A Case Study of Software Security Red Teams at Microsoft

Justin Smith*
Lafayette College
smithjus@lafayette.edu

Christopher Theisen
Microsoft
christopher.theisen@microsoft.com

Titus Barik
Microsoft
titus.barik@microsoft.com

***Abstract*—The modern software security adversary employs persistent and evasive attack techniques, for example—using zero-day exploits that have not been disclosed publicly—to target high-profile companies for political and economic espionage or to exfiltrate sensitive data or intellectual property. To combat these threats, large organizations are adopting an emerging practice of staffing full-time offensive security teams, or *red teams*. To understand the workflows, culture, and day-to-day practices of software security engineers in red teams, we conducted 17 interviews with informants across five red teams within Microsoft. We found that software security engineers have substantial impact in the organization as they harden security practices, drawing from their diverse backgrounds. Software security engineers are both agile yet specialized in their activities, and closely emulate malicious adversaries—subject to some reasonable constraints. Although software security engineers are in some respects software engineers, they also have several consequential differences in how they write, maintain, and distribute software. The results of this work are applicable to practitioners, researchers, and toolsmiths who wish to understand how offensive security teams operate, situate, and collaborate with partner teams in their organization.**

I. Introduction

> As humans, we've let security fall by the wayside. Now it's all catching up. Our organization hires people like me to try and find these issues, with the mindset of an attacker—somebody who is malicious—before somebody in the wild does.
>
> *Cliff*, Cloud Computing (Red Team)

Sophisticated adversaries attack high-profile companies for purposes of political and economic espionage or to exfiltrate sensitive data or intellectual property [1, 2, 3]. To defend against these adversaries, companies have traditionally adopted various practices, such as deploying software security tools [4], using firewalls to protect the perimeter, or adopting bug bounty programs. However, these defensive models of organizational security are inadequate against modern threats. Software security tools detect threats that are already-known—but sophisticated adversaries find and exploit 0-day vulnerabilities that have not been disclosed publicly [5, 6]. Shared, cloud-based infrastructure introduces substantial challenges to perimeter-based security models, blurring the boundaries between insider and outsider threats [7]. Bug bounty programs crowdsource software security and compensate external security researchers for reporting exploits; although these programs are beneficial, exploit discovery is front-loaded to newly released software and services, and primarily focuses on probing popular targets [8, 9]. In contrast, sophisticated adversaries—such as nation-states—employ long-term campaigns with specific objectives over repeated attempts [1].

An emerging and unconventional approach adopted within large software companies, like Amazon, Dropbox, Facebook, and Microsoft, is to embed full-time offensive security engineering teams—or *red teams*—that realistically emulate sophisticated adversaries to compromise, infiltrate, and control the organizations' production software and systems. To investigate the workflows, culture, and practices of software security engineers within red teams in modern software organizations, we applied a human-centered, "software studies" lens [10] by eliciting *first-hand accounts from professional software security engineers* at Microsoft, through semi-structured interviews with 17 informants. We used thematic analysis to identify patterns and themes important to these red teamers.

Our case study found that software security engineers have substantial impact in the organization as they harden security practices, drawing from their diverse backgrounds. Software security engineers are both agile yet specialized in their activities, and closely emulate malicious adversaries—subject to some reasonable constraints. Our informants felt that security culture was a crucial aspect to effective red teaming, and emphasized how their role has at times been inadequately supported by being conflated with software engineers. Although software security engineers are in some respects software engineers, they also have several significant differences, such as the *security mindset*. Our findings are applicable to practitioners, researchers, and toolsmiths who wish to understand how offensive security teams operate, situate, and collaborate with partner teams in their organization.

II. Methodology

Research context. We conducted the study at Microsoft in Redmond, Washington. Microsoft is a multinational technology company, with a recent focus on cloud computing.

Recruitment. Prior research has found that security practitioners are hesitant to divulge practices to perceived outsiders [11]. To recruit informants for our study, we used a combination of random sampling and snowball sampling. We initially randomly sampled informants from our company

*Author conducted this work while a visiting researcher at Microsoft.

978-1-7281-6901-9/20/$31.00

address book by searching for full-time engineers with the title of "Software Security Engineer," having at least three years of experience within the company. Next, we used snowball sampling [12] because this non-probabilistic approach is particularly effective at reaching hard-to-reach groups [13, 14]. At the end of each interview, we asked the current informant to connect us with individuals who engaged in very different activities from themselves. Informants were compensated for their time with meal vouchers.

To reduce interviewer bias and obtain data from different perspectives during the interviews, each interview was conducted by two researchers. It is important to note that not all topics were discussed at the same level of detail with all informants, due to the nature of semi-structured interviews.

Informants. We continued interviewing informants until obtaining *theoretical saturation*, that is, the point at which we were obtaining interchangeable stories. This occurred at 17 informants, across five primary software security teams with offensive security engagements (Table I). Informants had a mean experience of $\mu = 5$ years at Microsoft, and a mean experience of $\mu = 8$ years total for security-related engagements. To maintain anonymity, we refer to these informants using pseudonyms. Fifteen of our informants identified as male and two (Clara and Pam) identified as female. A typical security team consists of 3–10 full-time software security engineers and engineering leads.

