Examining Penetration Tester Behavior in the Collegiate Penetration Testing Competition

BENJAMIN S. MEYERS, SULTAN FAHAD ALMASSARI, BRANDON N. KELLER, and ANDREW MENEELY, Department of Software Engineering, Rochester Institute of Technology, USA

Penetration testing is a key practice toward engineering secure software. Malicious actors have many tactics at their disposal, and software engineers need to know what tactics attackers will prioritize in the first few hours of an attack. Projects like MITRE ATT&CK™ provide knowledge, but how do people actually deploy this knowledge in real situations? A penetration testing competition provides a realistic, controlled environment with which to measure and compare the efficacy of attackers. In this work, *we examine the details of vulnerability discovery and attacker behavior with the goal of improving existing vulnerability assessment processes* using data from the 2019 Collegiate Penetration Testing Competition (CPTC). We constructed 98 timelines of vulnerability discovery and exploits for 37 unique vulnerabilities discovered by 10 teams of penetration testers. We grouped related vulnerabilities together by mapping to Common Weakness Enumerations and MITRE ATT&CK™. We found that (1) vulnerabilities related to improper resource control (e.g., session fixation) are discovered faster and more often, as well as exploited faster, than vulnerabilities related to improper access control (e.g., weak password requirements), (2) there is a clear process followed by penetration testers of discovery/collection to lateral movement/pre-attack. Our methodology facilitates quicker analysis of vulnerabilities in future CPTC events.

CCS Concepts: • **Security and privacy → Software and application security;**

Additional Key Words and Phrases: Security, metrics, vulnerabilities, vulnerability assessment, likelihood of discovery, discoverability, cybersecurity competitions, attacker behavior, penetration testing

ACM Reference format:
Benjamin S. Meyers, Sultan Fahad Almassari, Brandon N. Keller, and Andrew Meneely. 2022. Examining Penetration Tester Behavior in the Collegiate Penetration Testing Competition. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 55 (April 2022), 25 pages.
https://doi.org/10.1145/3514040

1 INTRODUCTION

Security is a ubiquitous concern that threatens the confidentiality of company/user data, the integrity of said data, and the overall availability of software systems. The onus of managing security incidents can no longer rest solely on security experts; software engineers must seek out and mitigate security vulnerabilities before they can be exploited. Vulnerability assessment drives design

This work has been sponsored in part by the National Science Foundation grant 1922169 and by a Department of Defense DARPA SBIR program (grant 140D63-19-C-0018).
Authors' address: B. S. Meyers, S. F. Almassari, B. N. Keller, and A. Meneely, 1 Lomb Memorial Drive, Rochester, NY 14623; emails: bsm9339@rit.edu, sa2553@rit.edu, bnk5096@rit.edu, axmvse@rit.edu.

1049-331X/2022/04-ART55 $15.00
https://doi.org/10.1145/3514040

decisions, implementation, and testing processes for both software engineers and security experts (e.g., penetration testers, security response teams, tool developers). Through understanding the causes, consequences, and mitigations for cybersecurity attacks, we can build proper defenses and engineer more secure software.

Popular tools—such as **Common Vulnerability Scoring System (CVSS)** [21], **Common Weakness Scoring System (CWSS)** [18], **Common Vulnerabilities and Exposures (CVE)** [28]—facilitate shared understanding of vulnerabilities by providing unique identifiers and descriptions, as well as characterizing vulnerabilities based on a variety of factors, such as the complexity of their exploits (e.g., required privileges, attack vectors) and their impact on the confidentiality, integrity, and availability of software assets. CWSS and the **Open Web Application Security Project (OWASP)** [54] compute the likelihood of vulnerability discovery, **discoverability**, but recognize that discoverability is difficult to quantify and often subjective. Some researchers have proposed metrics to quantify discoverability using the time of vulnerability discovery, but these metrics are often impractical, since accurate time of discovery is rarely available [20, 34].

In this work, we quantify discoverability by using logs and team reports from the 2019 **Collegiate Penetration Testing Competition (CPTC)** to construct timelines of vulnerability discovery and exploit. We derived three metrics related to vulnerability discoverability from the timelines. We mapped vulnerabilities to **Common Weakness Enumerations (CWE)** [30], which enabled us to group vulnerabilities based on their shared weaknesses, compare our timeline-derived metrics across different groups of vulnerabilities, and examine mitigation strategies. Finally, we translated timeline events from a technical form to a standardized form by mapping them to MITRE ATT&CK™ tactics and techniques, enabling us to glean insights into attacker behavior.

We designed our methodology with automation in mind and experimented with automating MITRE ATT&CK™ mapping. With further automation and a steady stream of new CPTC data each year, our methodology and metrics can be used to study vulnerability discoverability, mitigation impact, and attacker behavior in future competitions, while also observing patterns/changes over time. All of these insights are valuable additions to the security community's collective understanding of vulnerabilities and can help improve vulnerability assessment. We address the following research questions:

RQ 1: Speed of Vulnerability Discovery
Which types of vulnerabilities are discovered faster and more often than others? Metrics: **Mean Time to Vulnerability Discovery (MTVD), Rate of Vulnerability Discovery (RVD)**.

RQ 2: Speed of Vulnerability Exploit
Which types of vulnerabilities are exploited faster than others? Metrics: **Mean Duration of Attack (MDA)**.

RQ 3: Mitigation Impact
Which mitigation strategies are more effective at reducing the attack surface than others?

RQ 4: Tactic Priority
What tactics did teams prioritize during their attack?

The remainder of this work is outlined as follows: In Section 2, we provide background knowledge on cybersecurity competitions, the Collegiate Penetration Testing Competition, and tools used for data collection/analysis. Section 3 describes related work that analyzes competition data

or studies vulnerability discoverability. In Section 4, we describe our methodology for data collection, annotation, and analysis. Section 5 provides a discussion of our results. We discuss limitations of our work, including concerns with the nature of competition data in Section 6. Finally, we summarize our work and discuss future work in Section 7.

2 BACKGROUND

In this section, we provide background knowledge relevant to our work. Notably, we discuss types of cybersecurity competitions, the Collegiate Penetration Testing Competition, and two taxonomies used for data annotation (Common Weakness Enumerations and MITRE ATT&CK™).

2.1 Types of Cybersecurity Competitions

Cybersecurity competitions are tools for providing hands-on, real-time experience with security concepts as well as "educating young individuals who can design and create secure systems and deploy these sophisticated tools to prevent malicious acts [48]." As outlined in Munaiah et al. [35], cybersecurity competitions typically fall into one of the following categories:

- *Capture the Flag:* Participants compete to gain access to specific pieces of information (i.e., flags) that have been secured on competition servers. Example capture the flag competitions include **High School Capture the Flag (HSCTF)**[1] and DEF CON CTF.[2]
- *Challenges:* Participants compete in various challenges designed to assess their security skills. Example challenges include those hosted by the **National Cyber League (NCL)**[3] and the U.S. Cyber Challenge **(USCC)**.[4]
- *Defensive:* Competitors must defend their network against active attacks from a team of security professionals. Competitors must "secure their infrastructure, root out persistent threats, and monitor for malicious activity, all while maintaining an operational network [35]." The **National Collegiate Cyber Defense Competition (NCCDC)**[5] is a popular defensive competition.
- *Enterprise:* These competitions are typically restricted to security professionals as a means of continued training and education.
- *King of the Hill:* Competitors must defend their own networks from attack while actively attacking other competitors' networks. An example king of the hill competition is the **Information Security Talent Search (ISTS)**.[6]
- *Penetration:* Rather than behaving as malicious attackers or as administrators securing a network, competitors in penetration testing competitions take on the role of professional penetration testers tasked with discovering as many vulnerabilities in the network as possible while documenting their findings and providing recommended mitigations for discovered vulnerabilities.

2.2 Collegiate Penetration Testing Competition

The **Collegiate Penetration Testing Competition (CPTC)** [12] is an annual event where teams of undergraduate and graduate students from top universities compete to discover and exploit vulnerabilities in a fictional corporate software/network environment. CPTC has been held since 2015

[1] https://hsctf.com/.
[2] https://oooverflow.io/.
[3] https://nationalcyberleague.org/.
[4] https://www.uscyberchallenge.org/.
[5] https://www.nationalccdc.org/.
[6] https://www.ists.io/.

with competition software environments designed to reflect real-word systems—e.g., medical systems (2016), transportation industry applications (2018), and critical energy infrastructure (2020). CPTC is a penetration testing competition, meaning teams are not attacking or defending against each other and were not allowed to access any other team's competition infrastructure. Each team is given access to a sandboxed copy of the infrastructure and tasked with discovering vulnerabilities, suggesting mitigations, and documenting their process/findings. Competitors' skills and training varies based on their university and their personal experience. No training was provided to teams by the CPTC development team/event organizers.

This work focuses on the collection, annotation, and analysis of data from the 2019 CPTC nationals competition. Details about the 2019 event are included in Section 4.1. We also utilize data from the 2018 nationals competition, which is described in detail in previous work [35, 36]. Briefly, the 2018 nationals competition involved a competition infrastructure for the fictional Wheelz company, which provides ride-sharing services such as Uber or Lyft. Each of the nine competing teams were given access to a sandboxed copy of the competition infrastructure, composed of four subnetworks spanning 46 host machines. Three of the nine teams did not consent to their data being used for research. 119 GB of Splunk log data was generated by the actions of the six consenting teams, totaling 525,946,388 log events.

2.3 Common Weakness Enumerations

Common Weakness Enumerations (CWE) [30] is a community-curated collection of software (and hardware) weaknesses that can be introduced at any point in the software development lifecycle. These technology-agnostic weaknesses (e.g., buffer overflow, SQL injection, insecure defaults) are common causes for specific vulnerabilities outlined in CVE [28]. CWE is organized as a hierarchy of parent-child relationships, with CWE *pillars* at the top, and each pillar containing any number of CWE *bases*, *variants*, or *classes*. A CWE class or base may be the parent of any number of bases or variants, and CWE variants may be the parent of any number of other variants. For example, CWE-759 (*use of a one-way hash without a salt*; variant) is the child of CWE-916 (*use of password hash with insufficient computational effort*; base), which is the child of CWE-327 (*use of a broken or risky cryptographic algorithm*; class), which is the child of CWE-693 (*protection mechanism failure*; pillar). The CWE hierarchy also contains composites—collections of two or more distinct weaknesses that must exist simultaneously for a vulnerability to arise from them (e.g., session fixation). Composites are disregarded in this work.

