SwarmFuzz: Discovering GPS Spoofing Attacks in Drone Swarms

Yingao (Elaine) Yao
The University of British Columbia
elainey@ece.ubc.ca

Pritam Dash
The University of British Columbia
pdash@ece.ubc.ca

Karthik Pattabiraman
The University of British Columbia
karthikp@ece.ubc.ca

***Abstract*—Swarm robotics, particularly drone swarms, are used in various safety-critical tasks. While a lot of attention has been given to improving swarm control algorithms for improved intelligence, the security implications of various design choices in swarm control algorithms have not been studied. We highlight how an attacker can exploit the vulnerabilities in swarm control algorithms to disrupt drone swarms. Specifically, we show that the attacker can target a swarm member (target drone) through GPS spoofing attacks, and indirectly cause *other* swarm members (victim drones) to veer from their course, resulting in a collision with an obstacle. We call these *Swarm Propagation Vulnerabilities*.**

In this paper, we introduce SwarmFuzz, a fuzzing framework to capture the attacker's ability, and efficiently find such vulnerabilities in swarm control algorithms. SwarmFuzz uses a combination of graph theory and gradient-guided optimization to find the potential attack parameters. Our evaluation on a popular swarm control algorithm shows that SwarmFuzz achieves an average success rate of 48.8% in finding vulnerabilities, and compared to random fuzzing, has a 10x higher success rate, and 3x lower runtime. We also find that swarms of a larger size are more vulnerable to this attack type, for a given spoofing distance.

***Index Terms*—Drone Swarm, GPS Spoofing, Resilience**

I. Introduction

Drone swarms are a type of distributed cyber-physical system inspired by swarm intelligence [1], consisting of multiple drones that can communicate with each other. They carry out large-scale missions that cannot be performed by a single drone, e.g., logistics, surveillance, search and rescue [2]–[4].

Unfortunately, drone swarms are vulnerable to threats such as logic flaws [5] in swarm control algorithms and masquerade attacks [6]. However, attacks exploiting such threats incur high costs (e.g., introducing an external drone [5], sending spurious messages to the swarm [6]), and can be thwarted using techniques such as intruder detection [7] and drone authentication [8]. In contrast, physical attacks that feed the drone with erroneous sensor measurements via physical channels (e.g., GPS, accelerator, gyroscope) require relatively little effort, and can lead to drones crashing [9]–[13].

GPS attacks [9], [10], which send malicious GPS signals to victim drones, are examples of physical attacks. GPS attacks have been shown in various systems such as self-driving cars, drones, and trucks [14]–[16]. For example, during a drone show in Hong Kong, 46 drones crashed due to GPS jamming [17]. GPS spoofing, in which the GPS value is falsified, can be more debilitating than GPS jamming as it uses more subtle signals, and thus is more difficult to detect [18].

In this paper, we first demonstrate that GPS spoofing attacks can disrupt drone swarms by exploiting vulnerabilities in swarm control algorithms - we call these Swarm Propagation Vulnerabilities (SPVs). Specifically, the attacker launches GPS spoofing in a swarm member (target drone), causing the target drone to deviate from its correct trajectory, but avoiding collision with other swarm members (for attack stealthiness). The deviation changes the inter-distance between the target drone and other members of the swarm (victim drones). This, in turn, leads to incorrect control commands generated by the swarm control algorithm. These incorrect commands cause the victim drones to veer off course, resulting in collisions.

Note that the target drone in the attack is *not* the one involved in the collision, and hence it is difficult to identify it as the "bad apple". Also, current defenses [19], [20] for GPS spoofing attacks in a single drone often ignore small GPS spoofing deviations (e.g., $0 - 10m$). Even if such defenses are deployed in all swarm members, they will fail to detect this type of attack. This allows the attacker to attack the other swarm members without being detected, causing collisions, thereby reducing the mission efficiency and leading to potentially catastrophic outcomes such as crashes.

The main cause of SPVs are the design choices of swarm control algorithms. To generate the control commands (e.g., heading directions), the swarm control algorithm has to balance conflicting goals with different priorities. For example, for a swarm control algorithm to maintain the formation in the swarm, it may give higher weights to goals involving interactions among swarm members than those that avoid the obstacle. While most swarm control algorithms are tuned to balance these goals under normal conditions and avoid collisions, they are unable to do so when the attacker can manipulate swarm members' perceived locations, such as through GPS spoofing attacks, at strategic times. This can result in collisions between the swarm members and obstacles.

To help defenders evaluate the resilience of the drone swarm mission against SPVs, we design a fuzzing technique that can capture the attackers' capabilities, map them to the drone swarm, and discover SPVs before running missions. Fuzzing is a testing technique that has been widely used to find vulnerabilities in real-world applications [5], [21]–[25]. However, directly applying previous fuzzing approaches for finding SPVs in drone swarms is challenging for the following two reasons. (1) The input spaces are large for long missions

2158-3927/23/$31.00 ©2023 IEEE
DOI 10.1109/DSN58367.2023.00043

and large drone swarms, making it difficult to efficiently find the target-victim drone pairs that are vulnerable to SPVs. (2) They typically target failures due to the exploitation of memory vulnerabilities, while we target failures that cause victim drones to crash into obstacles via GPS spoofing attacks.

We make the following two observations about the attacker who wants to successfully exploit SPVs in drone swarms. (1) To maximize the attack impact with minimal effort, the attacker should find the most influential drone as the target drone and the drone closest to the obstacle as the victim drone. (2) To efficiently cause collisions, the attacker should find GPS spoofing parameters that can minimize the distance between the victim drone and the obstacle. We find that this is a convex optimization problem, which can be solved efficiently.

Motivated by the above observations, we propose SwarmFuzz, a novel fuzzing technique to efficiently find SPVs in drone swarms. SwarmFuzz has two innovations. First, we develop an abstraction called the *Swarm Vulnerability Graph (SVG)*, and utilize graph centrality analysis [26] to measure the malicious influence of each target-drone pair. SwarmFuzz leverages graph centrality to decide the order of the target-victim drone pairs for fuzzing, so that the most influential drone pairs are prioritized to discover the vulnerabilities more efficiently. Second, SwarmFuzz employs gradient descent, which is known to be efficient for convex optimization problems, to search for the other GPS spoofing parameters that will cause collisions.

