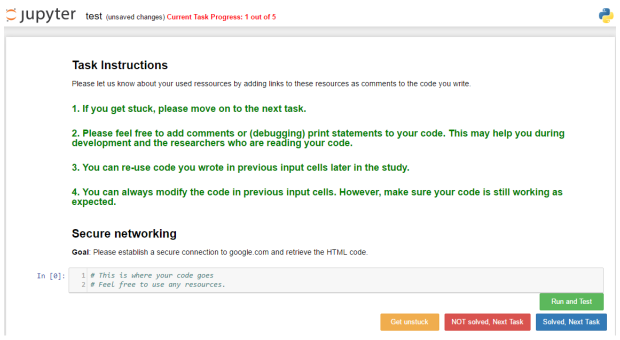
After all tasks have been completed or skipped, the participant is redirected to the exit questionnaire. Thus far, we have used Qualtrics and LimeSurvey to host exit questions. To help the participant remember their work, Developer Observatory can show participants' code for a given task next to questions about that task (cf. Figure 2, right). To achieve this, we used the Qualtrics Web Service module and a JavaScript solution for LimeSurvey.

Data collection. We configured Developer Observatory to capture the current state of the participant's code at each click of the run button. This reflects the fact that many developers add functionality incrementally, test, and add and remove debugging output as they work. By capturing snapshots of this process, we can examine how the code was developed and how many test runs were associated with different tasks. We store the result of each run, including return codes, error codes, exception messages, and stack traces.

We also gather copy-and-paste events to measure the fraction of code our participants write themselves versus copy-and-paste from information sources such as official API documentation, StackOverflow, and blog posts. Additionally, participants are encouraged to report the URL code was copied from. The text associated with copy-and-paste events is available for later analysis in order to distinguish code copied from information resources from code being moved around within the participant's work.

All events that we capture are logged with corresponding timestamps; in addition, we track timestamps when each task begins and when the participant chooses to end the task (by indicating success or by skipping the task).

[1]Jupyter supports kernels for most major programming languages: `github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages`



jupyter test (unsaved changes) Current Task Progress: 1 out of 5

Task Instructions

Please let us know about your used ressources by adding links to these resources as comments to the code you write.

1. If you get stuck, please move on to the next task.

2. Please feel free to add comments or (debugging) print statements to your code. This may help you during development and the researchers who are reading your code.

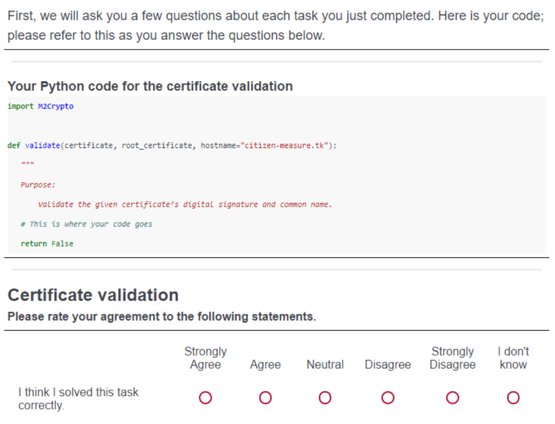
3. You can re-use code you wrote in previous input cells later in the study.

4. You can always modify the code in previous input cells. However, make sure your code is still working as expected.

Secure networking

Goal: Please establish a secure connection to google.com and retrieve the HTML code.

```
In [0]:
# This is where your code goes
# Feel free to use any resources.
```



First, we will ask you a few questions about each task you just completed. Here is your code; please refer to this as you answer the questions below.

Your Python code for the certificate validation

```
import M2Crypto


def validate(certificate, root_certificate, hostname="citizen-measure.tk"):
   """
   Purpose:
       Validate the given certificate’s digital signature and common name.
   # This is where your code goes
   return False
```

Certificate validation

Please rate your agreement to the following statements.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	I don't know
I think I solved this task correctly.	○	○	○	○	○	○

Figure 2: Screenshots from one study conducted using Developer Observatory. Left: Participants completed tasks using a Jupyter Notebook as an online code editor. Task instructions and skeleton code were provided. Right: We used the Qualtrics web service feature to show each participant their code for the task being asked about.

Developer Observatory is designed to be easily extensible for features that may be important to other researchers. For example, any event detected by JavaScript can be recorded and/or used to trigger a code snapshot. For example, a researcher interested in how developers scroll up and down within their code could add a listener for scroll events that triggers an AJAX request.

Security. Allowing unknown users to execute arbitrary code poses a severe security risk: Malicious users could try to overallocate resources, crashing the infrastructure; try to read other participants' data from the database or delete existing study data; use the infrastructure as a platform to send spam; or other problems.