2.4 MITRE ATT&CK™ Framework

MITRE ATT&CK™ is a curated knowledge base of tactics and techniques used by malicious actors to discover and exploit vulnerabilities in a system [46]. Tactics answer the question, *why did the attacker perform an action?*—the objective of the attack—while techniques (which are organized by tactic) answer the question, *how did the attacker perform an action?*—the approach used for an attack. For example, the discovery (TA0007) tactic collects techniques related to an attacker trying to figure out the environment, such as account discovery (T1087), network service scanning (T1046), and password policy discovery (T1201).

2.5 Splunk

Splunk[7] is an enterprise system that, at its core, facilitates fast and automatic log collection from multiple hosts and sources (e.g., HTTP connections, intrusion detection systems, bash/powershell history). In addition, Splunk enables real-time monitoring (e.g., custom dashboards, alerting,

[7] https://www.splunk.com/.

anomaly detection) and post-mortem analysis (e.g., distributed database indexing/querying, machine learning tools).

2.6 Term-Frequency–Inverse Document Frequency

Term Frequency–Inverse Document Frequency (TF–IDF), an information retrieval algorithm, is a measure of the relative importance (or significance) of a word within a document compared to its frequency of occurrence across all documents in a collection. The motivation behind TF–IDF is to balance the tradeoffs between precision (rejecting irrelevant documents) and recall (retrieving relevant documents) in the informational retrieval domain [43].

3 RELATED WORK

This section discusses related work on vulnerability discoverability and competition data analysis.

3.1 Vulnerability Discoverability

Quantifying the likelihood of a vulnerability being discovered—**discoverability**—is an active task in the software security domain. OWASP computes the likelihood of a vulnerability being discovered and exploited as a function of a variety of factors (e.g., the estimated number of potential attackers, their skill level and motivation, ease of discovery, and ease of exploit) that are assessed subjectively based on expert opinion [54]. CWSS uses likelihood of discovery (along with a number of other metrics) to compute a CWSS score that can then be used to prioritize assessment of software weaknesses. CWSS specifications note that likelihood of discovery is often subjective and difficult to measure [18].

Wilhjelm et al. proposed the promising Time to Vulnerability Disclosure metric—the time between the release date of the first affected version of the vulnerable source code and the date that the vulnerability is disclosed to the public—as a proxy for likelihood of discovery. They noted that using the date of vulnerability discovery would be more accurate, but is often not recorded and opted for the vulnerability disclosure date as it is usually within 30–60 days of discovery [53]. Similarly, Muegge et al. [34] attempted to collect the date of vulnerability discovery in open source projects, but found that common vulnerability databases—such as CVE [28]—while containing information on the introduction of a vulnerability and the time it takes to fix a vulnerability, did not reliably report vulnerability discovery dates. In a study by McQueen et al. involving 491 zero-day vulnerabilities, the discovery date could only be identified for 15 vulnerabilities [20].

This lack of reporting for vulnerability discovery dates is a gap in our collective understanding of vulnerabilities. There is a clear need for a dataset (and metrics) that reliably captures a vulnerability's time of discovery. In this work, we use evidence from Splunk logs and team reports to find the time of discovery for vulnerabilities in the 2019 CPTC infrastructure.

3.2 Cybersecurity Competitions

Cybersecurity competitions—such as the **National Collegiate Cyber Defense Competition (NCCDC)**,[8] the **Information Security Talent Search (ISTS)**,[9] and the **U.S. Cyber Challenge (USCC)**[10]—are effective tools for knowledge-sharing, training the next generation of security experts, and raising awareness of security concerns [3, 10, 44]. Additionally, cybersecurity competitions can be designed to control variables and produce datasets that are valuable to security research, which is an important endeavor given the lack of empirical security data [44]. Previous

[8] https://www.nationalccdc.org/.
[9] https://www.ists.io/.
[10] https://www.uscyberchallenge.org/.

work offers suggestions for designing, implementing, and hosting cybersecurity competitions [11, 49], as well as participating in them [45]. Other previous work has used survey results from competition participants to assess the educational impact of cybersecurity competitions [3].

Previous work studying the data collected during cybersecurity competitions has primarily focused on analyzing network traffic: Bachupally et al. and Boger et al. both used data collected from NCCDC to identify anomalies in network traffic indicating that a network may have been compromised [5, 7]. Mireles et al. used network traffic from NCCDC to demonstrate a new methodology for constructing attack narratives [24]. Moskal et al. used **Intrusion Detection System (IDS)** logs from CPTC to evaluate a new approach to aggregate IDS logs that revealed latent similarities between attacks [33]. Moving away from network traffic, Harrison et al. used system configuration information and survey responses from NCCDC to confirm that commonly suggested security measures do effectively improve the security stature of computer systems/networks [13].

Mirkovic & Peterson [25] provide a methodology for designing small-scale capture the flag competitions to be used in classrooms as a learning activity. These **Class Capture the Flag (CCTF)** exercises differ from normal **capture the flag (CTF)** competitions in that they (1) are targeted at less skilled students, (2) take considerably less time to prepare and to complete, (3) provide students with both defensive and adversarial experience, and (4) incorporate post-mortem discussions. McDaniel et al. [19] created GenCyber CTF competitions to expose high school students in summer camps to security topics. These competitions incorporated "tutorials and hints into the challenges, so participants could learn while doing [19]." Participants competed in 30 challenges spanning topics such as security basics, tool usage, network reconnaissance, and digital forensics. Students received flags upon completion of a challenge. McDaniel et al. found that GenCyber competitions were effective learning tools capable of providing students with computer security knowledge and experience.

Other research involving CTF challenges includes that of Švábenský et al. [47], which examined almost 16,000 written solutions to CTF challenges and found that the majority of challenges focus on cryptography and network security but neglect human aspects of security (e.g., social engineering, cybersecurity awareness). Kucek & Leitner [16] compared eight open source CTF environments and found that they offered very similar functionality, but varied widely in available game configuration options (e.g., time limits, hint penalties).

Most relevant to our work is a study in 2019, which sought to characterize attacker behavior by mapping timeline events from the 2018 CPTC dataset to MITRE ATT&CK™ tactics and techniques [36]. We improve upon the process from Munaiah et al. [36] by observing and utilizing patterns in the data to streamline mappings. Additionally, Munaiah et al. (1) only examined vulnerabilities and timeline events for a single team, (2) did not map vulnerabilities to CWE, (3) did not attempt to automate classification, and (4) did not attempt to measure vulnerability discoverability. To our knowledge, our work is the first to use cybersecurity competition data to study vulnerability discoverability.

4 METHODOLOGY

Figure 1 summarizes our methodology, as discussed in the remainder of this section.

4.1 Step 1: Data Collection

In 2019, 10 teams of college students competed in the CPTC nationals event hosted at the Rochester Institute of Technology on Nov. 22–24, 2019. The 2019 competition centered on the fictional DinoBank network infrastructure, which consisted of six subnetworks: banking (`bank`), corporate (`corp`), three bank branches (`spring`, `gotham`, and `metro`), and Virtual Desktop Environment (`vdi`). Each bank branch had between one and three bank teller machines, one or two workstations, and

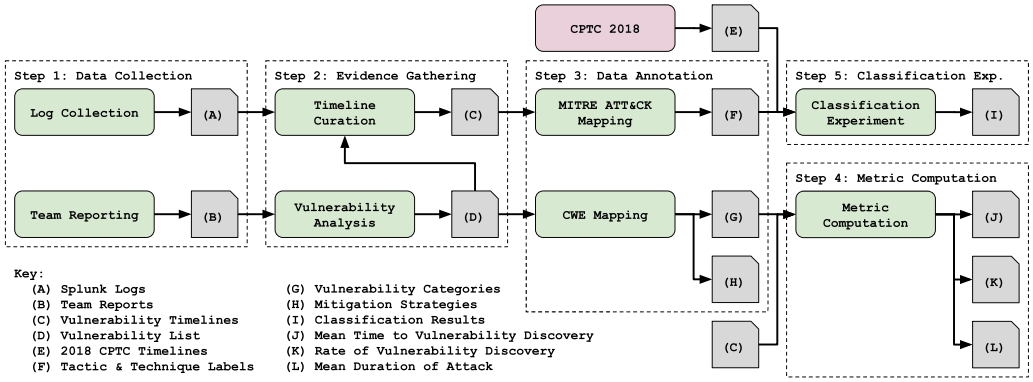

Fig. 1. Methodology.

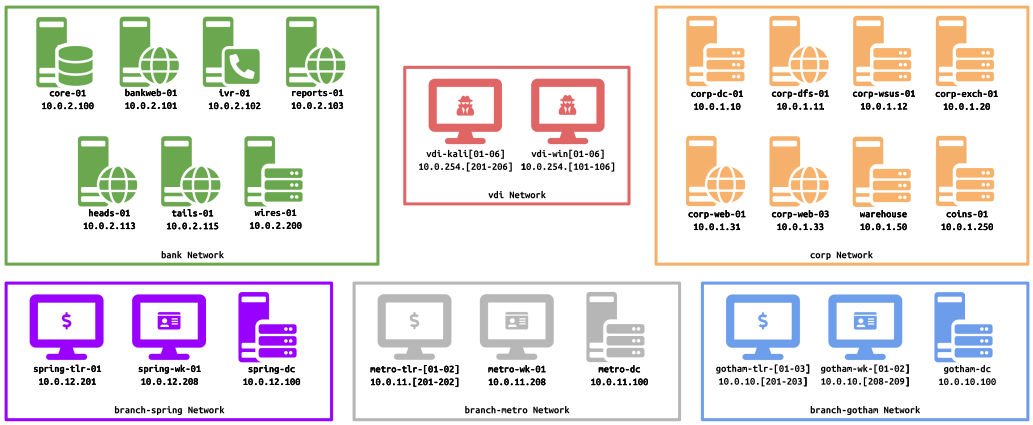

Fig. 2. DinoBank network topology.