We apply SwarmFuzz[1] to a popular swarm control algorithm [27] in the SwarmLab [28] simulator. We evaluate SwarmFuzz in different swarm configurations by varying the swarm size and the GPS spoofing distance. SwarmFuzz helps swarm designers to evaluate the resilience of the swarm mission beforehand. If the swarm mission is found to be vulnerable to SPVs, the designers can take actions (e.g., tuning the parameters in the control algorithm) to make it secure.

To the best of our knowledge, we are the first to demonstrate the presence of SPVs in drone swarm control algorithms, and propose a technique to systematically and efficiently find such vulnerabilities. We make the following contributions.

- Demonstrate attacks that exploit SPVs to indirectly cause disruptions (e.g., collisions) in drone swarms.
- Propose the SVG abstraction, and utilize centrality analysis to find drone pairs that are likely to result in collisions.
- Design SwarmFuzz, a fuzzer that uses SVG and gradient-guided optimization to efficiently evaluate the resilience of swarm missions by finding SPVs that cause collisions.
- Evaluate SwarmFuzz on a popular swarm control algorithm. We find that (1) SwarmFuzz can achieve an average success rate of 48.8% in finding SPVs across swarm configurations; (2) SwarmFuzz has a higher success rate for missions with a larger swarm size or with longer GPS spoofing distance. Further, for missions in which the swarm passes close to the obstacle, even a small spoofing distance causes collisions. Finally, both the SVG and gradient-guided optimization are necessary for SwarmFuzz.

[1]Available at: https://github.com/DependableSystemsLab/SwarmFuzz

II. Background and Threat Model

Drone swarms. Swarm control algorithms can be either distributed or centralized [1]. Centralized algorithms are less resilient as they have a single point of failure. Therefore, we focus on distributed swarm control algorithms in this paper.

Distributed drone swarms follow four steps in a periodic loop (Fig. 1): (1) each swarm member reads sensor data (e.g., GPS, IMU) to get its current physical states (e.g., location, velocity); (2) swarm members exchange physical states among themselves by sending messages through the communication system; (3) the swarm control algorithm running in each drone uses the physical states of other swarm members to compute the state difference (e.g., inter-distance, relative velocity); (4) each drone derives the control commands by itself independent of other drones based on the state difference and goals.

Specifically, to coordinate the swarm members, the swarm control algorithm has to adhere to the following three high-level goals: (1) mission-driven, to ensure the drone swarm is moving towards the destination; (2) collision-free, to ensure no collisions by maintaining minimum distance among drones/obstacles; (3) cohesive formation, to preserve the swarm formation by avoiding long distance among drones.

As a result, the final control command generated in a drone mainly consists of three sub-commands, each for a specific goal. One key component in the swarm control algorithm is to balance these potentially conflicting goals through careful tuning. For example, when an obstacle is in the path of a drone to the destination, the swarm control algorithm needs to assign higher weights to goals (1) and (2) to ensure that the drone moves towards the destination while avoiding the obstacle.

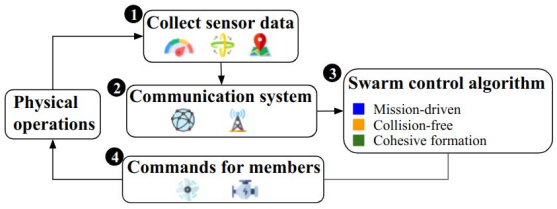


Fig. 1: Workflow of distributed drone swarm systems.

GPS Spoofing Attacks. The Global Position System (GPS) is widely used in Robotic Vehicles (RVs) for outdoor localization. The GPS receiver calculates the position based on the signal received from the GPS satellites. Civilian GPS systems lack signal authentication and encryption, and hence are vulnerable to GPS spoofing [10]. RV missions like delivery, search, and rescue [2]–[4] are vulnerable to GPS spoofing as they depend on the location information. We focus on missions in which the swarm goes from one point to another.

In a GPS spoofing attack, the attacker transmits fabricated GPS signals with stronger power than the original GPS signal, so that the victim GPS receiver locks on to the attacker's signal. The attacker controls the victim's perceived position by manipulating the GPS messages. GPS spoofing has been shown in laboratory [9] and real life settings [14], [16], [17].

Prior work has demonstrated how to infer a drone's information to perform GPS spoofing attacks on it [9], [10], [29]–[31]. While many defenses against GPS attacks have been proposed for a single drone [19], [20], [32], most of them ignore small GPS spoofing distances (e.g, $0 - 10m$) as those are indistinguishable from the standard GPS offset [33], This is often sufficient to ensure the safety of a single drone. However, in a drone swarm, since each swarm member acts based on its neighbors' physical states, even a slight location deviation in one drone's location may influence the other members, and cause disruptions (e.g., collisions) in the drone swarm.

Threat Model. We assume the attacker can perform GPS spoofing attacks in only one member in the drone swarm. This requires much less attack effort than spoofing the GPS for multiple drones in the swarm, and also makes the attack stealthier. The attacker's goal is to cause a swarm member to collide with the on-path obstacle. We do not consider collisions between the swarm members themselves. We assume the attacker knows the obstacle's GPS coordinates. She can achieve this by either 1) placing the obstacle by herself, or 2) choosing an existing obstacle in the drone swarm's trajectory, and looking up its GPS location. We also assume the attacker knows the swarm control algorithm being used, but she does not know the high-level goals explained in Section III as SwarmFuzz automatically finds the vulnerability. Further, the attacker does not have the capability to intercept or modify the messages exchanged among the swarm members as the messages may be encrypted [34]. Finally, we assume that the attacker cannot introduce any external drones into the swarm.

III. Motivating example and Challenges

We first present an example drone swarm running a highly-cited swarm control algorithm - Vicsek algorithm [27], to show how SPVs can be exploited (details in Section V-A). We then present the design challenges in systematically finding SPVs.

A. Motivating example

Swarm Setup. We emulate a 5-drone swarm in the Swarmlab [28] simulator for a delivery mission, as shown in Fig. 2-(a). It aims to reach a pre-defined destination (i.e., the flag) while avoiding the on-path obstacle (i.e., the purple triangle). For example, in Fig. 2-(a), the obstacle blocks drone 5's way to the destination. Therefore, the swarm control algorithm attempts to make drone 5 avoid the obstacle from the right.

Goals. As mentioned in Section II, the swarm control algorithm mainly follows three goals to generate the appropriate control commands, i.e., (1) mission-driven, (2) collision-avoidance, and (3) cohesive formation. The algorithm generates different sub-velocities for each swarm member, based on the above goals. Fig. 2-(b) shows the mapping between each goal and the corresponding sub-velocity generated.