We manage these risks in several ways. We use EC2 to ensure that damage caused by an attacker to the server hosting the online code editor is limited only to that participant's instance, rather than affecting other participants or the infrastructure more broadly. Additionally, we strictly limit the software installed on the task server instance, minimizing the harm that an attacker who breaks out of the Jupyter notebook can achieve. The only common resource for all instances is the interface to the database, which limits the data it accepts.

We implement public-key pinning for certificate validation for the communication between instances and the database server. Using EC2 security groups, we ensure that the task servers can only create outgoing connections to our database server, nowhere else. Unfortunately, the Jupyter notebook on the task servers is only reachable for participants via HTTP because the hostnames are assigned by Amazon's EC2 and we cannot retrieve a valid certificate for Amazon's EC2 subdomains.

Data transfer from the task to the database server is size-limited and sanitized at both ends.

Finally, we use Google's reCAPTCHA service [2] at the landing page to make it harder for an attacker to create many instances and exhaust the EC2 allocation.

Configuration. All configuration options for our tool are maintained in a single configuration file on the landing page server. The configuration file controls the VM pool size and identifies the type of instances that should be created for the pool (in our case, micro instances). The configuration file also provides an upper bound for total number of participants, so the experiment will stop if a maximum limit (such as one specified by an ethics committee) is reached.

The configuration file also manages several security items, including Amazon EC2 keys and security-group ID, a registration key for reCAPTCHA, and database login credentials. It also supports the use of a one-time token system to limit each invited participant to only starting the study once.

Meeting our requirements. In Table 1, we summarize how Developer Observatory achieves the requirements described in Section 3. We will release Developer Observatory publicly by the time of the workshop.

6 Using Developer Observatory

We successfully used Developer Observatory to run controlled experiments for two studies of Python developers. Below, we provide an overview of these studies, an evaluation of the efficacy of our tool for running them, and a discussion of our lessons learned. Both studies were approved by all our institutions' review boards.

Overview of studies. We first used Developer Observatory for an online, between-subjects study comparing how effectively Python developers could write correct, secure code using different cryptographic libraries (the *API study*) [1]. We recruited developers from GitHub and assigned them programming tasks using one of five con-

[2]cf. `https://www.google.com/recaptcha`

Table 1: The requirements we specified, and how they are met by Developer Observatory.

Validity	
Internal	Our framework serves all participants the same VM image, which comes with the prescribed development environment (i.e., pre-installed Python version and libraries) and the same online editor based on Jupyter. We allow for the random assignment of participants to conditions, the prevention of duplicate participation via a token system and reCAPTCHA, and we enable the consecutive display of randomized tasks.
Ecological	Our framework allows participants to solve programming tasks from home, their place of work, or their favorite coffee shop, using their own computer.
Data collection	Each code execution triggered by a participant is stored on the database server, along with the execution result and a time stamp. Additionally, copy-and-paste events are stored. Qualtrics stores questionnaire data.
Interaction	
Participants	Participation in a study using our framework does not require participants to download programming tasks or install any software such as an editor or libraries. Instead, they can participate using their web browser. Participants are directed through study tasks without needing to follow complicated directions.
Researchers	Setting up a new study is straightforward. Once the main server (hosting the landing page, VM manager, and database) has been installed and configured, very little must change to run a new study. Only the consent form, tasks and associated condition assignment plan, and exit questionnaire must be specified, and some configuration options (such as VM pool size) can be updated. Data can be retrieved from the database and Qualtrics in CSV or SQL format; the two sets of data are easily linked via participant ID.
Technical	
Reliability	We use Amazon's EC2 infrastructure to ensure a high level of reliability for virtual machines.
Scalability	Amazon's EC2 infrastructure can scale up well beyond our requirements.
Participant Isolation	Participants are each assigned their own virtual machine, guaranteeing strong isolation and preventing interference between participants.
Security	VMs that serve tasks are firewalled and only allow incoming network traffic via HTTP. Outgoing traffic is limited to the HTTPS port of the database server. Data storage requires strong authentication using a token system, and the database account is limited to only inserting new results.
Privacy	Participants are assigned a random ID after agreeing to the consent form. The mapping between participant email addresses and their random ID is stored on a different server that is not accessible from the tool.

ditions and one of two sets of encryption tasks. Assignment to conditions and task sets was initially random, with counterbalancing to ensure roughly equal numbers of participants started in each condition. However, when it became clear that dropout rates (starting but not completing the study) varied widely among conditions, we used Developer Observatory to weight the assignment to favor underpopulated conditions. Within each condition, task order varied using Latin-Square assignment. After finishing the programming tasks, participants completed a brief exit questionnaire with questions about the code they had just written, the assigned library, their prior programming and security experience, and demographics.

Later, we used Developer Observatory to compare correctness and security results for professional and student developers (the *recruitment study*), again recruiting participants from GitHub [4]. Here, no conditions were assigned (other than self-reported student/professional status post-hoc). Participants completed three randomly ordered security-relevant tasks and a brief exit questionnaire similar to the questionnaire from the API study.