Table 1. Summary of DinoBank Subnets and Hosts

	Subnets						
Hosts	**bank**	**corp**	**spring**	**gotham**	**metro**	**vdi**	**Total**
Linux	7	1	0	0	0	7	**15**
Windows	0	7	3	6	4	6	**26**
Total	7	8	3	6	4	13	**41**

a data server. The bank subnet contained seven hosts, including the primary bank website, monitoring/reporting, money transferring, customer service, and a database server. The corp subnet contained eight hosts, including Windows server management, a Microsoft Exchange server, auxiliary website hosts, and the central database server. Teams had control over 12 hosts in the vdi subnet and were tasked with discovering vulnerabilities in the other five subnets. Teams were not allowed to access the competition infrastructure with personal devices. The vdi subnet also had a nameserver that teams were not allowed to access. Table 1 summarizes the fictional DinoBank infrastructure. A network topology for the DinoBank infrastructure is included in Figure 2.

Table 2. Data Summary

Team	Vulnerabilities Discovered	Complete Timelines	Logs Collected
T01	18	7	367,099,634
T02	9	3	464,825,978
T03	14	7	361,042,023
T04	9	5	422,316,180
T05	9	5	438,572,452
T06	6	2	460,129,988
T07	17	7	367,166,162
T08	6	3	420,081,778
T09	6	2	340,866,472
T10	4	4	422,800,522
Total	98	45	4,064,901,189

Competitor machines were monitored and 83 different types of logs (Splunk source types)—bash history, powershell transcripts, network connections, intrusion detection system alerts, and so on—were collected using Splunk. Over four billion logs were collected in Splunk and teams made 98 vulnerability discoveries. We do not know the total number of vulnerabilities that were purposefully implemented in the competition infrastructure, but 37 unique vulnerabilities[11] were outlined in team reports. Those 37 unique vulnerabilities were mapped to CWE (outlined in Section 4.3). Evidence gathering (outlined in Section 4.2) yielded 45 complete attack timelines spanning 19 of the 37 unique vulnerabilities. Table 2 summarizes the data collected.

Teams also submitted reports outlining the vulnerabilities they discovered and the steps they took to discover them (e.g., executed commands, documented decisions, proof of discovery/exploit). The Splunk logs and team reports are critical building blocks for the remainder of our methodology.

4.1.1 Ethical Considerations. We received approval from our university's Institutional Review Board to preform this research involving data collected from human subjects. All data collected is anonymous and contains no personally identifiable information: (1) competitors were not allowed to use personal machines to access the competition infrastructure, (2) Splunk logs and team reports are associated with team numbers, not individual team members, and (3) team numbers cannot be used to identify the students on the team or their university. Additionally, all teams signed a waiver acknowledging that their data would be anonymous and available for research purposes.

4.2 Step 2: Evidence Gathering

A comparison of the 98 vulnerability discoveries revealed 37 unique vulnerabilities. Table 3 shows the 37 unique vulnerabilities discovered and which teams discovered them.

We gathered evidence corroborating discovery and exploitation details in team reports by querying Splunk. The events (logs) in Splunk reveal a wealth of information about the actions teams took during the competition. The most useful of these events revealed which commands teams ran (bash history, powershell history), the duration of those commands (`ps` output), the web pages that teams accessed (HTTP connections), and the internal services that teams connected to (TCP connections).

[11]We disregard a handful of reported vulnerabilities that dealt with the ATM and IVR systems, as logs from those systems were not collected in Splunk.

Table 3. Vulnerabilities Discovered and Exploited

		Teams										
VID	**Description**	**T01**	**T02**	**T03**	**T04**	**T05**	**T06**	**T07**	**T08**	**T09**	**T10**	**Total**
001	Default Credentials in Postgres	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
002	Anonymous FTP Enabled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
003	Weak (MD5) Password Hashing in Postgres	✓										1
004	Arbitrary Code Execution via Postgres	✓	✓		✓	✓	✓	✓	✓	✓		8
005	SSL/TLS Not Implemented	✓		✓		✓	✓		✓		✓	6
006	Plaintext Credentials (Workstations, Powershell Logs, Postgres)	✓	✓		✓	✓		✓				5
007	Weak Passwords & Password Policies	✓	✓		✓			✓				4
008	Improper Access Controls for Wiki	✓	✓		✓							3
009	Sensitive Data Leaked Publicly (Password on Reddit, SSH Private Key on GitHub)	✓	✓					✓		✓		4
010	Improper Access Controls for SMB Pipes	✓						✓				2
011	Cryptocurrency Miner on Workstations	✓			✓	✓						3
012	User Registration Exposes Password	✓					✓		✓			3
013	Missing Access Controls for Main Website	✓		✓								2
014	Clickjacking	✓		✓								2
015	Interactive Logon Enabled for Windows NT Service Accounts	✓										1
016	Missing Email Validation for Account Creation	✓										1
017	Guest Accounts Enabled on Workstations	✓										1
018	Malicious SSL Certificate for Service	✓										1
019	Hard-Coded API Key in Web-Server Root Directory		✓					✓				2
020	Image Uploads (Missing Input Sanitization Leads to Remote Code Execution)		✓	✓				✓				3
021	PHPInfo Page Not Disabled			✓			✓	✓				3
022	Lack of Database Transaction Protections (Transfer of Funds from Account into Itself)			✓		✓		✓				3
023	Username Enumeration via QueryTree Registration			✓								1
024	Web Server Directory Indexing Enabled			✓				✓				2
025	End-of-Life Linux kernel			✓								1
026	Duplicate Session ID's Across Logins			✓								1
027	Replay Attack After Logout (Logout Does Not Purge Session Cookies)			✓								1
028	Bank Account Enumeration via Fund Transfer Page			✓								1
029	Default Credentials on Internal Wiki				✓						✓	2
030	Malicious Scheduled Tasks (Disable Firewall/Antivirus/Encryption)				✓							1
031	Network Infrastructure Diagram on Internal Wiki					✓				✓		2
032	Developer Notes (Revealing Database Schema) in Website Source Code					✓						1
033	Reflected Cross-Site Scripting (XSS)							✓	✓	✓		3
034	Missing Access Controls for QueryTree Service							✓				1
035	SQL Injection in OpenTrade Service							✓				1
036	Cross-Site Request Forgery (CSRF)							✓				1
037	Username Enumeration via Kerberos Pre-Authentication							✓				1

While our approach is similar to previous work by Munaiah et al. [36], we observed patterns in the data that allowed us to streamline the process. The types of Splunk queries vary based on the actions that teams took and the tools/commands that they used, but the following processes were derived based on our observations and facilitated faster manual evidence gathering:

- **Network Service Scanning:** All exploits begin with the team scanning the network for hosts and services running on those hosts. This typically involved teams running `nmap` to enumerate hosts/services on specific subnets. For example: `nmap -T4 -sV 10.0.2.0/24 -vv`. During evidence gathering, we queried Splunk for a bash history event showing the

command used for network service scanning and then verified that the command actually ran by finding a `ps` event with the identical command running.

- **Remote Connections:** Whenever a team report indicated a connection to a target host, we queried Splunk for an outgoing connection from the `vdi` subnet to the target host and an identical incoming connection to the target host. For example, if a team connected from 10.0.254.203 (team host) to 10.0.1.12 (target host) over FTP (port 21), then we would search Splunk for an outgoing TCP connection logged on the team host and then search for a matching incoming TCP connection logged on the target host.

Using timestamps from Splunk events, we were able to build timelines of discovery and exploit for each vulnerability. As an example, we will outline the process Team 01 took to discover and exploit vulnerability 004 (arbitrary code execution via Postgres). The report shows that Team 01 discovered a PostgreSQL database running on `10.0.2.100` (target host) and was able to authenticate to the database using default credentials via the command `psql -h 10.0.2.100 -U postgres`. Using Splunk, we found evidence of the team performing network service scanning to map the potential inputs into the target host's subnet: `nmap -n 10.0.2.0/24 -T5 -oA pingscan` (bash history and `ps` output). We also verified that the team connected to the database using the command outlined in the report: `psql -h_10.0.2.100_-U_postgres` (`ps` output). Next, the team created a shell script (`k.sh`) that runs some Python code:

```
#!/bin/sh
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.
connect("10.0.0.1",1234));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);
=subprocess.call(["/bin/sh","-i"]);'
```

Bash history logs in Splunk reveal the team creating `k.sh`: `vim /tmp/k.sh`. The team proceeded to run the commands `python -m SimpleHTTPServer 80` and `nc -l 8888 -v`; the former verified from bash history logs; the latter could not be verified, but we consider this timeline complete, as the time of discovery and the time of exploit do not rely on whether that command was executed. The team then ran the following command inside of the PostgreSQL shell: `COPY cmd_exec FROM PROGRAM 'wget 10.0.254.204/s.sh -O /tmp/k.sh; bash /tmp/k.sh &';`. While we cannot see Postgres shell commands due to a limitation in logging capabilities, periodic logs of the output of `ps` revealed that the code inside `k.sh` was executed by the `postgres` user on the target host. All of this evidence from Splunk is timestamped, allowing the creation of the timeline in Table 4.

4.3 Step 3: Data Annotation

4.3.1 MITRE ATT&CK™ Mapping. The timelines provide a wealth of information, but insights are not immediately accessible due to the highly specific nature and granularity of the log data. To obtain more generalized insights, we manually mapped each event in a timeline to a MITRE ATT&CK™ tactic and technique using a similar approach to Munaiah et al. [36]. In this section, we outline the mapping decisions we made with examples. Due to the nature of the competition, we removed the following tactics from consideration:

- **Initial Access:** Since teams were granted access to the network infrastructure, they did not need to obtain initial access.
- **Resource Development:** Any custom tool/script implementation that teams may have done would have been completed prior to the start of the competition.
- **Defense Evasion:** Since teams were given permission to discover vulnerabilities in the competition infrastructure, they did not need to evade detection.
- **Exfiltration:** Teams were explicitly forbidden from exporting any data from the competition infrastructure.