Interview protocol. We conducted semi-structured interviews[1] that focused on the informants' recent experiences, responsibilities, workflow, collaboration, challenges, motivations, tools, processes, and learning. Using guidelines from Hove and Anda [15], we conducted interviews in pairs, with one interviewer taking the lead while the other takes notes and asks additional questions. We interviewed informants in their own offices, where they often showed us relevant artifacts, such as reports that they had prepared for development teams, custom tools they had built, and their security environment as configured for day-to-day work. We recorded and transcribed all interviews; interviews typically lasted just under an hour. We obtained informed consent using Microsoft's IRB protocol.

Analysis. We used the thematic analysis procedure described by Braun and Clarke [16], which in summary consists of six phases: 1) familiarizing yourself with your data, 2) generating initial codes, 3) searching for themes, 4) reviewing themes, 5) defining and naming and themes, and 6) producing the report. After transcribing the interviews, we used the ATLAS.ti data analysis software for qualitatively coding the data. We performed coding over multiple iterations to search for, review, define, and name themes. The results of this analysis are found in Section III.

Validity. We used several methods to mitigate against common validity threats in qualitative research [17], namely: contextual inquiry, disconfirming evidence, prolonged engagements, and thick description. Section IV describes details of these methods alongside the corresponding threats to validity.

[1]Interview script available at https://aka.ms/redteams-casestudy.

TABLE I
INFORMANTS

Informant[1]	Security Role	Experience (Yrs)[2] Organization	Total
Cloud Computing Group ($n = 3$)			
P1: Chase†	Cloud Applications	6	10
P2: Cliff	Cloud Services	3	7
P3: Clara†	Cloud Services	2	12
Databases Group ($n = 2$)			
P4: Bill†	Applications	5	10
P5: Brian	Networks	3	6
Devices and Gaming Group ($n = 6$)			
P6: Dan†	Browser	4	18
P7: David	Browser	8	8
P8: Dean†	Cloud Services	3	12
P9: Derek†	Operating Systems	6	7
P10: Gary†	Threat Protection	5	5
P11: George†	Threat Protection	5	5
Enterprise Group ($n = 2$)			
P12: Eric†	Enterprise Ecosystem	4	8
P13: Evan†	Enterprise Ecosystem	3	5
Productivity Software Group ($n = 4$)			
P14: Pam	Build Systems	5	5
P15: Pat†	Client Applications	8	8
P16: Paul	Client Applications	4	4
P17: Peter†	Compliance and Auditing	14	14

[1] All informants have the title Software Security Engineer. Informants marked with † are Senior-level or higher within the organization.
[2] Years of software security experience within the current organization and total experience across all professional security engagements.

III. FINDINGS

We identified three top-level themes relating to workflow, security culture, and differences between software engineering and software security engineering. Because the interviews were semi-structured, we covered themes depending on the direction of the conversation. Table II summarizes those themes.

Through our analysis, we found that software security engineers conduct long-term campaigns to exploit specific goals and targets, that team members have unique specializations that they apply to their tactics, and that their work is context-sensitive and requires on-demand tools. Red teams collaborate with partner teams and communicate vulnerabilities to them (Section III-A). Informants were diverse in their backgrounds and told us about their motivations for becoming a software security engineer. We found that red teams closely emulate the activities of malicious adversaries, but differ in their ethical responsibility to prevent harm to the organization (Section III-B). Informants also reported that while software engineers and software *security* engineers often use the same tools, there are substantial differences in how they make impact in their organization (Section III-C).

TABLE II
SUMMARY OF INFORMANT THEMES

THEME	SUMMARY	REPRESENTATIVE EXAMPLES	PARTICIPANTS
Workflow (Section III-A)	Red teams undertake campaigns with goals and targets defined a priori. Software security engineers must both develop specializations and maintain the ability to pivot quickly to new contexts. Red teams aren't just responsible for reporting the vulnerabilities they find during campaigns; they must also motivate and enable developers to fix them.	"Every product is unique." "We serve as sparring partners for blue teams." "The vulnerabilities we find skew toward the expertise of people on our team." "[We] follow up with the [devs] to make sure they actually fix [vulnerabilities]."	P1, P2, P3, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17
Security Culture (Section III-B)	Software security engineers come from diverse backgrounds, with varying motivations. Red teams are most effective when they realistically emulate a malicious adversary. Red teams take seriously the ethical responsibilities of white hat hacking, and apply their skills to improve security.	"It's a diverse group. We don't look like your typical hacker." "Rules of Engagement document[s] state what we can and can't do." "Once we've proven the point that it could be done, we don't actually have to push the detonator button."	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P16
Software vs. Security (Section III-C)	Although software security engineers conduct software development activities, they have several consequential differences in how they write, maintain, and distribute software. Software security engineers have unique specializations such that they should not be evaluated directly as software engineers.	"Both are development in one way or another." "It's the difference between seeing something as a tool versus a weapon." "The main difference is having the right mindset for security, having an adversary's mindset."	P1, P2, P3, P4, P5, P6, P7, P9, P12, P13, P14, P15, P16

A. Workflow

We describe the structure of red team security campaigns with partner teams (Section III-A1), the tactics used to execute these campaigns (Section III-A2), and how red teams communicate their results to developers in partner teams (Section III-A3).

1) Campaigns: **A typical campaign.** Informants' workflows comprise a series of *campaigns*. Here we characterize a typical campaign to provide a framework for understanding software security engineers' activities. Dean describes a recent red team campaign that more-or-less mirrors an offensive kill chain—kill chains describe the different phases of an intrusion [18]. The campaign begins with "customer engagement and scoping, and ends with reporting. End-to-end, it's normally six to eight weeks. Everything else in between is testing, and it's usually based on the environment."