For goal (1), each drone has a sub-velocity for moving towards the destination (i.e., blue arrows). For goal (2), with the short distance between drone 1 and drone 2, repulsive sub-velocities (i.e., orange arrows) are generated to avoid collisions between them. For goal (3), with the relatively long distance between drone 1 and drone 5, attractive sub-velocities (i.e., green arrows) are generated between the drones to maintain the formation. These sub-velocities may conflict with each other.

Exploiting SPVs. The attacker first launches GPS spoofing and causes the target drone to perceive a wrong location, which it communicates to the swarm. This deviation changes the inter-distance between the target drone and the victim drone, causing the swarm control algorithm to generate incorrect control commands. These incorrect commands cause the victim to deviate from the course, potentially resulting in collisions.

For example, in Fig. 2-(c), drone 4 (target drone) is under the GPS spoofing attack and thus deviates to the right. This deviation increases the inter-distance between drone 4 and drone 5 (victim drone), making it difficult for the drone swarm to maintain the formation. Hence, according to goal (3) in Section II, attractive sub-velocities (i.e., green arrows) between drone 4 and drone 5 are generated. Note that originally, drone 5 avoids the obstacle from the right. However, with this new sub-velocity, the overall velocity (i.e., red arrow) for drone 5 points towards the obstacle, thereby leading to a collision.

According to goal (2) in Section II, drone 5 also has a sub-velocity for avoiding the obstacle. However, since the sub-velocities generated by other goals are bigger than the sub-velocity to avoid the obstacle, drone 5 flies towards the obstacle and collides with it, thereby violating the swarm's safety constraints. Thus, the above attack was successful.

We found that the above attack succeeds in different swarm missions with different GPS spoofing distances (i.e., $5m$ and $10m$), indicating that this particular swarm configuration is highly susceptible to SPVs. However, it is tedious to perform this evaluation manually for different swarm configurations.

B. Design Challenges

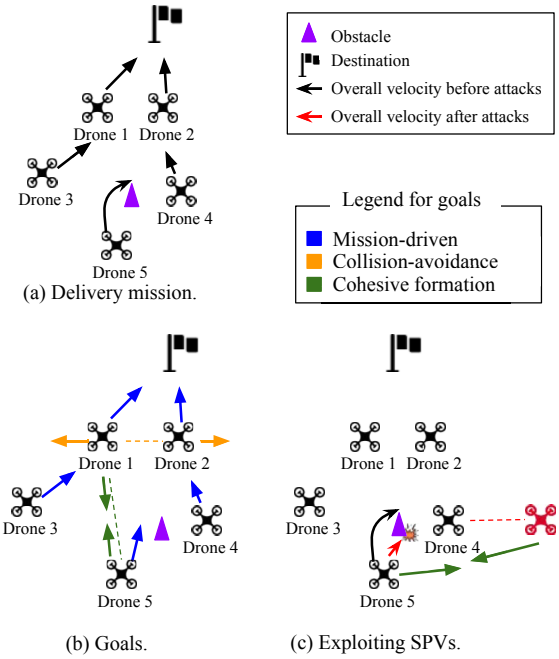
For defenders to evaluate the resilience of swarm configurations, it is important to develop an automated approach to systematically and efficiently find SPVs in drone swarms. This leads to two unique challenges, as follows.

C1: Finding Target-Victim Pairs. To exploit the SPVs and launch a stealthy attack, appropriate target-victim drone pairs need to be chosen strategically. This is because in a large drone swarm, the number of possible combinations of target-victim drone pairs is huge, which leads to a large input space.

C2: Choosing Spoofing Parameters. Given the spoofing deviation, performing GPS spoofing attacks involves two key parameters, i.e., spoofing direction and spoofing time. Spoofing direction refers to the direction the target drone deviates under GPS spoofing (e.g., right, left), while spoofing time refers to the duration of time that the GPS spoofing attack lasts. The wrong choice of the spoofing parameters (i.e., direction and time) in the target drone will cause the victim drone to avoid the collision, thereby defeating the point of the attack.

IV. Methodology

Fig. 3 presents the overview of SwarmFuzz. SwarmFuzz only takes as inputs (1) the swarm control algorithm, (2) the mission parameters (including the swarm size and the location of the



(a) Delivery mission. (b) Goals. (c) Exploiting SPVs.

Fig. 2: Motivating example for SPVs in drone swarms.

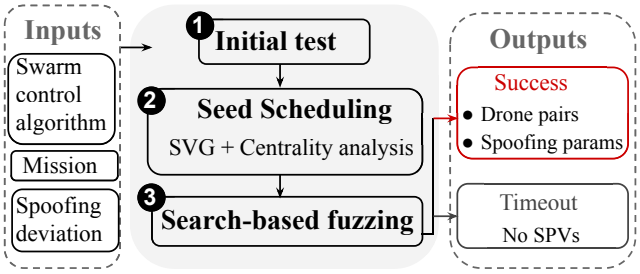


Fig. 3: Overview of SwarmFuzz

obstacle), and (3) the GPS spoofing deviation. It performs the following three steps. (1) It runs an initial test without any attack. If this test mission is successful (i.e., no collisions), it records mission information to construct the *Swarm Vulnerability Graph (SVG)*. (2) It performs centrality analysis on the SVG to analyze the influence of each drone with a certain spoofing direction, and uses this to decide the order in which target-victim drone pairs are selected for fuzzing (**C1**). (3) For a certain target-victim drone pair with the spoofing direction, it searches for the spoofing parameters (i.e., start time and duration) that minimize the distance between the victim drone and the obstacle with gradient-guided optimization (**C2**). It repeats step (3) until a collision occurs or it reaches a predefined number of search iterations. If a collision occurs, SwarmFuzz has found a successful SPV, and it outputs the target-victim drone pair(s), and the spoofing parameters for the collision. Otherwise, SwarmFuzz reports that no SPVs were found in the mission, and the mission is resilient to SPVs.

A. Test Definition and Initial Test Creation

A test-run is defined as a set of tuples $< T-V, t_s, \Delta t, \theta >$, where $T-V$ represents the target-victim drone pair, t_s represents the spoofing start time, Δt represents the spoofing duration, and θ represents the spoofing direction. We aim for as few spoofing parameters as possible to minimize the attacker's effort, and thus focus on *horizontal constant spoofing*, i.e., always setting the GPS spoofing distance to a constant d (provided as the input) during time Δt, and only perform spoofing horizontally. Thus, the value for θ is +1/-1, representing right and left respectively. The value range for T and V could be any drone in the swarm as long as they are different from each other. t_s and Δt range from 0 to the entire mission time.