Table 4. Example Timeline of Vulnerability Discovery and Exploit

Epoch Time	Log Type	Log	Description
1574473453	bash_history	`nmap -n 10.0.2.0/24 -T5 -oA pingscan`	Network service scanning.
1574473588	ps	`root 30283 ... nmap -sn_10.0.2.0/24_-oG_2-hosts.txt`	Network service scanning.
1574475207	ps	`root 22092 ... psql -h_10.0.2.100_-U_postgres`	Authentication to remote Postgres server using default credentials.
1574475214	stream:tcp	`{''src_ip'':''10.0.254.201'', ..., ''dest_ip'':''10.0.2.100'', ''dest_port'':5432}`	Verification of remote connection in previous event.
1574475402	bash_history	`vim /tmp/k.sh`	Creation of `k.sh` bash script.
1574475486	bash_history	`python -m SimpleHTTPServer 80`	Opening a Python HTTP server.
1574475492	ps	`root 27660 ... python -m_SimpleHTTPServer_80`	Opening a Python HTTP server.
N/A	N/A	`nc -l 8888 -v`	Starting a Netcat listener.
N/A	N/A	`COPY cmd_exec FROM PROGRAM 'wget 10.0.254.204/s.sh -O /tmp/k.sh; bash /tmp/k.sh &';`	Postgres query that executes arbitrary code (`k.sh`).
1574475505	ps	`postgres 26363 ... python -c_import_socket ... p=subprocess.call([''/bin/sh'',''-i'']);`	Contents of `k.sh` executed by `postgres` user on target host.

We used the following approach for mapping timeline events to tactics and techniques:

(1) **Infer Attacker Objective:** Given a timeline event and its associated metadata, we inferred the objective of the attacker. For example, consider the first event in Table 4. The team used `nmap`, a network scanning tool, targeted at a competition subnet. We can infer that the objective of this event is to enumerate hosts/services on the target subnet.
(2) **Assign Tactic:** Given the inferred objective of a timeline event, we assigned the most appropriate tactic. Since the `nmap` event in our example both scans the subnet and saves the output to disk (`-o` flag), we assign two tactics to this event: *discovery* and *collection*. Mapping to multiple tactics, when applicable, allows us to capture a more complete picture of attacker behavior. If the `nmap` event was not saving the output to disk, then we would only have assigned the *discovery* tactic.
(3) **Assign Technique:** Each tactic has an associated set of techniques. After assigning a tactic for an event, we searched the associated techniques for the one that most closely matched what the attacker did. Since `nmap` is a network scanning tool, the most appropriate *discovery* technique is *network service scanning*. However, since the example event was also mapped to the *collection* tactic, we also assigned the *automated collection* technique.

Here is a summary of our common mapping decisions, which should serve as precedents for future CPTC analysis:

- **Network scanning a host/subnet** was mapped to the *discovery* and *collection* tactics and the *network service scanning* and *automated collection* techniques when the scan results were saved to disk. Otherwise, we mapped to just *discovery* and *network service scanning*.
- **Connecting to a service on another host** (e.g., Postgres, ssh) was mapped to the *lateral movement* tactic. Technique mapping depended on the actual method of connection; typically *valid accounts* and/or *remote services*. We recognize that *valid accounts* is an *initial access* technique, but when teams were not using *remote services* for *lateral movement*, *valid accounts* is the most appropriate description in the MITRE ATT&CK™ framework for their activities.
- **Collection of data from a web service** (e.g., `phpinfo`) was mapped to the *collection* tactic and the *data from local system* technique.

- **Running code-based exploits** (e.g., XSS) was mapped to the *execution* tactic. Technique mapping varied depending on the nature of the event.
- Any event with the objective of **privilege escalation** (e.g., bypassing broken authorization) was mapped to the *privilege escalation* tactic. Technique mapping varied depending on the nature of the event.
- When we found evidence of teams **manually visiting web pages looking for sensitive information**, we mapped these events to the *discovery* and *collection* tactics. However, we could not find an appropriate technique in the MITRE ATT&CK™ framework, so we labeled the technique for these events *manual search for sensitive information in webpages*.
- When we found evidence of teams **preparing for attack** (e.g., moving files around, writing quick one-use scripts), these events were mapped to "pre-attack" for both tactic and technique, since we could not find an adequate mapping in the MITRE ATT&CK™ framework.
- **TCP/HTTP connections and ps logs verifying another log event** were mapped to the same tactic and technique.

One of the authors completed the mapping process for all timeline events. The mappings were reviewed by another author and disagreements were discussed until agreement was reached.

4.3.2 CWE Mapping. During vulnerability assessment, comparing vulnerabilities to software weaknesses (e.g., CWE [30]), attack patterns (e.g., CAPEC [27]), and related vulnerabilities (e.g., CVE [28]) enables software engineers and other security practitioners to learn from past security mistakes. In this work, we mapped vulnerabilities from team reports to CWE, which enabled us to group related vulnerabilities together for analysis and study their mitigation strategies.

The CWE project outlines an explicit process for mapping vulnerabilities to CWE entries [31]. Two of the authors followed this process closely, using the "view" method for mapping CVE to CWE suggested by CWE. The process we followed was:

(1) **Choose View:** Choose a CWE view (one of software development, hardware design, or research concepts) to start with. For the majority of vulnerabilities, we used the Research Concepts view, as it is the most complete.
(2) **Identify Pillar:** Read the descriptions for the higher-level categories (called pillars) within the chosen view and identify those that are related to the vulnerability in question.
(3) **Assign CWE:** Traverse the children (CWE classes, bases, and variants) of each CWE pillar, reading their descriptions. Continue recursively traversing the parent-child relationships between CWEs until the most relevant/applicable CWE for the vulnerability in question has been identified.

For example, vulnerability 001—default credentials in Postgres—is related to insecure defaults. Starting from the Research Concepts view in CWE, the most closely related pillar is CWE-664 (improper resource control). Expanding the pillar, we traversed the immediate child classes, bases, and variants. We determined that CWE-665 (improper initialization) was the most relevant. We then explored the immediate children of CWE-665 and determined that CWE-1188 (insecure default initialization) was the most relevant. After examining the children of CWE-1188, we determined that they did not accurately capture vulnerability 001, so we stopped our CWE mapping at CWE-1188.

If CWE names and descriptions did not contain enough information for immediate classification, then we searched CVE and **Common Attack Pattern Enumeration and Classification (CAPEC)** [27] databases for keywords related to the vulnerability in question. Many CVEs and CAPECs link to related CWEs, narrowing down the search space. For example, none of the CWE pillar descriptions allowed us to immediately map vulnerability 018—malicious SSL certificates—so

Table 5. Vulnerability Categories

Category	Vulnerabilities	Discoveries
CWE-284	006, 007, 008, 010, 013, 015, 017, 018, 019, 029, 034, 037	25
CWE-664	001, 002, 009, 012, 014, 021, 023, 024, 026, 027, 028, 031, 032	42
CWE-693	003, 005, 016, 036	9
CWE-707	004, 020, 033, 035	15
CWE-710	025	1
CWE-840	022	3
Insider Threat	011, 030	2

we searched CAPEC and found CAPEC-479 (malicious root certificate). The entry for CAPEC-479 linked to CWE-284 (improper access control). Using that information, we continued the mapping process as normal.

Two of the authors independently completed mapping vulnerabilities to CWE. We computed inter-rater reliability (Cohen's $\kappa = 0.39$) using scikit-learn [40]. A Cohen's $\kappa = 0.39$ is considered low. However, if we consider the hierarchical nature of CWEs and trace the authors' independent mappings to their respective pillars, then we can recompute inter-rater reliability for pillars (Cohen's $\kappa = 0.67$). This indicates that while mapping to specific CWE bases/variants is somewhat subjective, mapping to CWE pillars is less subjective. After independent mapping to CWE, the authors discussed their mappings and resolved any disagreements.

We utilized the pre-existing, hierarchical CWE framework to determine higher-level categories (see Figure 3). This resulted in six vulnerability categories for analysis: **CWE-284:** Improper Access Control, **CWE-664:** Improper Control of a Resource Through its Lifetime, **CWE-693:** Protection Mechanism Failure, **CWE-707:** Improper Neutralization, **CWE-710:** Improper Adherence to Coding Standards, and **CWE-840:** Business Logic Errors. Additionally, we classified two vulnerabilities (011, 030) as insider threat, which is not captured in CWE. Our vulnerability categories are outlined in Table 5. The vulnerabilities in this dataset span 60% of the high-level CWE pillars.

After disregarding categories with less than 10 complete timelines, we are left with the following vulnerability categories for analysis:

- **CWE-284:** 8 vulnerabilities (006, 008, 010, 017, 018, 029, 034, 037), 13 complete timelines
- **CWE-664:** 6 vulnerabilities (001, 002, 021, 023, 027, 031), 24 complete timelines

4.4 Step 4: Metric Computation

As discussed in Section 3.1, quantifying vulnerability discoverability is an active task in the software security domain. Existing metrics are either subjective or incomplete. We drew inspiration from the field of software reliability testing to define three metrics in the spirit of Mean Time to Failure [41]. We defined two discoverability metrics (Mean Time to Vulnerability Discovery, Rate of Vulnerability Discovery) and one exploitability metric (Mean Duration of Attack) to aid in summarizing and understanding attacker behavior in this dataset.