As Dean explains, campaigns begin with *scoping*, which includes "buy-in," "meeting with the customer," "deciding the target," obtaining "permission to own their stuff and to go attack their things," and following legal protocol, or "getting legal cover so no one comes and yells at me." Scoping takes about a week. The next stage is *initial access*, or "cracking the perimeter"; it is "the longest portion of the assessment because we have to find a flaw to get inside the perimeter." This activity usually involves "a little bit of reverse engineering, using virtual machines, and using the source code to figure out what is what." Dean adds, "there's something called 'assume breach' where for this campaign we assume one adversary already has credentials within the network. And the reason why that exists is because it's true." The next two weeks are *lateral movement*, essentially, "moving around the network to get to wherever we need to go." The final step is *command and control* where the red teamer installs a remote access tool in order to establish a persistent, interactive channel for exfiltration of data or other intellectual property. At this point, Dean notes, "I am you as far as everything else in the network is concerned."

Every campaign is different. Other campaigns are more open-ended. As Clara (and Cliff) observe, software security red teams are an emerging role, and "all the teams seem to have different styles of operations: sometimes they go fast and they aren't expected to maintain persistence, and sometimes they do the full shebang." Cliff adds, "in some of these cases, it's months or years. They're 'low and slow,' absolutely, but I'd be scared of what that person was able to do."

A campaign's scope is also "very variable and could be anything" (Eric). For example, sometimes the campaign is exploit hunting, "which essentially ends when we have enough bugs to make a chain of exploits that are required to do whatever we want" (Eric). This type of campaign ends with a mitigation proposal to the software engineers. Other campaigns, as Paul describes, "try and figure out where to go by things that either have had very little or a lot of public interest lately. In the middle I don't care so much. When there is a lot of public interest you feel like it's likely that someone external is going to discover something pretty critical." But regardless of the type of campaign, they always have a specific objective, "no matter how you get that to happen" (Chase).

2) Tactics: "It's challenging to put structure onto this high-flying, edge-of-your-seat, break-stuff-however-you-can, everything-is-different sort of thing. Boy, it's tough" (Cliff).

Software security engineers are agile in their abilities, but also specialized, chaining together a sequence of conventional and domain-specific tactics to execute red team activities. Much of the work is exploratory, requiring creativity and insight, and sometimes a bit of luck. As Paul says, "there are definitely some anxiety-inducing aspects. I think that's just the nature of

anything exploratory. You have no idea if there is anything to find." Evan comments, "that's just the job. You are looking for something that might not exist." Cliff adds, "you have to be willing to pivot really fast."

Specializations within red teams. Red teams assemble individual software security engineers with specialized skills, allowing team members to offer "different perspectives" (Dan) and make the team more effective overall. They specialize by software domain, such as "desktop applications, browsers, networks, kernels, or virtualization. There are a lot of options" (Derek). Security engineers primarily focus on "where they're more interested in, such as network security, infrastructure, or deployment. We divide our responsibilities" (Brian). Pat notes, "there's also quite a lot of specialized knowledge around different types of vulnerabilities: the sorts of mistakes developers make, or potential problems in processes, for instance."

But specialization has its disadvantages, particularly if a team's composition becomes unbalanced: "the vulnerabilities we find are skewed towards the expertise of the people on our team, based on our past experience, and not necessarily how our customers are getting owned in the wild" (Pam). If security engineers overspecialize, it means they are less likely to have a "broader view or understanding of all of the modules and what the modules are responsible for" (Paul).

Context-sensitive work and on-demand tools. The context-sensitive nature of their work often necessitates that software security engineers create on-demand tools, "because every product is unique" (Chase). As a result, *a lot of our tooling is developed on our own* (Dean, Pam, Pat, Brian, Bill). Using off-the-shelf security tools to identify vulnerabilities is often ineffective, because "it's really low-hanging fruit and rarely yields much. Developers are already using those to filter out most of those bad behaviors" (Dan, Evan). Dan adds that they do use fuzzing tools, which are developed internally: "We write our own. They are constantly changing." Though some red teams report using playbooks (Dean)—collections of reusable procedures—others like Bill argue that the "situations are much too custom or unique to provide blanket tactics." Instead, Cliff argues, "you build stuff for single purposes, especially when you're writing offensive tools. It's more about making it work."

3) Collaboration: **Team rotations.** Red teams rotate amongst different engagements, depending on organizational priorities (Dean, Evan, Peter, Eric), and collaborate with project managers and software engineers in the partner teams.

Communicating vulnerabilities. Finding vulnerabilities, as Pam says, isn't just "throwing vulnerabilities over the fence: it's following up with the team to make sure that they actually fix them." To support developers, Cliff says, "there's a science to explaining things to software engineers and project managers." Bug reports filed against developers "start with an introduction about the problem, explain step-by-step what is going on and why it's bug" (David); "it's not just explaining, but giving some suggestions on how to fix the issue and that's an important part of what you're communicating back to them" (Clara). Dean explains, "we only file bugs if we can prove an exploit. Even if it's super small."

Not all vulnerabilities require detailed reports. Developers are generally responsive to fixing issues when the organization has a strong security culture: "it's usually in people's best interest to fix it" (Cliff). For example, says Pat, "some vulnerabilities, like XSS, are more well-known than others. But other vulnerabilities are more complex and less well-known. Eventually you have to work through questions people have and just get them on the same page as you, building up a foundation of tribal knowledge."