We create the initial test case by running the swarm mission without any spoofing attack. During the test, we collect the following information: (1) each drone i's location (x_i, y_i, z_i) at each timestamp t_j, i.e., $< x_i, y_i, z_i, t_j >$; (2) the minimum distance between each drone i and the obstacle throughout the mission ${D_{ob}}^i$; and (3) the mission duration $t_{mission}$.

B. Seed scheduling

To tackle **C1**, we measure the malicious influence of each target-victim drone pair, and choose drone pairs for fuzzing in decreasing order of the influence. We first develop an abstraction called the *Swarm Vulnerability Graph (SVG)*. We then utilize graph centrality analysis to measure the influence of a swarm member in the SVG. Finally, we choose the most influential member as the target drone and the drone closest to the obstacle as the victim drone, to order the seeds for fuzzing.

Seedpool. The seedpool consists of a set of seeds $< T-V, \theta >$. We observe that the probability of causing collisions for each seed depends on two factors: (1) the influence of the drone pair $T-V$; and (2) the Victim drone's closest Distance to the Obstacle (VDO) (in the absence of attacks). For factor (1), the malicious impact of the target drone is a function of its influence over the whole drone swarm, especially the victim drone. Thus, choosing the most influential drone as the target is more likely to cause collisions. For factor (2), a drone closer to the obstacle (i.e. low VDO) is more promising as a victim drone, as it requires lower effort to crash into the obstacle.

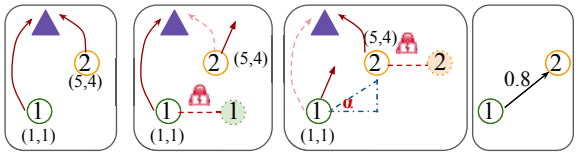
SVG Definition. The malicious influence of each drone can be intuitively measured by the swarm member's reactions to the spoofing deviation of the target drone. To quantify this influence, we propose the *Swarm Vulnerability Graph (SVG)*. The idea behind the SVG is to utilize the centrality analysis in graph theory [35], which measures a node's influence in the graph and is used to identify the graph's most influential node.

We define $SVG = (N, E, W)$, where N is the set of nodes representing the swarm members, E is the set of edges capturing the connections between drones, and W is the set of weights measuring the local influence on each edge e. SVG is a directed graph, and the direction of the edge models the source of the influence. For example, if node n_i is influenced by node n_j, a directed edge e_{ij} is created from n_i to n_j.

We observe that when the drones are closest to each other, the influence among drone members are the strongest. Therefore, we first calculate the average inter-distance among drone members during the test-run, with the recorded information $< x_i, y_i, z_i, t_j >$ in Section IV-A. We then choose the time t_{clo} with the shortest average inter-distance, and use the swarm's location $< x_i, y_i, z_i, t_{clo} >$ at t_{clo} to construct the SVG.

To construct the SVG, we first analyze each drone i's potential velocity direction when the other drone j is under GPS spoofing at t_{col}, based on the goals in Section II. We then create an edge e_{ij} if drone i is under the malicious influence (i.e., closer to the obstacle) of drone j. Finally, we compute the weight w_{ij} based on the inter-distance between drone i and drone j, to capture the local influence between these drones.

Creating Edges in the SVG. We add an edge e_{ij} if and only if node j has a malicious influence on node i. We infer the malicious influence based on the distance change between the drone and the obstacle. If the spoofing deviation in drone j causes the *distance to decrease* between drone i and the obstacle, according to goals in Section II, then drone j has a malicious influence on drone i, and we add the edge e_{ij}.



(a) No attack (b) Spoof drone 1 (c) Spoofing drone 2 (d) SVG

Fig. 4: Constructing SVG with appropriate spoofing directions.

Fig. 4 shows how to create the edge in the SVG in a two-drone scenario, for the right spoofing direction. In the no attack scenario (Fig. 4-(a)), an obstacle blocks drone 1 and drone 2's way to the destination. Therefore, the swarm control algorithm generates control commands to make drone 1 avoid the obstacle from the left and drone 2 from the right. To exploit SPVs, either drone 1 or drone 2 could be chosen as the target drone to be spoofed. In Fig. 4-(b), drone 1 is chosen as the target drone and thus deviates to the right. This deviation decreases the inter-distance between drone 1 and drone 2. Based on goal (2) in Section II, repulsive velocities are generated between the two drones. With this new repulsive velocity, drone 2 moves *further* away from the obstacle (i.e., solid red arrow), thereby leading to an increase in its distance with the obstacle. This means drone 1 has no malicious influence on drone 2, and hence the edge e_{21} is not created.

In Fig. 4-(c), however, drone 2 is chosen as the target drone and deviates to the right. Similarly, according to goal (3), we can infer that drone 1 moves *closer* to the obstacle (i.e., solid red arrow), thereby leading to a decrease in its distance to obstacle. This means drone 2 has a malicious influence on drone 1, and hence the edge e_{12} is created in Fig. 4-(d). The edges for the left spoofing can be constructed similarly.

Computing Weights in the SVG. If e_{ij} exists, we use w_{ij} to represent the local influence that drone j has on drone i. Higher the influence, higher is the value of w_{ij}. When drone j is far away from drone i, the deviation in drone j has less influence on drone i, thus w_{ij} is relatively small. Specifically, we use the cosine value of angle α (Fig. 4-(c)) in the right triangle involving drones i and j to calculate w_{ij} (Fig. 4-(d)).

Centrality Analysis. The centrality of a node represents its influence. Various centrality measures have been proposed in the literature, such as the degree centrality [36], Eigenvector centrality [37], and PageRank centrality [38], etc. We choose *PageRank* as it has three properties that make it a good fit to approximate a drone's malicious influence in the SVG. (1) It is efficient to compute for even large graphs using the power method [39]. (2) It increases a drone's influence if more swarm members are maliciously influenced by the drone. (3) It decreases a drone's influence if its neighboring drones are difficult to influence, or if they are many hops away.

We use the SVG to calculate the influence of potential target drones, and the transposed SVG to calculate the influence of potential victim drones. This is because the transposed SVG models how a specific drone is influenced by its neighbors.