Overall, 2396 participants accepted the consent form and started one of the studies (API: 1571, recruitment: 825). Of these, 563 (API: 256, recruitment: 307) completed all tasks. We note that the dropout rate was significantly higher for the crypto API study (84%) than the recruitment study (43.6%) ($X^2 = 130.5$, $p < 0.0001$). We believe this difference is primarily attributable to the tasks in the API study being harder: in the API study, participants achieved functional solutions for only 64% of all tasks, compared to 88% in the recruitment study.

Participants who did not drop out spent median 56 minutes (Q1 = 35, Q3 = 106) on the programming tasks in the API study and 53 minutes (Q1 = 30, Q3 = 85) on the programming tasks in the recruitment study.

Participant experience. In both studies, participants provided feedback in free-text questions within the exit questionnaire, by leaving their opinions commented into their task code, and by emailing us. Overall, most feedback was positive. A few participants wrote us when they were denied access to the study during an overload incident (described below) or an AWS outage. Nine participants complained about using the online editor; most reported falling back to a local editor, then pasting their code into the Notebook. Seven API-study participants reported annoyance with a popover message that appeared after each copy-and-paste event reminding the participant to document the source of any reused code; we reworked the popover to not show after copy-and-pasting within the editor field and received no further complaints. A few participants mentioned that Python libraries we included in the Notebook were insufficient or out of date (easily fixable for the next study), and two had trouble with LimeSurvey. A few potential participants were disappointed the platform didn't work in a mobile browser.

Researcher experience. From our perspective as researchers, the studies ran (mostly) smoothly. We easily retrieved and analyzed the code our participants wrote, their copy-paste events, and data about how frequently they pushed the "run" button. We were also able to quickly link participants' code to their exit questionnaire responses and analyze all facets of their results: whether their code was functional, whether it met our security criteria, and how their self-reported responses correlated with their coding results.

Compared to running a developer study in the lab [2], using Developer Observatory we found recruiting and running study participants to be easier and faster. Because we couldn't ask participants to think aloud, we missed out on some qualitative data. However, participants left many detailed and passionate responses both in the comments to the code they wrote and in free-text portions of the exit questionnaire, so we did collect interesting qualitative nuggets. Our measurements of participants' code were as good as in a lab study, but timing data was less precise, because participants stepped away from and then resumed tasks. However, we found that online participants were willing to commit as much or more total time as lab participants, despite not being compensated, perhaps because they could work on their own schedules from the comfort of their preferred environment. Overall, we successfully collected and analyzed detailed data about participants' behaviors.

Our biggest challenges using Developer Observatory were managing the pool of VM instances and calibrating how many recruitment emails to send to avoid overloading the system (see below). After the first week, however, we became proficient, and managing these resources required little additional effort.

Managing participants and resources. Initially, instead of the heartbeat feature, we expired an instance four hours after it was assigned to a participant. Many potential participants opened the online code editor before deciding not to participate; these instances remained idle for four hours. Given our initial (default for EC2) limit of 20 instances running simultaneously, our available instances were quickly exhausted, and some potential participants were rejected due to lack of resources. We solved this problem by implementing heartbeats, by requesting an increase from Amazon to 50 concurrent instances, and by manually emailing participants who were rejected to re-invite them. The original four-hour timeout also inconvenienced a small number of participants who were cut off while still working on their code. We manually restored these participants' sessions and emailed them a link to continue. Adding the heartbeat feature solved this issue.

We also made interesting observations about timing. For the most part, we periodically sent invitation emails and potential participants arrived at the landing page in a fairly even stream. However, we once sent emails periodically over a weekend. We saw almost no traffic during the weekend, but then received a huge influx of participants on Monday morning, presumably because invited participants had arrived back at work and were all checking their email at a similar time. This rush nearly overloaded our infrastructure: A few participants were unable to start the study. We recognized the issue, freed up unused instances, re-invited the participants who had experienced issues and paused invites until the rush had evened out. For future studies, we recommend not sending large numbers of invitations immediately before or during a weekend or holiday.

Future work. In future work, we plan to expand and improve Developer Observatory. For researchers, we plan to add a web-based configuration and management tool that makes it easy to set up study parameters, and then monitor participants in progress, from one graphical interface. This interface should also allow researchers to retrieve data from both programming tasks and the exit questionnaire in one step. We would also like to add further instrumentation for collecting participants' behavioral patterns while writing code, and perhaps to allow a study design that switches back and forth between coding tasks and survey questions as many times as necessary. We would also like to add an explicit feature allowing participants to pause and resume sessions at their convenience, without worrying about closing their browser or having their instance killed by the VM manager as inactive. Furthermore, we plan to add an automatic invitation system, which invites new participants depending on the current workload as well as sending re-invites to participants who were rejected because no resources were available. We also plan to support other researchers who wish to extend Developer Observatory, and will open-source Developer Observatory as well as provide our contact information for support.[3] While we did not use debugging during the study, it is possible to extend Jupyter with de-

[3] `https://developer-observatory.com`

bugging functionality using ipdb[4] to give developers an additional tool that they might normally use.