Developer push back. Occasionally, developers push back on the red team's recommendations because they don't recognize the security implications, or because they must "balance their time between shipping features and fixing bugs" (Chase). There are sometimes challenges in "getting developers to resolve or even triage their bugs" (Peter). In these situations, Paul explains that he will occasionally develop proof-of-concepts or more elaborate "weaponized exploits" when "the developer doesn't believe that this is a serious thing. You can prove it is by actually weaponizing it." As Chase says, "but at the end of the day it's not about being unhackable, it's about how hard it is to get hacked."

Engaging with blue teams. Red teams "serve as sparring partners for blue teams" (Dean). The blue teams are "the incident response teams, who run attack investigations in the event of a potential threat, evict the hacker, and basically drive the service back to normal operations" (Chase). Many of the blue team activities rely on "machine learning, analytics, and combining telemetry obtained from host-based, network, service, and application monitoring" (Gary). When red teams discover an exploit, "they hand it over to the blue team for them to signature" (George). These signatures allow the blue teams to identify if the same threat appears from an outside adversary in the future.

B. Security Culture

We describe the security culture of red teams, which includes how security engineers become interested in software security (Section III-B1) and the set of shared attitudes, values, and goals that characterize this role (Section III-B2).

1) Backstory: **Breaking stereotypes.** Do all hackers wear hoodies? "There are definitely stereotypes. Everybody thinks it's just like this high school hacker in his black hoodie in his mom's basement. And you look around and it's a diverse group. We don't look like your typical hacker. I don't look like your typical hacker. I think people think of hackers always as a bad person. It's not necessarily bad. They're not doing things maliciously to hurt somebody else, they're doing it to make it better" (Clara). Cliff reflects on malicious hacking: "I wouldn't say it's worth it. The black mark is not worth it. That's why I'm white hat in the first place."

Motivations. Software security engineers arrived at their positions via a variety of interests or hobbies, such as "pirating games" and "hacking on video games" (Bill, Derek), "writing emulators" (Eric), and "hacking on game consoles in my

free time, which led to studying more about crypto, memory corruption issues, how to exploit them, and how they work" (David). Brian adds, "games are where you start. For example, with games, you try to understand how the thing works and then you start to do some small hacking. From there, you start learning about security. I started doing some practical stuff before college."

Others were introduced to security more formally. For example, Paul "started doing capture-the-flags at University. I've always been interested in security before I started formally studying computer science. I took a couple of courses in security. They were very practical—exploitation based. Very offensive security-based. I met up with some like-minded people there." Cliff also attributes his success to formal education, adding, "that's funny, without school I probably wouldn't have been in security: we wrote worms, we did cross-site scripting and buffer overflows—stuff that is dated, but this is how it all works. I had never thought about making a program do something that I totally didn't intend for it to do as a developer. I thought, this is really cool."

From motivations to careers. Our informants' motivations and interests from casual hacking ultimately developed into careers in security. Before joining the organization as a red teamer, some previously worked in national security, military, or the defense industry (Dean, Cliff, Clara). Others, including Brian and Bill, worked previously in security consulting firms as external penetration testers. Two informants came from a prior software engineering role, such as software testing (Pam, Chase), where "through testing, they found security bugs in products" (Chase). Practices, such as creating "playbooks" (Dean) and using "reverse engineering tools" (David), from these prior experiences were brought into their current red team activities.

Clara observes that, despite these divergent backstories, "the security industry is a pretty tight-knit group," sharing their knowledge and experiences through "conferences," "meetups," and "slack-chat." "It's a small world. We chose security because we truly care about it and we want to do something good," Clara says: "I feel security is the way I can leave an imprint and leave the world a better place."

2) Realism and Ethics: **Emulating the adversary.** "We usually set goals at the beginning of our red team campaigns because we want them to be more like real life," says Chase. Software security engineers are obligated to accurately model malicious actors, yet stop short of actually causing harm, such as data loss. For example, "if the goal is just to own a host, that's what we'll do. If the goal is to emulate a persistent threat, we'll install a remote access tool to maintain access" (Chase). When using a black box approach that models an outside attacker, "you can say with some kind of certainty that the attacker would be more likely to find what we found, because we followed the same kind of steps. We can say [to our engineers] that you should fix these versus other kinds of bugs because they might be found sooner" (Bill).

Practical shortcuts. Nevertheless, there are also practical constraints that malicious adversaries don't have. In particular, "it's usually true that an attacker has no deadline. They don't have the same boundaries that we have" (Brian). "Because of time constraints, our activities are loud, and they're more detectable, which means you're going to get caught more quickly than a sophisticated attacker might. When you're going against real-life attackers you need every advantage you get that's reasonable" (Cliff).

Consequently, red teams employ principled shortcuts, or "gray boxing" (Dean). Cliff explains, "the biggest advantage I get is inside network access. As an employee of Microsoft, I get to see whatever is available on the corporate network and I have insider knowledge, both shared by other red teams and from presentations, meetings, and talking to the people who actually wrote the code. That way every project doesn't turn into a 6-month endeavor."

Cliff continues, "of course, the more shortcuts you take, the weaker your argument will be. But if you think the corporate network is a boundary, we are already at a problem." This might seem like an unrealistic assumption, but as Pam notes, "there have been some high-profile incidents at other companies where attackers have gotten into the network somehow. We don't want to be the next one."