Seed Scheduling. As mentioned before, the probability of causing collisions for each seed depends on (1) the influence score of the drone pair $T - V$; and (2) the VDO. To schedule the seeds based on these two factors, given θ, we (1) sort each victim drone based on the VDO in the ascending order; (2) calculate the summative influence $I(\theta)_{ij}$ of each possible combination of drone pairs; (3) for each victim drone v, choose the target drone T involved in the drone pairs that have the highest summative influence, i.e., $T = \underset{j}{\text{argmax}} \quad I(\theta)_{jv}$.

C. Search-based fuzzing

To tackle C2, our goal is to find the appropriate spoofing parameters (i.e., t_s and Δt) that will cause the collision, given a specific seed. We model this as an optimization problem - the goal is minimizing the objective function $f(t_s, \Delta t)$, the distance between the victim drone and the obstacle, subject to the timing constraints, i.e., $t_s + \Delta t < t_{mission}$. Collision occurs only if the global minimum is found to be non-positive.

We make the observation that the objective function f *is convex* (Fig. 5-(e)). This is because performing spoofing for either too short (e.g., Δt_1) or too long (e.g., Δt_3) a time will cause the victim drone to avoid the obstacle from either side. For example, the victim drone avoids the obstacle from the left side in the absence of the attack (Fig. 5-(a)). A too short spoofing time Δt_1 would make it miss the obstacle from the left (Fig. 5-(b)), When the time increases to Δt_2 (Fig. 5-(c)), it causes a collision. However, if the spoofing time is too long and increases to Δt_3, the victim drone avoids the obstacle from the right (Fig. 5-(d)), thus increasing the value of $f(t_s, \Delta t)$ (Fig. 5-(e)). Thus, we need to find a spoofing time that is neither too short nor too long, to cause the collision.

Gradient-guided Optimization. Since the objective function is convex, we use gradient-guided search to efficiently find an optimal solution. We (1) calculate the partial derivatives of $f(t_s, \Delta t)$, i.e., $\frac{\partial f}{\partial t_s}$ and $\frac{\partial f}{\partial \Delta t}$; (2) update the spoofing parameters t_s as in Equation 1a and Δt as shown in Equation 1b.

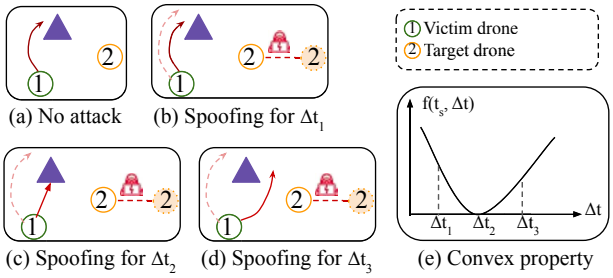


Fig. 5: Convex property of the objective function in SwarmFuzz.

$$t_s = max(t_s - lr * \frac{\partial f}{\partial t_s}, 0) \tag{1a}$$

$$\Delta t = max(\Delta t - lr * \frac{\partial f}{\partial \Delta t}, 0) \tag{1b}$$

lr denotes the learning rate, which is used to scale the input updates and thus increase the searching speed. We restrict the value of t_s and Δt to be positive, as time cannot be negative.

For each target-victim drone pair seed in the seedpool, SwarmFuzz repeats the above gradient-guided searching process. If a collision occurs, SwarmFuzz reports that it successfully found the SPV, and outputs $< T - V, \theta, t_s, \Delta t >$. However, if no collision occurs within the maximum number of search iterations (set empirically - Section V-A), SwarmFuzz abandons the current seed and switches to the next seed in the seedpool.

V. Evaluation

First, we present our experimental setup, and then the results for effectiveness of SwarmFuzz in various swarm configurations. Finally, we perform an ablation study of SwarmFuzz's heuristics.

A. Experimental Setup

Simulator. We use SwarmLab [28], a popular swarm simulator, as it implements the drone dynamics accurately [28], and also implements state-of-the-art control algorithms. To evaluate SwarmFuzz, we choose the Viscek algorithm [27], one of the two swarm control algorithms implemented by the simulator. This algorithm was published in 2018 and has been cited 315 times as of February 2023. It has also been validated on real hardware with a 30 drone swarm, and performs well in terms of maintaining the formation as well as obstacle avoidance [27]. The Viscek algorithm performs collision avoidance based solely on the GPS sensor reading. Each drone in SwarmLab is a quadcopter weighted at 0.296kg (by default), and using a PID (Proportional-Integral-Derivative) flight controller.

GPS Spoofing. As we did not have access to GPS transmitter hardware, we simulate GPS spoofing attacks through software code modification - this is similar to what a lot of prior work has done [13], [19]. We launch GPS spoofing in software by manipulating the GPS reading to $GPS + d$ at the GPS sampling rate (100 Hz in SwarmLab by default), where d is the spoofing deviation. We consider d to be $5m$ and $10m$ respectively. Most defense techniques [19], [20] ignore spoofing distances of less than $10m$ as such small deviations are indistinguishable from standard GPS offset [33], in order to avoid false-positives. Therefore, we inject only small amounts of noise (i.e., $5m/10m$) during GPS spoofing as this is hard to detect by current defense techniques, and thus is stealthy. Even with this small GPS spoofing distance, we show that the attacker can achieve her objective with a high success rate.

Mission Details. We consider the whole phase of a mission that aims to reach a pre-defined destination (233.5m away), while avoiding a single on-path obstacle. Each instance of this mission in SwarmLab takes around $120s$ to finish on average. The obstacle is placed at roughly the half-way mark of the mission, so that the drone swarm has enough time to react for the collision avoidance. Because the obstacle is known to the mission, the drone swarm can avoid colliding with the obstacle under normal operation. To reduce the bias, the initial location of the drone swarm is randomly generated within a range of $0 - 50m$ relative to the mission starting point. We limit the range to $50m$ as 1) the drone swarm is sparse even with a large size (e.g., 15 drones) and is unlikely to have collisions in the absence of attack; 2) the drone swarm has sufficient time to react for collision avoidance in the $233.6m$ long mission.

Success Metric. In the absence of attacks, we find that no collision occurs in any mission. We consider an SPV to be successfully found, if and only if the victim drone crashes into the on-path obstacle under GPS spoofing. Note that we do not consider collisions caused directly by the target drone (i.e., the target drone collides with the victim or the obstacle).