4.4.1 Discoverability Metrics. For each vulnerability in a category, we measured the time from the beginning of the competition (epoch time 1574467200) to the first non-network service scanning event in the vulnerability's timeline. For example, when discovering vulnerability 001, Team 06 scanned the bank network for services at epoch time 1574475502. The difference between those timestamps (the time to discovery) is 138.37 minutes. We call the average of each time to discovery for vulnerabilities within a category the **Mean Time to Vulnerability Discovery (MTVD)**. We

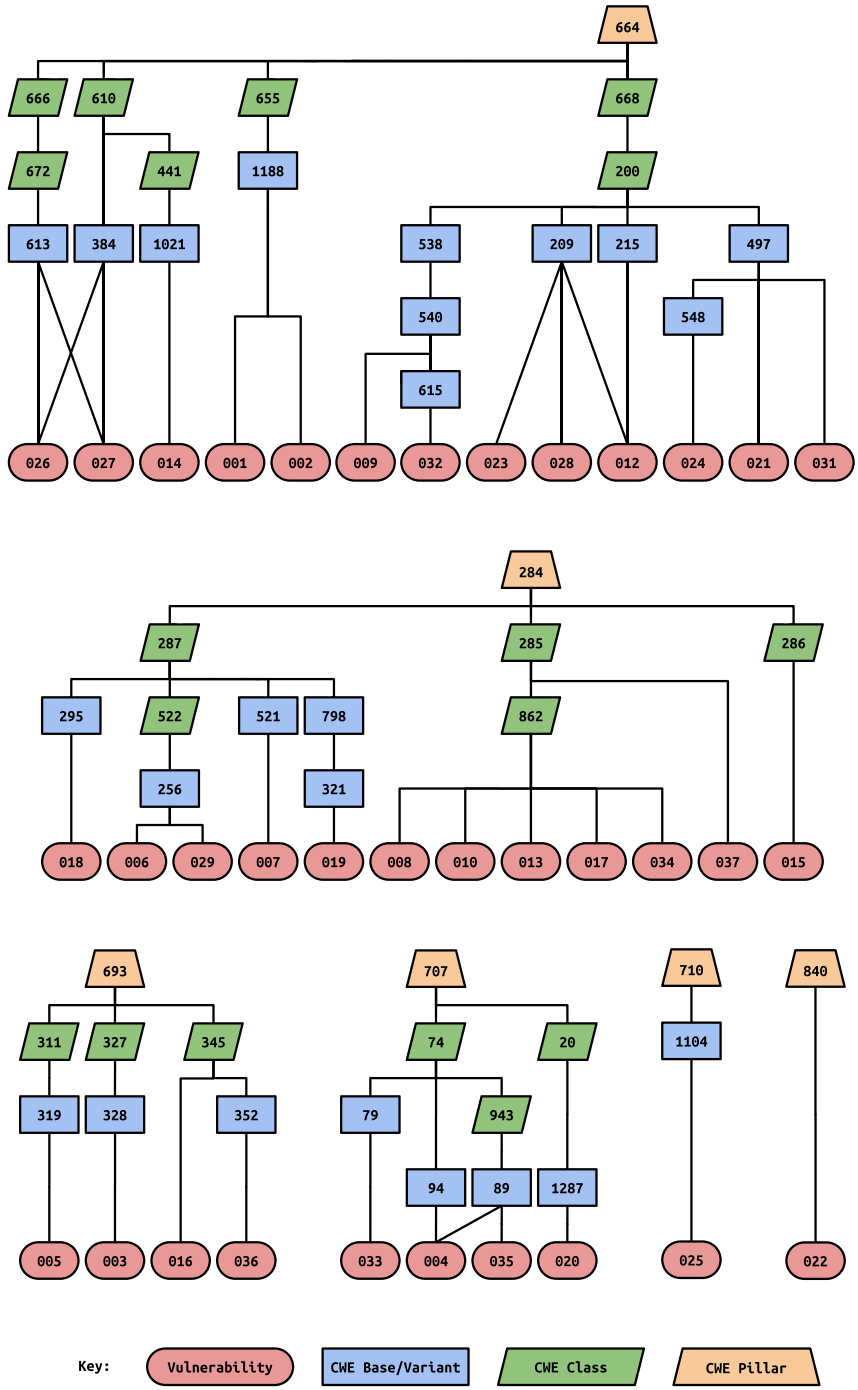

Fig. 3. Mapping from vulnerability to CWE, as described in Section 4.3.2. For each vulnerability (red ovals), we identified the most relevant CWE pillar (orange trapezoids) and then recursively traversed its child CWE classes (green rhombuses) and bases/variants (blue rectangles) until we found the most applicable CWE for the vulnerability was identified.

Table 6. Classification Datasets

Dataset	# Logs	# Unique Tactics	# Unique Techniques
2018	261	23	35
2019	413	10	21
Combined	678	24	46

do not consider the timestamp for the first network service scanning event in a timeline to be the time of discovery, because evidence suggests that all teams performed network service scanning near the beginning of the competition and stored results for later use.

$$\text{MTVD}_{\text{category}} = \frac{1}{N}\sum_{i=0}^{N}\{t_{\text{discovery}} - t_{\text{start}}\}_i \tag{1}$$

Each vulnerability was found by at least one team and at most 10 teams. We call the sum of the number of teams that found each vulnerability in a category the total discoveries. For example, in CWE-707, two different teams found vulnerability 004 and one team found vulnerability 033, making the total discoveries equal to three. The **Rate of Vulnerability Discovery (RVD)** for a category is the total discoveries for the category divided by the product of the number of vulnerabilities in the category times 10 (the number of teams that could have discovered each of the vulnerabilities).

$$\text{RVD}_{\text{category}} = \frac{\text{\# vulnerability discoveries}}{\text{\# vulnerabilities} \times \text{\# teams}} \tag{2}$$

A lower MTVD and a higher RVD represents a more easily discoverable vulnerability category.

4.4.2 Exploitability Metric. For each timeline, the duration of attack is the time from the first non-network service scanning event (the time of discovery) to the last timeline event, and the **Mean Duration of Attack (MDA)** for a category is the average of the duration of attack for each vulnerability within a category. A lower value for MDA indicates a more easily (or quickly) exploitable category of vulnerabilities.

$$\text{MDA}_{\text{category}} = \frac{1}{N}\sum_{i=0}^{N}\{t_{\text{end}} - t_{\text{discovery}}\}_i \tag{3}$$

4.5 Step 5: Classification Experiment

Manual mapping of timeline events to MITRE ATT&CK™ tactics and techniques requires a considerable amount of human effort, as well as some subjectivity. To streamline analysis of future CPTC data, we experimented with automated classification of tactics and techniques for timeline events, using **Linear Support Vector Classifiers (LSVC)** and **Logistic Regression (LR)** classifiers, both with and without **Recursive Feature Elimination (RFE)**. In total, four models were trained for each of the three datasets outlined in Table 6. For model features, we tokenized and transformed raw Splunk logs into **Term Frequency–Inverse Document Frequency (TF–IDF)** matrices. While TF–IDF features are typically used for natural language classification, we took inspiration from Aussel et al., who used natural language processing based features for mining log data [4].

Classification results (shown in Table 7) were poor (<40%) when models were trained on 2018 timeline events and tested on 2019 timeline events. We saw some improvement when using 2019 timeline events for training and 2018 timeline events for testing, suggesting the need for more training data from future competitions before we can automatically map timeline events to tactics and techniques. We saw significant improvement (precision, recall, f1 > 85%) when training models

Table 7. Classification Performance on Validation Data

		Train 2018/ Test 2019			Train 2019/ Test 2018			Train 2018+2019/ Test Leave-One-Out CV		
	Model	**precision**	**recall**	**f1**	**precision**	**recall**	**f1**	**precision**	**recall**	**f1**
Tactics	**LSVC**	0.13	0.20	0.13	0.37	0.23	0.26	0.85	0.91	0.87
	LSVC+RFE	0.09	0.29	0.13	0.00	0.09	0.01	**0.87**	**0.92**	**0.89**
	LR	0.25	0.38	0.28	0.31	0.08	0.11	0.86	0.88	0.87
	LR+RFE	0.00	0.00	0.00	0.01	0.09	0.01	0.86	0.91	0.88
Techniques	**LSVC**	0.12	0.30	0.16	0.31	0.21	0.24	0.79	0.84	0.80
	LSVC+RFE	0.00	0.00	0.00	0.03	0.00	0.10	0.79	0.83	0.79
	LR	0.12	0.31	0.18	0.25	0.23	0.27	**0.87**	**0.90**	**0.88**
	LR+RFE	0.07	0.08	0.07	0.39	0.12	0.19	0.80	0.86	0.83

Table 8. Results for Discoverability/Exploitability Metrics

Metric Type	**Metric**	**Units**	**Most**	**Least**
Discoverability	**MTVD**	Minutes	CWE-664 (517.45)	CWE-284 (596.43)
	RVD	Ratio	CWE-664 (0.400)	CWE-284 (0.163)
Exploitability	**MDA**	Minutes	CWE-664 (10.65)	CWE-284 (279.44)

on both 2018 and 2019 timeline events and evaluating using leave-one-out cross validation, further suggesting the need for more training data. We discuss more improvements in Section 7.1.

5 RESULTS

We computed our discoverability (MTVD, RVD) and exploitability (MDA) metrics for each vulnerability category using only the completed timelines within each category. Table 8 shows a ranking of the metric scores (from most discoverable/exploitable to least) for each of the vulnerability categories. Vulnerabilities in CWE-664 include improper session handling (e.g., session fixation), insecure defaults, and sensitive information leakages (e.g., code comments, error messages). Vulnerabilities in CWE-284 include improper authentication (e.g., weak password requirements, passwords stored in plaintext, improper certificate validation) and lack of authorization mechanisms.

5.1 RQ1: Speed of Vulnerability Discovery

RQ 1: **Which types of vulnerabilities are discovered faster and more often than others?**

Results: Improper resource control (CWE-664) vulnerabilities were discovered faster—in terms of Mean Time to Vulnerability Discovery—and more often—in terms of Rate of Vulnerability Discovery—than improper access control (CWE-284) vulnerabilities.

The likelihood of vulnerability discovery is influenced by a variety of factors ranging from human behavior to technical reasons. One might expect that initial access (via authentication) and authorization would be required before accessing sensitive resources, but our results, and CWEs ranking, suggest otherwise.