Ethics. Software security engineers both acknowledge and take seriously the ethical responsibility of white hat hacking, voluntarily choosing to give up certain expectations of privacy. Red teams have a "Rules of Engagement document that states what we can and can't do. We've had that spelled out for us so we don't get into trouble" (Clara). Dean adds that these rules "require them to undergo additional background checks. There are additional security controls in place and they tell you this before coming on board. If you are going to do this kind of work, you need to be okay with no privacy." Thus, the latitude granted to software security engineers also leads to more scrutiny: for instance, the activities of software security engineers are subject to heightened logging and monitoring.

In contrast to malicious actors, red teams emulating "root and loot" (Chase) scenarios are "not going to touch customer data" (Cliff). Instead, "they'll explore what the solutions might be to protect that data" (Cliff). Clara notes, "sometimes we'll work the service owner and we'll show them how we can get into an exploit situation. Once we've proven the point that it could be done, we don't actually have to push the detonator button and blow up something."

C. Software Engineering vs. Security Software Engineering

We describe the similarities and differences between software engineers and software security engineers in red teams. We compare how the impact of their work is measured (Section III-C1) and discuss a defining characteristic of red teams, the security mindset (Section III-C2).

1) Organizational Impact: **Similarities.** On the surface, software engineering and software security engineering, as Cliff says, are "both development in one way or another." Brian explains, "you need to have a background as a software engineer to understand what is involved in doing security processes and assessments." Paul adds, "even as a software

security engineer, I do some software engineering, especially around improving [security features in client applications]" and Pam explains that both roles "still need to know how to dive deep into things and poke around."

Differences. But when looking more closely at the two roles, there are substantial differences between software security engineers and engineers. Derek explains, "in security software development, you might write tools that have an offensive purpose but it's usually a quick script and you hack it together. You really don't have the same practices, formality, foundation, and stability to build on [as in software engineering]. As a software engineer, you have to have good coding standards, and stick to specific timelines. If something is delayed, the product won't ship, and that attracts attention."

In contrast, tools developed by software security red teams "aren't shared broadly, because mostly these tools start as a collection of scripts. You don't really care if you run the software and it crashes on you. The tool works well for our team, but not for everyone else. Since the kinds of tools that we write can be dangerous, we try to limit the audience of who can access those tools" (Brian).

Making an impact. As a result of these differences, it can be tricky to measure impact if software security engineers are evaluated directly against software engineers. For instance, for software engineers "you could measure how many check-ins developers do or how many features they developed. But if you are a software security engineer reviewing a very secure piece of software, you might not produce anything. Sometimes when we don't find any vulnerabilities, we explain what we tried to do—how we tested this feature in this particular way or tried to reach this network. Across the industry, it's difficult to measure our output if you don't know the person who is doing the job" (Brian). Paul explains two other approaches to quantifying impact. First, one way is to "find critical vulnerabilities that would have been serious if discovered externally." The second approach is to "take a deep dive into a feature—you may not necessarily find any critical bugs, but you instead make a bunch of recommendations to actively improve the security."

Although software security engineers "probably aren't doing a lot of development—especially on features or products that you'd give to other people—I'd say the security moniker gives you room to make impact in other ways" (Eric). Bill concludes that "it's incorrect for me to call myself a software engineer when I haven't specialized in software engineering. These need to be distinct roles."

2) Security Mindset: **Defining the mindset.** Security is a mindset and it's a way of thinking about the world that is necessary for effective security engineering [19]. But, as Pat explains, "I wouldn't say there's any one process. There's several different ones, and it depends on what I'm trying to achieve." For instance, Pat sometimes thinks about the security mindset as "recognizing how something can be used maliciously or broken" and that "it tends to be a blind spot among many people." Pat explains, "it's the difference between seeing something as a tool versus a weapon I suppose. There is a mantra among some people in the security space that everything is broken."

Recognizing risk. Another aspect of the security mindset "is recognizing when someone is using something that they don't understand. The average developer has no understanding of how dangerous [API] is. A lot of things out there are very dangerous to do, but are not advertised as such. People don't have any awareness of the risk they're undertaking by using [API]. Keeping aware of that sort of risk is part of being a software security engineer" (Pat). Chase clarifies, "what that means is that there are always ways to break into or exploit software. So to be successful you need to have an exploitation mindset, having a broad understanding of the world and looking at possible vectors." As David points out, "when you are a software security engineer, the question you always have in mind is, what if? I'm not saying developers write bad code, because when you develop something your goal is to make it work. You don't think about the possible edge cases."

Thinking like an adversary. One way of thinking about possible vectors is by "having the willingness and motivation just to keep looking for new things. Just because something isn't an issue today, doesn't mean it's not going to be further down the line. Some security engineers are so ingenious with all the things they do it surprises me. I wouldn't have thought to do it but it's really cool how they thought to poke a hole in something" (Clara).

When comparing the mindset between software security engineers and software engineers, Paul explains that "the main difference is having the right mindset for security—having an adversary's mindset—and always thinking about how things are going to break instead of how things are going to perform correctly." Pat reflects, "there's a way of thinking and looking at things that's important for it that not everyone quite manages."

IV. Threats to Validity

Using Maxwell [17], we discuss three practical dimensions of validity that routinely arise in qualitative research: descriptive validity, interpretive validity, and generalizability.