B. Effectiveness of SwarmFuzz across Swarm Configurations

We apply SwarmFuzz to six drone swarm configurations, with swarm sizes of 5, 10, 15 drones, and GPS spoofing distances of $5m$ and $10m$. We perform 100 missions for each configuration, to obtain a representative sample. We also manually validate each vulnerability found by SwarmFuzz via simulation, and find that all of them are True Positives (TP). Then we measure in how many missions SwarmFuzz found SPVs- this is the success rate of SwarmFuzz for that configuration. An alternate way to measure success rate is the ratio of the number of SPVs found to the maximum number of SPVs for a given configuration. However, we cannot easily obtain the maximum possible number of SPVs as this requires exhaustive sampling of the input space, which is prohibitively expensive.

TABLE I: Success rates of SwarmFuzz in finding SPVs.

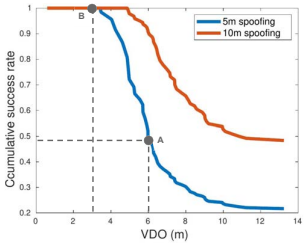
Swarm size	5 drones	10 drones	15 drones
5m spoofing	21%	36%	54%
10m spoofing	49%	59%	74%

Table I shows the results. We observe that success rates of SwarmFuzz vary from 21% to 74% across different configurations (average 48.8%). *This shows that SwarmFuzz is highly effective in finding SPVs for different swarm configurations.*

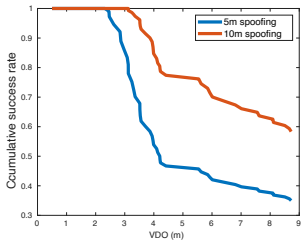
Further, SwarmFuzz has a higher success rate for missions when the (1) GPS spoofing distance is higher, as it allows the attacker to disrupt the swarm more, and (2) when the swarm

size is higher, as larger swarms are denser, making obstacle avoidance more difficult. We analyze the reasons below.

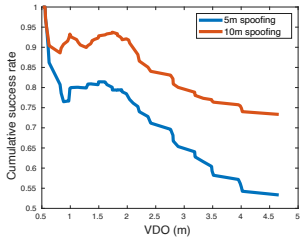
Recall that in Section IV-B, we choose the drones with lower VDO (i.e., closer to the obstacle) as the promising victim drones. Given the same GPS spoofing deviation and swarm size, we find that the success rate varies significantly based on the VDO of each mission. The VDO is determined based on the mission parameters, which are generated randomly. Therefore, to analyze how the VDO influences the success rate, we choose the cumulative success rate as the metric. For a VDO value x, the cumulative success rate represents the success rate of missions whose VDO value is smaller than x. For example, in Fig. 6a, for all missions with VDO no larger than $6m$, the success rate of SwarmFuzz is 48% (point 'A').



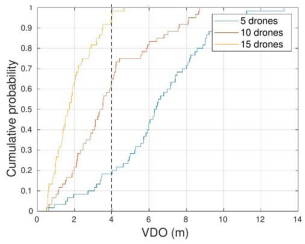
(a) Success rates in 5 drones.



(b) Success rates in 10 drones.



(c) Success rates in 15 drones.



(d) CDF for the VDO.

Fig. 6: Success rates across swarm configurations.

Fig. 6a- 6c show the cumulative success rate of SwarmFuzz with respect to the VDO under different swarm configurations. We make the following observations: (1) The cumulative success rate of SwarmFuzz roughly decreases with the VDO, meaning that missions with lower VDOs are more vulnerable to the attacks exploiting SPVs. This is understandable, as drone missions that fly closer to the obstacle are easier to cause collisions with the obstacle, (2) Higher GPS spoofing distances achieve higher success rates. Again, this is intuitive as higher spoofing distance causes more disruptions. In Fig. 6a, in a 5-drone swarm, for missions with VDOs lower than $3m$, even a $5m$ spoofing can achieve 100% success rate (point 'B'). Therefore, missions with low VDOs are highly vulnerable.

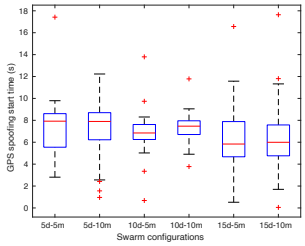
In Table I, we observe that the success rate increases with the swarm size. To understand why, we plot the Cumulative Distribution Function (CDF) of the VDOs in all missions in Fig. 6d. For a VDO x, the empirical CDF $F(x)$ is the proportion of missions with VDOs no larger than x. For example, in Fig. 6d, in a 5-drone swarm, only 20% of missions

TABLE II: Average number of search iterations taken by SwarmFuzz to find SPVs across swarm configurations.

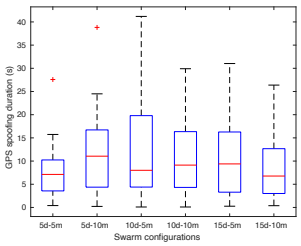
	5-drone	10-drone	15-drone
5m-spoofing	6.33	9.3	12.65
10m-spoofing	6.93	9.91	13.47

have a VDO of atmost $4m$. In a 10-drone and 15-drone swarm, however, the proportion of missions with atmost $4m$-VDO increases to 65% and 98% respectively. Thus, the 5-drone swarm is more resilient than swarms of 10 and 15 drones. Therefore, the increase of swarm size leads to a decrease of VDO, making the swarm mission more vulnerable to attacks.

Fig. 7 shows the GPS spoofing parameters (i.e., the starting time and the duration) to trigger the SPVs under different swarm configurations. These are found by SwarmFuzz during the gradient-based optimization process. We find that the average GPS spoofing starting time across different configurations is $6.91s$, and the average GPS spoofing duration is $10.33s$.



(a) GPS spoofing starting time.



(b) GPS spoofing duration.

Fig. 7: GPS spoofing parameters found by SwarmFuzz across swarm configurations. In the figure, "5d-5m" means 5-drone swarms under 5m-spoofing, and so on.

We also report the average number of search iterations across different swarm configurations in Table II. Each iteration takes 120s (the mission time), and hence this is correlated with the runtime. We find that the average runtime increases with the size of the drone swarm, since the interactions among drones become more complex. For a given swarm size, the runtime overheads under different GPS spoofing deviations exhibit no significant differences across configurations.

C. Ablation Study: Effectiveness of SwarmFuzz's Heuristics

To the best of our knowledge, there is no existing fuzzing tool to find SPVs in drone swarms, and thus there is no prior work we can use for comparison with SwarmFuzz. Instead, we perform an ablation study and compare SwarmFuzz with different variants of itself, as well as with random fuzzing.