Takeaways. Security experiments in which developers are asked to write code are increasingly common. We developed Developer Observatory, a distributed, online platform for administering security programming studies to developers remotely. We found that Developer Observatory allowed us to recruit professional and hobbyist developers, from all over the world, faster and more easily than for a lab study. This allowed us to increase the power and generalizability of our experiments, with little to no loss of internal validity.

We designed Developer Observatory both to meet our immediate needs and to be extensible to other security experiments. Our experience suggests that interaction with Developer Observatory was satisfactory both for us as researchers and for most of our participants. We hope that by providing Developer Observatory as an open-source tool for the security community, we can help increase researchers' access to developers, lessen the inconvenience of study participation for developers, and increase the scale and validity of future research.

7 Acknowledgements and disclaimer

The authors wish to thank Mary Theofanos, the anonymous reviewers, and all our study participants. This work was supported in part by the German Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA), and by the U.S. Department of Commerce, National Institute for Standards and Technology, under Cooperative Agreement 70NANB15H330.

Any mention of commercial products is for information only; such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that these entities, materials, or equipment are necessarily the best for the purpose. This paper is presented with the hope that its content may be of interest to the general scientific community. The views in this paper are those of the authors, and do not necessarily represent those of the Census Bureau, the Department of Commerce, or the U.S. Government.

[4]`https://pypi.python.org/pypi/ipdb`

References

[1] Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M. L., and Stransky, C. Comparing the Usability of Cryptographic APIs. In *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)* (2017), IEEE.

[2] Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. You Get Where You're Looking For: The Impact of Information Sources on Code Security. In *Proc. 37th IEEE Symposium on Security and Privacy (SP'16)* (2016), IEEE.

[3] Acar, Y., Fahl, S., and Mazurek, M. L. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In *Proc. IEEE Secure Development Conference (SecDev'16)* (2016), IEEE.

[4] Acar, Y., Stransky, C., Wermke, D., Mazurek, M. L., and Fahl, S. Security Developer Studies with GitHub Users: Exploring a Convenience Sample. In *Proc. 13th Symposium on Usable Privacy and Security (SOUPS'17)* (2017), USENIX Association.

[5] Benzel, T. The science of cyber security experimentation: the DETER project. In *Proc. 27th Annual Computer Security Applications Conference (ACSAC'11)* (2011), ACM.

[6] Di Penta, M., Stirewal, R., and Kraemer, E. Designing your Next Empirical Study on Program Comprehension. In *Proc. 15th IEEE International Conference on Program Comprehension (ICPC'07)* (2007), IEEE.

[7] Forget, A., Komanduri, S., Acquisti, A., Christin, N., Cranor, L. F., and Telang, R. Security behavior observatory: Infrastructure for long-term monitoring of client machines. Tech. rep., Carnegie Mellon University, CyLab, 2014.

[8] Lvesque, F. L., and Fernandez, J. M. Computer security clinical trials: Lessons learned from a 4-month pilot study. In *Proc. 7th USENIX Workshop on Cyber Security Experimentation and Test (CSET'14)* (2014), USENIX Association.

[9] Nandugudi, A., Maiti, A., Ki, T., Bulut, F., Demirbas, M., Kosar, T., Qiao, C., Ko, S. Y., and Challen, G. PhoneLab: A Large Programmable Smartphone Testbed. In *Proc. 1st International Workshop on Sensing and Big Data Mining (SENSEMINE'13)* (2013), ACM.

[10] Ruef, A., Hicks, M., Parker, J., Levin, D., Mazurek, M. L., and Mardziel, P. Build It, Break It, Fix It: Contesting Secure Development. In *Proc. 23nd ACM Conference on Computer and Communication Security (CCS'16)* (2016), ACM.

[11] Siaterlis, C., and Masera, M. A Review of Available Software for the Creation of Testbeds for Internet Security Research. In *Proc. 1st International Conference on Advances in System Simulation (SIMUL'09)* (2009), IEEE.

[12] Siegmund, J., Siegmund, N., and Apel, S. Views on Internal and External Validity in Empirical Software Engineering. In *Proc. 37th IEEE International Conference on Software Engineering (ICSE'15)* (2015), IEEE.

[13] Wash, R., Rader, E., and Fennell, C. Can People Self-Report Security Accurately?: Agreement Between Self-Report and Behavioral Measures. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI'17)* (2017), ACM.