Descriptive validity. Informants may have misremembered or unintentionally distorted their accounts. We adopted a *contextual inquiry* approach from Lutters and Seaman [20], and asked informants to provide specific accounts of recent experiences to mitigate against tendencies to generalize. We asked multiple informants about the same theme and searched for *disconfirming evidence* where statements from one informant would contradict the statements from another. We also had *prolonged engagements* over a period of six months, during which the researchers built trust with various red teams by collaborating with team leads ("gatekeepers"), and establishing rapport with the informants so that they were comfortable disclosing information. Finally, we provided assurances that feedback would remain anonymous and confidential information, such as details regarding undisclosed vulnerabilities, would be scrubbed from any reports.

Interpretive validity. This dimension concerns a misinterpretation in how the researchers understand informants'

statements during thematic analysis, and that the resulting themes are accurate. To reduce misinterpretation, we used *thick description* [21], that is, grounding our findings by relying on the informants' own words whenever possible. We also conducted interviews in pairs, and immediately after each discussed whether our interpretations about the interview were consistent. In cases of remaining uncertainty, we simply went back to the informant using online chat and asked for clarification—a form of lightweight member checking [22].

Generalizability. Semi-structured interviews do not offer external validity in the traditional sense of nomothetic, sample-to-population, or statistical generalization. In place of statistical generalization, our qualitative findings support an idiographic means of satisfying external validity: *analytic generalization* [23]. In analytic generalization, we generalize from informant conversations to themes.

We conducted our study at a single company, and thus the results may not generalize to other organizations. However, Kotulic and Clark [11] found security research to be one of the most intrusive types of organizational research and that "there is undoubtedly a general mistrust of any 'outsider' attempting to gain data" about an organization's security practices. Thus, it is difficult to obtain access to informants from organizations other than our own. We did recruit from five different autonomous red teams at Microsoft. Several informants also had prior experience before joining our organization, and could contrast their prior experiences with their current role.

V. Related Work

Software security activities. Several empirical studies explore different lenses on software security activities. Thomas et al. [24] conduct a broad interview study with application security experts who examine source code, or use off-the-shelf static or dynamic analysis tools, labeling them under the general umbrella of security auditors. Their findings suggest that security auditors and penetration testers are generalists, and able to conduct security activities interchangeably. However, none of the informants in Thomas et al. [24] belonged to red teams, and our study addresses this gap.

In contrast, our findings show that within red teams, software security engineers are agile in their abilities but also deeply specialize in a particular aspect of software security. The use of off-the-shelf analysis tools for red team activities may be useful in organizations that do not have a strong security culture [25, 4, 26], and thus, underutilize security tools.

Hafiz and Fang [27] conducted an empirical study through reports of three prominent security vulnerabilities, including buffer overflows, SQL injection, and cross-site scripting—with the goal of understanding the methods and tools used during vulnerability discovery. We credit Hafiz and Fang [27] for their inventive approach to obtaining informants: the study looks at reported vulnerabilities at SecurityFocus[2] and then contacts the authors of the reports to obtain *first-hand information* about activities relating to security vulnerability discovery (see Section IV). A notable difference in the study methods is that their study investigates offensive security from informants who report vulnerabilities as a diversion or hobby, and so they tend to attack products that are personally interesting to them.

Specialized development roles. We examine the specialized role of software security engineers. Other studies have examined software engineers through the role of testing [28] or as machine learning engineers [29, 30]. Votipka et al. [31] interviewed hackers and testers about how they find vulnerabilities, develop their skills, and the challenges they face. Our findings also include some informants who worked as testers before transitioning to software security engineering; their responses support the findings of Votipka et al. [31].

Gagné et al. [32] examined differences between security professionals and IT professionals; their findings show that compared to other IT, security professionals have to manage a higher level of complexity. Werlinger et al. [33] use participatory observation and interviews to identify activities that require interactions between security practitioners and other stakeholders. The security practitioners within the study have the perspective of IT professionals, not software security engineering. Posey et al. [34] conducted interviews with organizational insiders, that is, employees and security professionals, for differences in their security mindset. Vaughn et al. [35] also observe the undeserved assumption on the effectiveness of perimeter security. These findings triangulate with the "assume breach" mindset employed at large organizations.

VI. Discussion

A. Organizationally Supporting Red Teams

Red teams thrive in a security-aware organization. For example, static and dynamic analysis are part of the continuous integration pipeline at Microsoft, and "software engineers are are not allowed to build unless they have these security tools in place" (Peter). Thus, most issues detected by these approaches are already addressed by software developers before they reach a red team. Because developers are willing to take on additional security responsibilities, red teams are able investigate more sophisticated attack scenarios [36].

Ultimately, as Da Veiga and Eloff [37] find, an "organization's success or failure effectively depends on the things that its employees do or fail to do." When red teams function within a security-aware organization, software security engineers are able to expedite their security campaigns. For instance, red teams at Microsoft typically skip *initial access* tactics such as spearphishing, a tactic often used by malicious adversaries to gain a foothold within a network. This step can be reasonably skipped in a campaign because employees already understand the importance of maintaining their credentials securely.

In addition to conducting offensive campaigns (Section III-A1), red teams at Microsoft offer a number of pathways (Section III-A3) for software engineers to engage more closely with the red teams. One pathway is *design reviews*, where "a red team will meet with a partner before implementation to identify early design problems in the architecture and connections between different components" (Bill). Doing design reviews

[2]https://www.securityfocus.com/

helps catch potential security problems earlier in the process, when the issue can be more easily corrected. A second pathway is *ride-alongs*, where one or more software engineers join the red team to participate in a security campaign against their own product or service. These pathways propagate security knowledge throughout the organization and enable software engineers to take a more active security role.