One class of vulnerabilities in the CWE-664 category is CWE-200—exposure of sensitive information to an unauthorized actor—which encompasses seven vulnerabilities from CPTC team reports. The CWE website curates a list of the top most dangerous software weaknesses, ranked by prevalence and severity (based on CVE and CVSS data). CWE-200 was ranked #4 and #7 most

dangerous in 2019 [26] and 2020 [29], respectively. In the CWE-284 category, CWE-862—missing authorization—encompasses five vulnerabilities from team reports and was ranked #34 and #25 most dangerous in 2019 and 2020, respectively. Another ranking weakness from CWE-284 is CWE-287—improper authentication—which encompasses five vulnerabilities from team reports and was ranked #13 and #14 most dangerous in 2019 and 2020, respectively. Since vulnerabilities in CWE-664 are consistently dangerous and discovered faster/more often than vulnerabilities in CWE-284, mitigation of CWE-664 vulnerabilities should be prioritized. Our findings provide empirical support for some of CWE's top most dangerous weakness rankings, which is valuable, since CWE's rankings are based on sometimes incomplete CVE and **National Vulnerability Database (NVD)** [8] entries as well as often subjective CVSS scores [14, 51]. Our results strengthen the argument that vulnerabilities falling under CWE-664 need to be prioritized over those vulnerabilities that fall under CWE-284.

5.2 RQ2: Speed of Vulnerability Exploit

RQ 2: **Which types of vulnerabilities are exploited faster than others?**

Results: Improper resource control (CWE-664) vulnerabilities are exploited faster (more easily) than improper access control (CWE-284) vulnerabilities in terms of Mean Duration of Attack.

Not only are vulnerabilities in CWE-664 more dangerous and discovered faster/more often, but they are also exploited more easily than vulnerabilities in CWE-284, making prioritization of mitigations for CWE-664 vulnerabilities even more important.

A reasonable assumption is that vulnerabilities that can be exploited via automated tools would be exploited faster than vulnerabilities that require manual human effort. However, our results indicate the opposite: automated tools could exploit vulnerabilities in authentication and authorization mechanisms (CWE-284), but discovering and exploiting vulnerabilities in CWE-664 (e.g., insecure default passwords/configurations, sensitive data leaks, account enumeration) requires manual human time and effort. Despite the need for human effort, vulnerabilities in CWE-664 were discovered and exploited faster than vulnerabilities in CWE-284.

5.3 RQ3: Mitigation Impact

RQ 3: **Which mitigation strategies are more effective at reducing the attack surface than others?**

Results: Out of 62 unique mitigation techniques suggested by CWE, 51 only apply to one or two vulnerabilities in our dataset and 59 are not applicable outside of a vulnerability category.

Each CWE includes a list of potential mitigations that have been suggested and curated by security experts. Using this information, we examined mitigations that apply to multiple vulnerabilities. In total, we examined 62 unique mitigation strategies spanning the 37 vulnerabilities discovered by penetrating testing teams. Note that we consider all vulnerabilities[12] for all teams in our analysis of mitigation strategies, not just vulnerabilities with complete timelines.

[12]Our analysis does not include mitigations for vulnerabilities that we could not map to CWE (i.e., Insider Threat). No mitigations were suggested for CWE-840.

Fifty-one out of 62 mitigation strategies only applied to one or two vulnerabilities, further validating the need to apply multiple mitigations to adequately protect a system from vulnerabilities and exploits. We also found that 59 out of 62 mitigation strategies are not applicable outside of a vulnerability category, suggesting the need to apply multiple *unrelated* mitigations to properly secure a system.

For improper access control (CWE-284) vulnerabilities, mapping user roles to data/functionality, performing access control checks related to business logic, utilizing pre-established authorization libraries/frameworks, performing server-side access control checks, and utilizing OS access control mechanisms were the most effective mitigation strategies, covering 6 of the 11 vulnerabilities in the category. We found that sanitizing error messages, handling exceptions internally (rather than displaying them to the user), and reducing the attack surface are the most effective mitigation strategies for improper resource control (CWE-664) vulnerabilities, but these mitigations only address 3 of the 13 vulnerabilities in the category. For protection mechanism failure (CWE-693) vulnerabilities, no single mitigation strategy addressed more than one vulnerability in the category. Finally, for improper neutralization (CWE-707) vulnerabilities, assuming that all input is malicious and implementing input validation could have mitigated all four vulnerabilities in the category.

5.4 RQ4: Tactic Priority

RQ 4: **What tactics did teams prioritize during their attack?**

Results: In the first three hours of the competition, teams discovered 21 vulnerabilities using primarily discovery, collection, and lateral movement tactics. In the second three hours, teams discovered 22 more vulnerabilities using collection, lateral movement, and pre-attack tactics. The last seven hours of the competition saw sporadic collection, lateral movement, and pre-attack and only two vulnerability discoveries.

Teams were allowed access to the competition infrastructure from 7–10PM EST on Friday, November 22nd and 9AM–7PM EST on Saturday, November 23rd. Figure 4 shows the transitions between MITRE ATT&CK™ tactics in timelines. Figure 5 shows timelines of MITRE ATT&CK™ tactics used by each team throughout the competition.

Most teams began by discovering their environment and/or collecting information to be used later in the exploit. Some teams began trying to traverse through the competition infrastructure (lateral movement) on Friday evening, and a few attempted to escalate their privileges and/or began preparing to exploit vulnerabilities (pre-attack). When the competition resumed Saturday morning, teams finished up the discovery phase and ramped up collection, lateral movement, and pre-attack techniques. Of the total 45 complete timelines, 21 vulnerabilities were discovered on Friday evening and 22 more were discovered in the first three hours on Saturday morning.

The MITRE ATT&CK™ framework has 215 techniques spanning 14 tactics. If we exclude the techniques belonging to tactics that we did not map to, then we are left with 112 techniques. Teams only used 21 unique techniques, the most common being network service scanning, automated collection, and lateral movement via valid accounts and remote services. This suggests that mitigations for these activities should be prioritized.

5.5 Summary of Results

In summary, we found that improper resource control (CWE-664) vulnerabilities were discovered faster and more often and exploited faster than improper access control (CWE-284) vulnerabilities. Additionally, we found that teams followed a similar process for vulnerability discovery and exploit; teams performed most of their discovery, collection, and lateral movement tactics within

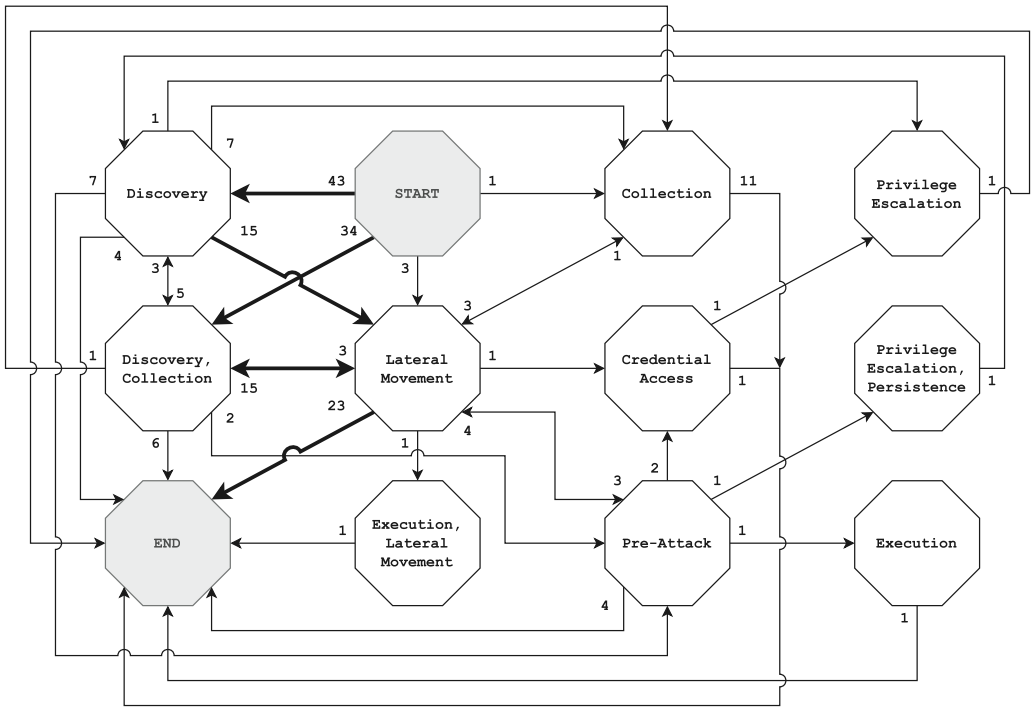

Fig. 4. Transitions between MITRE ATT&CK™ tactics (octagons). Frequencies denoted at the base of each transition. Thick lines: frequency ≥ 10.

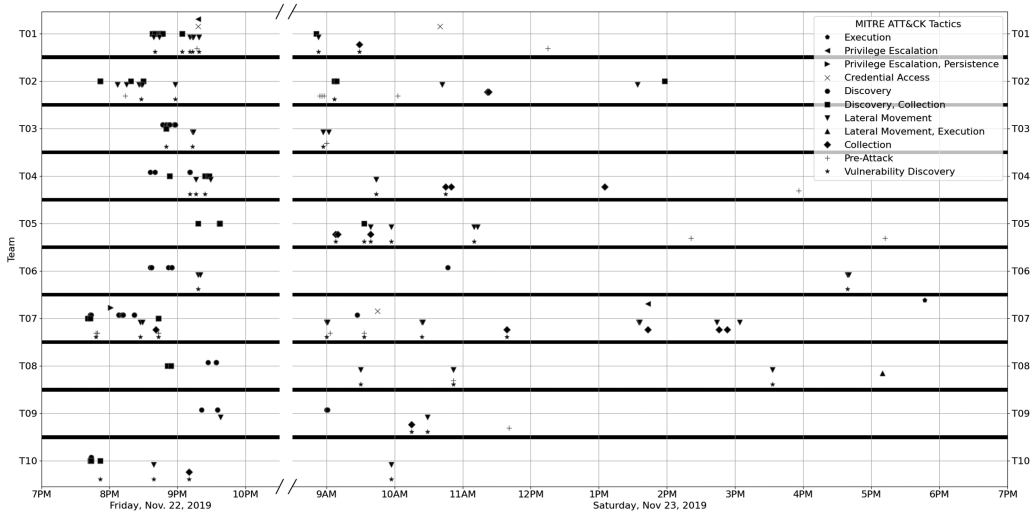

Fig. 5. Timeline of MITRE ATT&CK™ tactics performed by each team. Each point denotes the start time for a given tactic.

the first three hours before transitioning away from discovery to pre-attack in the second three hours. Finally, we found that the majority of mitigations recommended by CWE only apply to one or two vulnerabilities; multiple mitigations spanning different categories of vulnerabilities need to be implemented to properly secure a system.