We used two heuristics to improve the efficiency of SwarmFuzz in Section IV, (1) using the SVG to prioritize the influential drone pairs selected for fuzzing; (2) using gradient-guided optimization to find spoofing parameters for causing collisions. To evaluate the effect of these heuristics, we compare SwarmFuzz with three other fuzzers as follows. (1) R_Fuzz does not implement either heuristic, and instead randomly chooses the drone pairs as well as the spoofing parameters.

TABLE III: Comparison of fuzzers in 5 drones, 10m spoofing.

	SwarmFuzz	R_Fuzz	G_Fuzz	S_Fuzz
Success rate	49%	8%	5%	12%
Avg. iterations	6.93	19.52	6.75	19.85

(2) G_Fuzz only implements the gradient-guided optimization to search for spoofing parameters but not the SVG, instead choosing these pairs randomly. (3) S_Fuzz only implements the SVG to choose the drones pairs but not the gradient-guided optimization, instead choosing the spoofing parameters randomly. Therefore, we evaluate the efficacy of the SVG by comparing SwarmFuzz with G_Fuzz, and the efficacy of gradient-guided optimization by comparing it with S_Fuzz.

We measure two metrics for each fuzzer, the success rate and the runtime overhead. For runtime overhead, we report the average number of search iterations taken by the fuzzer - each iteration takes about 2 minutes (time taken by a mission).

We also cap the number of search iterations for each seed to 20 in all the fuzzers based on our empirical observations. We experimentally observe that the number of vulnerabilities found saturates at 20 search iterations, which corresponds to 40 min. We evaluate the fuzzers with the 5-drone swarm with a $10m$ GPS spoofing distance. Table III shows the results.

We find that the success rate of SwarmFuzz is 49%, whereas the success rate of R_Fuzz is just 8%, and that of the G_Fuzz just 5%. Further, the runtime overhead of SwarmFuzz and G_Fuzz is low (about 7 iterations), unlike those of R_Fuzz and S_Fuzz (about 20 iterations, i.e., maximum number of search iterations). Compared to G_Fuzz, the success rate of SwarmFuzz is about 10x higher, while compared to S_Fuzz, its runtime overhead is 3x lower. Thus, the SVG boosts the success rate of SwarmFuzz by almost 10x, while the gradient-guided optimization reduces its runtime overhead by about 3x. Compared to the random fuzzer (R_Fuzz), SwarmFuzz has about 6x higher success rate and 3x smaller runtime overhead.

VI. Discussion

Implications SwarmFuzz has a high success rate in finding SPVs in drone swarms, within a small amount of time. This allows these vulnerable missions to be discovered early.

SwarmFuzz finds that swarm missions with a larger size are more vulnerable to attacks exploiting SPVs, since they have low VDOs. This suggests that designers should expend more effort in securing large-size drone swarms. SwarmFuzz further finds that swarm missions with low VDOs are generally more vulnerable. Therefore, if the VDO is low, then a stricter protection technique against GPS spoofing attacks may be needed. These are potential future work directions.

Moreover, to the best of our knowledge, there are no fault-tolerance mechanisms [40], [41] (e.g., achieving agreement and consensus among drones; verifying other drones' location through other sensors) deployed in mainstream distributed drone swarm systems today. Our work provides a strong motivation for why drone swarm algorithms need such fault-tolerance mechanisms, in order to be resilient to SPVs.

Limitations Our work has two limitations. First, we only tested SwarmFuzz on one swarm control algorithm. However, SwarmFuzz does not utilize any knowledge specific to this swarm control algorithm or the mission during fuzzing. Instead, it only uses general goals designed in the swarm control algorithms and physical properties of collisions, i.e., the convex property of the objective function. Therefore, it should also work on other decentralized swarm control algorithms.

Second, in our experiments, we used missions of fixed length, and with a fixed obstacle placement in order to keep the experiment simple. However, to model other types of missions, for example, with multiple obstacles, we only need to change one input - the coordinates of the obstacle for collision. Hence, SwarmFuzz should also work on other swarm missions.

VII. Related work

Physical attack on drones have targeted GPS [9], [10], [29]–[31], [42], [43], gyroscope [12], accelerometer [11] and optical-flow sensors [44]. However, these attacks only focus on single drones, and none of them have considered drone swarms. Defense techniques against physical attacks targeting GPS sensors have also been proposed. Unfortunately, most defense techniques [19], [20] ignore $0 - 10m$ GPS spoofing distance as it is indistinguishable from the standard GPS offset [33], and is sufficient to ensure the safety of a single drone. Thus, these defenses cannot handle attacks exploiting SPVs.

Fuzzing has been used in drones for finding software bugs [45], policy violation bugs [46], and logic flaws [5], [47]. Model checking has been used to evaluate the effects of a single drone's sensor failures [48]. SWARMFLAWFINDER [5] tests the drone swarm's behavior by introducing an external attack drone, which incurs relatively high costs and is easy to detect with intruder detection techniques [7]. Other work finds vulnerabilities in drone swarms by introducing spurious messages in the swarm's communication [6]. However, such attacks can be prevented using message encryption [34]. Therefore, prior work cannot find SPVs in drone swarms. In a prior poster [49], we demonstrated the existence of SPVs, but did not design a systematic methodology to find them.

VIII. Conclusion

In this paper, we highlight a new kind of vulnerability in drone swarms called Swarm Propagation Vulnerabilities (SPVs), which can be exploited to indirectly cause disruptions (e.g., collisions) in drone swarms with GPS spoofing. To help defenders evaluate the resilience of swarms against SPVs, we propose SwarmFuzz, a fuzzing technique to efficiently find SPVs in drone swarms. SwarmFuzz utilizes graph centrality analysis to find influential drone pairs and gradient-guided optimization to find the spoofing parameters to cause collisions.

We evaluate SwarmFuzz on a popular swarm control algorithm. We find that (1) SwarmFuzz can achieve an average success rate of 48.8% in finding SPVs; (2) swarms of larger size are more vulnerable to SPVs, for a given spoofing distance; and (3) SwarmFuzz's efficacy and efficiency are due to the graph centrality analysis and gradient-guided optimization.

ACKNOWLEDGEMENTS

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and a Four Year Fellowship from UBC. We acknowledge travel support from the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC. We also thank the anonymous reviewers of DSN'23 for their helpful comments.

REFERENCES

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, 2013.