B. Designing Tools for Software Security Engineering

Informants expressed both a desire for scripting capabilities in tools to enable automation, and for tools that could synthesize source code and binary artifacts to construct explanations.

Recommendation I—Tools should support scripting or extensions to allow for automation. As we show in Section III-A2, the context-sensitive nature of software security requires software security engineers to write their own ad-hoc tools and scripts while conducting security campaigns. In addition, many informants ($n = 11$) expressed a desire to initially explore manually with their tools, but automate their strategies if that manual exploration was productive. Informants felt some tools supported extensibility well, including Wireshark and IDA Pro; however, one criticism of many scripting extensions was that they required learning a proprietary language specific to that tool (Pat, Cliff, Eric). This suggests that tools support extensibility, but also use a common language for extension.

Recommendation II—Tools should support both comprehension of code and explanation of code. The tools that software security engineers use are similar to the tools software engineers use, but they use them in different ways. Rather than using integrated development environments and debugging tools for writing code, our informants primarily used these tools to help understand code written by others. Informants reported that code comprehension was mostly a manual process, and had available to them very few tools to assist, other than basic navigation. Instead, informants had to write ad-hoc scripts to overcome these limitations. For example, Evan built a plugin that improved highlighting by allowing him to highlight multiple variables at once.

As we discussed in Section III-A3, red teams have to communicate their findings to software engineers, often through reports. To generate these reports, informants manually collected code from multiple different projects and then stitched the relevant code snippets together to construct logical explanations, interleaving their own commentary between the code snippets. In some cases, they would rely on text or ASCII diagrams as part of their explanation. For red teams, program comprehension and program explanation are first-class activities, and proper tools need to be developed to support them. These include code search tools that work across multiple representations, such as disassembled binaries or obfuscated code; code explanation and bookkeeping tools for interleaving text annotations, diagrams, and code; and tools that facilitate understanding of data flow.

C. Tools as Both Offensive Weapons and Defensive Tools

Software security engineers have a complicated relationship with their tools, due to their dual nature: tools can be used as both offensive weapons, such as by an adversary—a sword, or as a defensive tool to harden security—a shield [38]. As we found (Section III-C1), this causes subtle but important differences in how software security engineers publish tools.

Software security engineers are mindful about how their offensive tools might be used by others. For this reason, security tools are often not shared broadly, or made visible to other teams even within the organization. As a result, the tools that software security engineers use don't receive as much investment, and are therefore less mature in terms of features and stability. Our informants reported not using third-party tools because security engineers "aren't confident there isn't a backdoor unless the tool is verified. We don't trust security tools from the internet unless you control the environment and have reviewed the code" (Brian). This makes it difficult to introduce new security tools. Paradoxically, software security engineers see the introduction of software security tools as potentially reducing the security of their systems.

D. Implications for Practitioners

First, red teams should comprise diverse and complementary skills, rather than organizing teams by functional specializations. It is precisely this diversity that allows red teams to creatively discover vulnerabilities and exploit them in real-world, full-stack scenarios. Second, red team activities are most effective when their campaigns are conducted directly on production systems; such campaigns also continually exercise detection procedures and blue team remediation activities. We recognize that for some organizations, conducting campaigns against production systems is deeply uncomfortable. However, they are essential to mitigate sophisticated threats. Third, as an emerging role, software security engineers in red teams require drastically different metrics and methods of evaluation for assessing success and impact. Although software security engineers write software, the types of activities they conduct—exploitation, identification of novel techniques, and improvements to general security posture—require organizational changes that support these activities and reward them appropriately. Finally, security is a shared responsibility: red teams are best supported within a security-aware organizational culture.

VII. Conclusion

To understand the emerging role of software security red teams, we conducted an interview study with 17 informants. Through these interviews, we found that red teams undertake campaigns with goals and targets that emulate the activities of sophisticated adversaries. To achieve their goals, software security engineers must specialize and maintain the ability to pivot quickly. We found that software security engineers challenge the stereotypes often applied to hackers; our software security engineers come from diverse backgrounds. Unlike malicious adversaries, red teams take seriously the ethical responsibilities of white hat hacking. Although software security engineers have some overlapping responsibilities with security engineers, a key difference is in how they apply a security mindset—thinking like an adversary.

ACKNOWLEDGMENTS

We thank Patrick Malone and Octavian Timofte for supporting this study, and the software security engineers at Microsoft who participated in our interviews.

REFERENCES

[1] M. Ussath, D. Jaeger, and C. Meinel, "Advanced persistent threats: Behind the scenes," in *2016 Annual Conference on Information Science and Systems (CISS)*, Mar. 2016, pp. 181–186.

[2] F. Li, A. Lai, and D. Ddl, "Evidence of advanced persistent threat: A case study of malware for political espionage," in *International Conference on Malicious and Unwanted Software*, Oct. 2011, pp. 102–109.

[3] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber attacks," *Computers & Security*, vol. 72, pp. 212–233, 2018.

[4] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann, "Quantifying developers' adoption of security tools," in *Foundations of Software Engineering (FSE)*, 2015, pp. 260–271.