6 LIMITATIONS

6.1 Does Competition Data Reflect the Real World?

There is a wide range of cybersecurity competitions—defensive, capture-the-flag, king-of-the-hill, penetration, and challenges—all with their own purposes and goals. A common concern is whether cybersecurity competitions reflect the real world. More specifically, there are concerns about (1) the goals of competitions being unrealistic, (2) the compressed timeline of events, (3) the contrived nature of competition environments, and (4) treating college students as a proxy for security experts when analyzing competition data. We discuss these concerns below. In lieu of a real organization releasing the results of penetration testing for their infrastructure (beyond network packet captures), we believe the data from this competition is the best available and contains a wealth of potential insights that should not be ignored.

6.1.1 Realism of Goals. As stated on the CPTC website, the explicit goal of the competition is "*mimicking the activities performed during a real-world penetration testing engagement conducted by companies, professional services firms, and internal security departments around the world* [12]." The goal of real-world penetration testing is to discover as many vulnerabilities as possible, assess their risks, and suggest mitigation strategies [9]. This is not always true for all cybersecurity competitions. Indeed, gaining access to a piece of data for the sake of it (e.g. capture-the-flag), teams actively defending their own network while attacking another (king-of-the-hill), or challenges to test the skills of individuals/teams may not reflect the real-world security game—attackers discovering and exploiting vulnerabilities with malicious intent.

6.1.2 Realism of Timeline. CPTC is intended to be a simulation of the first few hours of a significant cyber attack. In the real world, responding to a security incident is often a high-intensity, overwhelming, stressful, and time-sensitive process [52] of determining where a vulnerability exists, how it was exploited, and how it can be fixed. CPTC is analogous to this in that teams were only given 13 hours to discover and report on as many vulnerabilities as possible.

6.1.3 Realism of Environment. While the legitimacy of competition environments is often under scrutiny, we believe that CPTC has a realistic competition infrastructure. The CPTC development team is a diverse group of industry professionals and academics. They use actual commercial and open source software whenever possible, and when required to write custom software, they research the software that would exist in the industry domain of the competition, often receiving guidance/advice from experts or companies (e.g., Uber in 2018, Eaton in 2020). Many members of the CPTC development team are professional penetration testers that use their experience to design and implement vulnerabilities in the competition infrastructure.

6.1.4 Students vs. Experts. Industry professionals use CPTC (and other competitions) to identify talent in cybsersecurity [6], so the competition covers material relevant to industry. To be invited to a national CPTC competition, teams first have to compete in and win a regional CPTC event, ensuring that these penetration testing teams are top-tier in terms of skill. Competitors have access to the same tools and resources that a professional would have. While these are students, they are an elite group; these students are the next generation of security experts. Additionally, there is some evidence that students and professional software developers behave similarly when implementing functionality with security implications. For example, Naiakshina et al. [37] replicated a previous study [38] conducted with **computer science (CS)** students with professional software developers. In both studies, participants were tasked with implementing registration functionality for a social networking platform. Naiakshina et al. found that both CS students and professional software developers behaved similarly when told explicitly to consider secure password storage

and when using the same Java framework, concluding that "security developer studies conducted with CS students can offer valuable insights [37]." While this is only one example, and other factors could be at play, other studies have also found that computing students and professional developers have similar security behaviors [1, 2, 15, 39].

6.1.5 Penetration Testers vs. Malicious Actors. We recognize that the actions of penetration testers and malicious attackers may be different and that insights based on penetration testers may not generalize to malicious attackers. However, the CPTC data provides a unique opportunity to study penetration testers in a controlled environment, which is worth studying, given the incomplete information for vulnerability discoverability/exploitability in the real world (see Section 3.1). Additionally, there is some evidence that penetration testers and hackers follow a very similar process for vulnerability discovery [50].

6.2 Incomplete Timelines

Of the 98 total vulnerability discoveries, 53 timelines could not be completed, either due to missing evidence in Splunk or a lack of information in team reports. For example, Splunk logs show that a Postgres connection was active or a Metasploit console was running, but Splunk did not log individual Postgres/Metasploit shell commands, which are crucial to corroborating certain attacks. We have contacted the CPTC development team and asked them to implement logging for Metasploit console and Postgres shell commands; Metasploit console logging was implemented in 2020. There were also 28 instances of teams describing commands or web page access that could not be corroborated using Splunk logs. We only use completed timelines in our metric-based analyses (**RQ1** and **RQ2**).

6.3 Mean Time to Vulnerability Discovery

The MTVD metric is based on the time from the beginning of the penetration testing competition to the time of discovery for a vulnerability. We recognize that this measurement, being in minutes (or hours), does not reflect the time to discovery of vulnerabilities in the real world, as they often exist for days or years before being discovered. However, our purpose is to quantify *relative* vulnerability discoverability, and the MTVD metric does that. While the MTVD measurements for this dataset are in minutes, the metric can be calculated at different time scales (hours, days, or years) for other datasets to achieve the same purpose.

7 SUMMARY & FUTURE WORK

In this work, we curated 98 timelines of vulnerability discoverability for 37 unique vulnerabilities discovered by 10 teams in the 2019 **Collegiate Penetration Testing Competition (CPTC)**. We derived three discoverability and exploitability metrics from the timelines and compared those metrics across categories of vulnerabilities with related weaknesses. We also examined the impact of mitigation strategies (suggested by CWE) and attacker behavior using MITRE ATT&CK™ tactics and techniques. The contributions of our work are outlined below:

- a publicly available dataset [23] of vulnerability discovery timelines from the 2019 CPTC nationals event;
- a methodology for gathering evidence of vulnerability discovery using Splunk logs and vulnerability reports;
- a methodology for mapping reported vulnerabilities to their underlying weakness in CWE;
- an improved methodology for mapping events in an attack timeline to MITRE ATT&CK™ tactics and techniques;
- new insights into penetration tester behavior;

- new metrics for measuring and comparing vulnerability discoverability; and
- an in-depth discussion of common concerns with competition data and how CPTC addresses these concerns.

We found that vulnerabilities related to improper resource control (e.g., insecure defaults, sensitive data leakage) were among the most dangerous according to CWE rankings, while also being discovered faster/more often and exploited more easily than vulnerabilities related to improper access control (e.g., improper authentication, lack of authorization). Our findings provide empirical support for some of CWE's top most dangerous weakness rankings. Software engineers can use these findings to prioritize their vulnerability assessment processes. Examining suggested mitigation strategies from CWE revealed that no single mitigation technique is enough to sufficiently secure a software system. We recommend that software engineers examine different types of vulnerabilities in the context of their software systems and implement mitigations for multiple vulnerabilities to adequately secure their software. Finally, we found that there is a clear process employed by penetration testers: (1) discover as many vulnerabilities as possible while exploring the infrastructure and collecting information to aid exploits and (2) move throughout the infrastructure while preparing to attack. Since penetration testers follow a similar vulnerability discovery process as malicious attackers [50], we recommend that software engineers prioritize defenses against discovery, collection, and lateral movement tactics. We also experimented with automating MITRE ATT&CK™ mapping.

These are valuable insights that can inform vulnerability prioritization and mitigation techniques for security practitioners, software engineers included. We intend to repeat this analysis for future CPTC competitions, which will allow us to improve the collective understanding of vulnerability discoverability and exploitability. To facilitate future analysis, we discuss the following directions for future work.

7.1 Automation

As previously discussed, we intend to use machine learning techniques to automatically map timeline events to MITRE ATT&CK™ tactics and techniques. Initial results (discussed in Section 4.5) from classification suggest the need for more training data. After we manually map 2020 timeline events to tactics and techniques, we will re-evaluate our models trained on 2018 and 2019 data and determine our next steps. We also intend to explore different models for classification, including convolutional neural networks that use the previous events in a timeline to classify tactics and techniques. Mapping timeline events to MITRE D3FEND™[13] or smaller frameworks, such as the Cyber Attack Kill Chain [17] or the Action-Intent Framework [32] may also yield meaningful insights while reducing the complexity of automated mapping.

We also intend to use machine learning and natural language processing techniques to automate CWE mapping. We have started experimenting with different classification models, but there is still much work to be done. Our previous work [22] has shown promise using word embeddings to automatically cluster related vulnerabilities, which may be adapted to help automate CWE mapping. We may also be able to use other natural language processing techniques, such as latent Dirichlet allocation. CWE classification models could suggest potential CWE candidates for vulnerabilities, narrowing the search space of CWEs for manual mapping and reducing classification time and effort.

The evidence-gathering process (outlined in Section 4.2) currently requires a large amount of manual human time and effort. We intend to automate this process as much as possible. There is

[13] https://d3fend.mitre.org/.

potential to apply machine learning models trained on existing manually curated timelines with the goal of automatically identifying potential timelines in future competitions. To facilitate this, we recorded the Splunk query used to narrow down the search space for every log event within a timeline. These potential timelines would still need to be manually analyzed to determine if they match attacks described in team reports, but that may take less time than manual curation of timelines. Additionally, this automated approach could potentially identify failed attack paths, which could benefit other areas of security research.

7.2 Analysis & Understanding of Trends

Since CPTC is an annual competition, repeating our analysis (and adding new analyses) each year affords us an opportunity to study vulnerability discoverability and attacker behavior across the dimension of time. We intend to study the insights from CPTC data each year with the goals of (1) identifying trends in vulnerability discoverability and (2) observing changes in attacker behavior. The insights gained from annual analysis of CPTC data will aid software engineers and other security practitioners in vulnerability assessment.