[5] L. Bilge and T. Dumitraş, "Before we knew it: An empirical study of zero-day attacks in the real world," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 833–844.

[6] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *Communications and Multimedia Security*. Springer, 2014, pp. 63–72.

[7] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[8] T. Maillart, M. Zhao, J. Grossklags, and J. Chuang, "Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs," *Journal of Cybersecurity*, vol. 3, no. 2, pp. 81–90, Oct. 2017.

[9] S. Ransbotham, S. Mitra, and J. Ramsey, "Are markets for vulnerabilities effective?" *MIS Quarterly*, vol. 36, no. 1, pp. 43–64, 2012.

[10] S. Cubitt and R. F. Malina, *Software Studies: A Lexicon*. MIT Press, 2008.

[11] A. G. Kotulic and J. G. Clark, "Why there aren't more information security research studies," *Information & Management*, vol. 41, no. 5, pp. 597–607, 2004.

[12] P. Biernacki and D. Waldorf, "Snowball sampling: Problems and techniques of chain referral sampling," *Sociological Methods & Research*, vol. 10, no. 2, pp. 141–163, 1981.

[13] G. R. Sadler, H.-C. Lee, R. S.-H. Lim, and J. Fullerton, "Recruitment of hard-to-reach population subgroups via adaptations of the snowball sampling strategy," *Nursing & Health Sciences*, vol. 12, no. 3, pp. 369–374, 2010.

[14] C. Noy, "Sampling knowledge: The hermeneutics of snowball sampling in qualitative research," *International Journal of Social Research Methodology*, vol. 11, no. 4, pp. 327–344, 2008.

[15] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *International Software Metrics Symposium (METRICS)*, Sep. 2005, pp. 23–32.

[16] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[17] J. Maxwell, "Understanding and validity in qualitative research," *Harvard Educational Review*, vol. 62, no. 3, pp. 279–301, 1992.

[18] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, pp. 80–106, 2011.

[19] C. Severance, "Bruce Schneier: The security mindset," *Computer*, vol. 49, no. 2, pp. 7–8, Feb 2016.

[20] W. G. Lutters and C. B. Seaman, "Revealing actual documentation usage in software maintenance through war stories," *Information and Software Technology*, vol. 49, no. 6, pp. 576–587, Jun. 2007.

[21] J. Ponterotto, "Brief note on the origins, evolution, and meaning of the qualitative research concept thick description," *The Qualitative Report*, vol. 11, no. 3, 2006.

[22] L. E. Koelsch, "Reconceptualizing the member check interview," *International Journal of Qualitative Methods*, vol. 12, no. 1, pp. 168–179, 2013.

[23] D. F. Polit and C. T. Beck, "Generalization in quantitative and qualitative research: Myths and strategies," *International Journal of Nursing Studies*, vol. 47, no. 11, pp. 1451–1458, 2010.

[24] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security during application development: An application security expert perspective," in *Conference on Human Factors in Computing Systems (CHI)*, 2018, pp. 262:1–262:12.

[25] S. Xiao, J. Witschey, and E. Murphy-Hill, "Social influences on secure development tool adoption: Why security tools spread," in *Computer Supported Cooperative Work (CSCW)*, 2014, pp. 1095–1106.

[26] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, "Can security become a routine? A study of organizational change in an agile software development group," in *Computer Supported Cooperative Work and Social Computing (CSCW)*, 2017, pp. 2489–2503.

[27] M. Hafiz and M. Fang, "Game of detections: How are security vulnerabilities discovered in the wild?" *Empirical Software Engineering*, vol. 21, no. 5, pp. 1920–1959, 2016.

[28] T. Kanij, R. Merkel, and J. Grundy, "An empirical study to review and revise job responsibilities of software testers," in *Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 89–92.

[29] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 162–170.

[30] C. J. Cai and P. J. Guo, "Software developers learning machine learning: Motivations, hurdles, and desires," in *Visual Languages and Human-Centric Computing (VL/HCC)*, 2019, pp. 25–34.

[31] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in *Security and Privacy (SP)*, May 2018, pp. 374–391.

[32] A. Gagné, K. Muldner, and K. Beznosov, "Identifying differences between security and other IT professionals: A qualitative analysis," in *Human Aspects of Information Security & Assurance (HAISA)*, 2008, pp. 69–80.

[33] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, "Security practitioners in context: Their activities and interactions with other stakeholders within organizations," *International Journal of Human-Computer Studies*, vol. 67, no. 7, pp. 584–606, 2009.

[34] C. Posey, T. L. Roberts, P. B. Lowry, and R. T. Hightower, "Bridging the divide: A qualitative comparison of information security thought patterns between information security professionals and ordinary organizational insiders," *Information & Management*, vol. 51, no. 5, pp. 551–567, 2014.

[35] R. B. Vaughn, R. Henning, and K. Fox, "An empirical study of industrial security-engineering practices," *Journal of Systems and Software*, vol. 61, no. 3, pp. 225–232, 2002.

[36] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: Assurance, security, and flexibility," in *Foundations of Software Engineering (FSE)*, 2017, pp. 197–207.

[37] A. Da Veiga and J. Eloff, "A framework and assessment instrument for information security culture," *Computers & Security*, vol. 29, no. 2, pp. 196–207, Mar. 2010.

[38] T. S. Rad, "The sword and the shield: Hacking tools as offensive weapons and defensive tools," *Georgetown Journal of International Affairs*, pp. 123–133, 2015.