7.3 Curated Insights Portal

We hope to build and maintain a publicly accessible website that presents the insights gained from CPTC analysis in the form of graphs, visual timelines, and other curated metadata (e.g., mistakes made, lessons learned, CVSS scores, mitigation strategies), similar in spirit to the **Vulnerability History Project (VHP)**.[14] By presenting our insights with actionable recommendations, we can aid software engineers and other security practitioners in vulnerability assessment and learning from history.

ACKNOWLEDGMENTS

The authors wish to thank the CPTC development team for their commitment to releasing high-quality research data. The authors also wish to thank Research Computing [42] at the Rochester Institute of Technology for providing computational resources and support that have contributed to the research results reported in this work.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 289–305.

[2] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. 2017. Security developer studies with github users: Exploring a convenience sample. In *13th Symposium on Usable Privacy and Security (SOUPS'17)*. 81–95.

[3] Raymond Albert, George Markowsky, and Joanne Wallingford. 2010. High school cyber defense competitions: Lessons from the trenches. In *International Conference on Security and Management (SAM'10)*. 280–285.

[4] Nicolas Aussel, Yohan Petetin, and Sophie Chabridon. 2018. Improving performances of log mining for anomaly prediction through NLP-based log parsing. In *IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 237–243.

[5] Yogeshwar Rao Bachupally, Xiaohong Yuan, and Kaushik Roy. 2016. Network security analysis using big data technology. In *SoutheastCon*. IEEE, 1–4.

[6] Masooda Bashir, April Lambert, Jian Ming Colin Wee, and Boyi Guo. 2015. An examination of the vocational and psychological characteristics of cybersecurity competition participants. In *USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE'15)*.

[7] Mark Boger, Tianyuan Liu, Jacqueline Ratliff, William Nick, Xiaohong Yuan, and Albert Esterline. 2016. Network traffic classification for security analysis. In *SoutheastCon*. IEEE, 1–2.

[14] http://vulnerabilityhistory.org/.

[8] Harold Booth, Doug Rike, and Gregory Witte. 2013. The National Vulnerability Database (NVD): Overview. Retrieved from https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172.
[9] Steve Caldeira. 2020. CTF vs Real Penetration Testing. Retrieved from https://www.triaxiomsecurity.com/ctf-vs-real-penetration-testing/.
[10] Peter Chapman, Jonathan Burket, and David Brumley. 2014. PicoCTF: A game-based computer security competition for high school students. In *USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE'14)*.
[11] Nicholas Childers, Bryce Boe, Lorenzo Cavallaro, Ludovico Cavedon, Marco Cova, Manuel Egele, and Giovanni Vigna. 2010. Organizing large scale hacking competitions. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 132–152.
[12] CPTC. 2021. Global Collegiate Penetration Testing Competition. Retrieved from https://globalcptc.org.
[13] Keith Harrison and Gregory White. 2010. An empirical study on the effectiveness of common security measures. In *43rd Hawaii International Conference on System Sciences*. IEEE, 1–7.
[14] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami. 2016. An automatic method for CVSS score prediction using vulnerabilities description. *J. Intell. Fuzzy Syst.* 30, 1 (2016), 89–96.
[15] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I have no idea what I'm doing"—On the usability of deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security'17)*. 1339–1356.
[16] Stela Kucek and Maria Leitner. 2020. An empirical survey of functions and configurations of open-source capture the flag (CTF) environments. *J. Netw. Comput. Applic.* 151 (2020), 102470.
[17] Lockheed Martin. 2018. The Cyber Kill Chain. Retrieved from https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.
[18] Bob Martin and Steve Christey Coley. 2014. Common Weakness Scoring System (CWSS). Retrieved from https://cwe.mitre.org/cwss/cwss_v1.0.1.html.
[19] Lucas McDaniel, Erik Talvi, and Brian Hay. 2016. Capture the flag as cyber security introduction. In *49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 5479–5486.
[20] Miles A. McQueen, Trevor A. McQueen, Wayne F. Boyer, and May R. Chaffin. 2009. Empirical estimates and observations of 0day vulnerabilities. In *42nd Hawaii International Conference on System Sciences*. IEEE, 1–12.
[21] Peter Mell, Karen Scarfone, and Sasha Romanosky. 2006. Common vulnerability scoring system. *IEEE Secur. Priv.* 4, 6 (2006), 85–89.
[22] Benjamin S. Meyers and Andrew Meneely. 2021. An automated post-mortem analysis of vulnerability relationships using natural language word embeddings. *Procedia Comput. Sci.* 184 (2021), 953–958.
[23] Benjamin S. Meyers and Andrew Meneely. 2021. *Vulnerability Discoverability Timelines from the 2019 Collegiate Penetration Testing Competition*. DOI: https://doi.org/10.5281/zenodo.5781239
[24] Jose David Mireles, Jin-Hee Cho, and Shouhuai Xu. 2016. Extracting attack narratives from traffic datasets. In *International Conference on Cyber Conflict (CyCon US)*. IEEE, 1–6.
[25] Jelena Mirkovic and Peter A. H. Peterson. 2014. Class capture-the-flag exercises. In *USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE'14)*.
[26] MITRE. 2019. 2019 CWE Top 25 Most Dangerous Software Weaknesses. Retrieved from https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.
[27] MITRE. 2019. Common attack Pattern Enumeration and Classification (CAPEC). Retrieved from https://capec.mitre.org/about/index.html.
[28] MITRE. 2019. Common Vulnerabilities and Exposures (CVE). Retrieved from https://cve.mitre.org/about/index.html.
[29] MITRE. 2020. 2020 CWE Top 25 Most Dangerous Software Weaknesses. Retrieved from https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html.
[30] MITRE. 2020. Common Weakness Enumeration (CWE). Retrieved from https://cwe.mitre.org/about/index.html.
[31] MITRE. 2021. CVE to CWE Mapping Guidance. Retrieved from https://cwe.mitre.org/documents/cwe_usage/guidance.html.
[32] Stephen Moskal and Shanchieh Jay Yang. 2020. Cyberattack action-intent-framework for mapping intrusion observables. *CoRR* (2020). https://arxiv.org/abs/2002.07838.
[33] Stephen Moskal, Shanchieh Jay Yang, and Michael E. Kuhl. 2018. Extracting and evaluating similar and unique cyber attack strategies from intrusion alerts. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 49–54.
[34] Steven M. Muegge and S. M. Monzur Murshed. 2018. Time to discover and fix software vulnerabilities in open source software projects: Notes on measurement and data availability. In *Portland International Conference on Management of Engineering and Technology (PICMET)*. IEEE, 1–10.
[35] Nuthan Munaiah, Justin Pelletier, Shau-Hsuan Su, S. Jay Yang, and Andrew Meneely. 2019. A cybersecurity dataset derived from the national collegiate penetration testing competition. In *HICSS Symposium on Cybersecurity Big Data Analytics*.

[36] Nuthan Munaiah, Akond Rahman, Justin Pelletier, Laurie Williams, and Andrew Meneely. 2019. Characterizing attacker behavior in a cybersecurity penetration testing competition. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–6.

[37] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. On conducting security developer studies with CS students: Examining a password-storage study with cs students, freelancers, and company developers. In *CHI Conference on Human Factors in Computing Systems*. 1–13.

[38] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel Von Zezschwitz, and Matthew Smith. 2019. "If you want, I can store the encrypted password" a password-storage field study with freelance developers. In *CHI Conference on Human Factors in Computing Systems*. 1–12.

[39] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A stitch in time: Supporting Android developers in writing secure code. In *ACM SIGSAC Conference on Computer and Communications Security*. 1065–1077.

[40] Fabian Pedregosa, Gael Varoquax, and Alexandre Gramfort. 2021. sklearn.metrics.cohen_kappa_score. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html.

[41] Roger S. Pressman. 2005. *Software Engineering: A Practitioner's Approach* (6th ed.). Palgrave Macmillan.

[42] Rochester Institute of Technology. 2020. Research Computing Services. DOI: https://doi.org/10.34788/0S3G-QD15

[43] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manag.* 24, 5 (1988), 513–523. DOI: https://doi.org/10.1016/0306-4573(88)90021-0

[44] Teodor Sommestad and Jonas Hallberg. 2012. Cyber security exercises and competitions as a platform for cyber security experiments. In *Nordic Conference on Secure IT Systems*. Springer, 47–60.

[45] Paul Sroufe, Steve Tate, Ram Dantu, and Ebru Celikel Cankaya. 2010. Experiences during a collegiate cyber defense competition. *J. Appl. Secur. Res.* 5, 3 (2010), 382–396.

[46] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. 2018. *MITRE ATT&CK™: Design and philosophy.* Technical Report. MITRE Corporation.

[47] Valdemar Švábenskỳ, Pavel Čeleda, Jan Vykopal, and Silvia Brišáková. 2021. Cybersecurity knowledge and skills taught in capture the flag challenges. *Comput. Secur.* 102 (2021), 102154.

[48] U.S. Department of Homeland Security. 2021. Science and Technology: Cybersecurity Competitions. Retrieved from https://www.dhs.gov/science-and-technology/cybersecurity-competitions.

[49] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupe, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. 2014. Ten years of iCTF: The good, the bad, and the ugly. In *USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE'14)*. USENIX Association.

[50] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. 2018. Hackers vs. testers: A comparison of software vulnerability discovery processes. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 374–391.

[51] Ruyi Wang, Ling Gao, Qian Sun, and Deheng Sun. 2011. An improved CVSS-based vulnerability scoring mechanism. In *3rd International Conference on Multimedia Information Networking and Security*. IEEE, 352–355.

[52] Molra J. West-Brown, Don Stikvoort, Klaus-Peter Kossakowski, Georgia Killcrece, and Robin Ruefle. 2003. *Handbook for Computer Security Incident Response Teams (CSIRTs).* Technical Report. Carnegie-Mellon University, Software Engineering Institute, PA.

[53] Carl Wilhjelm, Taslima Kotadiya, and Awad A. Younis. 2020. Empirical characterization of the likelihood of vulnerability discovery. *Int. J. Perform. Eng.* 16, 7 (2020), 1008.

[54] Jeff Williams. 2020. OWASP Risk Rating Methodology. Retrieved from https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.

Received August 2021; revised December 2021; accepted January 2022