[2] A. Boyd, "The pentagon wants ai-driven drone swarms for search and rescue ops," Available at: https://www.nextgov.com/emerging-tech/2019/12/pentagon-wants-ai-driven-drone-swarms-search-and-rescue-ops/162113/, 2019.

[3] "Droneseed," Available at: https://droneseed.com/rapid-reforestation.

[4] S. Dimitropoulos, "If one drone isn't enough, try a drone swarm," Available at: https://www.bbc.com/news/business-49177704, 2019.

[5] C. Jung, A. Ahad, Y. Jeon, and Y. Kwon, "Swarmflawfinder: Discovering and exploiting logic flaws of swarm algorithms," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[6] X. Huang, Y. Tian, Y. He, E. Tong, W. Niu, C. Li, J. Liu, and L. Chang, "Exposing spoofing attack on flocking-based unmanned aerial vehicle cluster: A threat to swarm intelligence," *Security and Communication Networks*, vol. 2020, pp. 1–15, 2020.

[7] M. R. Brust, G. Danoy, P. Bouvry, D. Gashi, H. Pathak, and M. P. Gonçalves, "Defending against intrusion of malicious uavs with networked uav defense swarms," in *2017 IEEE 42nd conference on local computer networks workshops (LCN workshops)*. IEEE, 2017, pp. 103–111.

[8] L. Liu, H. Qian, and F. Hu, "Random label based security authentication mechanism for large-scale uav swarm," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking*. IEEE, 2019, pp. 229–235.

[9] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 2011 ACM SIGSAC Conference on Computer and Communications Security*, 2011.

[10] T. E. Humphreys, B. M. Ledvina, M. L. Psiaki, B. W. O'Hanlon, P. M. Kintner *et al.*, "Assessing the spoofing threat: Development of a portable gps civilian spoofer," in *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, 2008, pp. 2314–2325.

[11] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks," in *2017 IEEE European Symposium on Security and Privacy*, 2017.

[12] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium (USENIX Security 15)*, Aug. 2015.

[13] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Out of control: Stealthy attacks against robotic vehicles protected by control-based techniques," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.

[14] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen, "Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under gps spoofing," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security '20)*, Aug. 2020.

[15] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[16] "Gps jammers used in 85% of cargo truck thefts – mexico has taken action," Available at: https://rntfnd.org/2020/10/30/gps-jammers-used-in-85-of-cargo-truck-thefts-mexico-has-taken-action/, October 2020.

[17] W. Z. Simone McCarthy and D. Tsang, "Hk $1 million in damage caused by gps jamming that caused 46 drones to plummet during hong kong show," Available at: https://www.scmp.com/news/hong-kong/law-and-crime/article/2170669/hk13-million-damage-caused-gps-jamming-caused-46-drones, 2018.

[18] M. Posch, "Knowing your place: the implications of gps spoofing and jamming," Available at: https://hackaday.com/2022/05/23/knowing-your-place-the-implications-of-gps-spoofing-and-jamming/, 2022.

[19] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "Pid-piper: Recovering robotic vehicles from physical attacks," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021.

[20] H. Sathaye, G. LaMountain, P. Closas, and A. Ranganathan, "Semperfi: Anti-spoofing gps receiver for uavs," in *Proceedings 2022 Network and Distributed System Security Symposium*, 2022.

[21] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "RVFuzzer: Finding input validation bugs in robotic vehicles through Control-Guided testing," in *28th USENIX Security Symposium (USENIX Security 19)*, Aug. 2019.

[22] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles," in *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

[23] S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, "kafl: Hardware-assisted feedback fuzzing for os kernels." in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 167–182.

[24] C. Aschermann, S. Schumilo, T. Blazytko, R. Gawlik, and T. Holz, "Redqueen: Fuzzing with input-to-state correspondence." in *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[25] P. Chen and H. Chen, "Angora: Efficient fuzzing by principled search," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[26] D. She, A. Shah, and S. Jana, "Effective seed scheduling for fuzzing with graph centrality analysis," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[27] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, p. eaat3536, 2018.

[28] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: a matlab drone swarm simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005–8011.

[29] H. Sathaye, M. Strohmeier, V. Lenders, and A. Ranganathan, "An experimental study of GPS spoofing and takeover attacks on UAVs," in *31st USENIX Security Symposium (USENIX Security 22)*, Aug. 2022.

[30] J. Noh, Y. Kwon, Y. Son, H. Shin, D. Kim, J. Choi, and Y. Kim, "Tractor beam: Safe-hijacking of consumer drones with adaptive gps spoofing," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–26, 2019.

[31] D. He, Y. Qiao, S. Chen, X. Du, W. Chen, S. Zhu, and M. Guizani, "A friendly and low-cost technique for capturing non-cooperative civilian unmanned aerial vehicles," *IEEE Network*, 2019.

[32] K. D. Wesson, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "An evaluation of the vestigial signal defense for civil gps anti-spoofing," in *Proceedings of the 24th International Technical Meeting of the Satellite Division of The institute of navigation (ION GNSS 2011)*, 2011, pp. 2646–2656.

[33] "Global positioning system standard positioning service performance standard," Available at: https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf, 2008.

[34] Y.-M. Kwon, J. Yu, B.-M. Cho, Y. Eun, and K.-J. Park, "Empirical analysis of mavlink protocol vulnerability for attacking unmanned aerial vehicles," *IEEE Access*, 2018.

[35] M. E. Newman, "The mathematics of networks," *The new palgrave encyclopedia of economics*, vol. 2, no. 2008, pp. 1–12, 2008.

[36] W. Dubitzky, O. Wolkenhauer, K.-H. Cho, and H. Yokota, *Encyclopedia of systems biology*. Springer New York, 2013, vol. 402.

[37] J. Golbeck, *Analyzing the social web*. Newnes, 2013.

[38] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford infolab, Tech. Rep., 1999.

[39] D. Herrmann and D. Herrmann, "Von-mises-iteration," *Numerische Mathematik—40 BASIC-Programme*, pp. 93–96, 1983.

[40] M. Dorigo, G. Theraulaz, and V. Trianni, "Swarm robotics: Past, present, and future [point of view]," *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[41] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Fault-tolerant cooperative navigation of networked uav swarms for forest fire monitoring," *Aerospace Science and Technology*, vol. 123, p. 107494, 2022.

[42] S. P. Arteaga, L. A. M. Hernández, G. S. Pérez, A. L. S. Orozco, and L. J. G. Villalba, "Analysis of the gps spoofing vulnerability in the drone 3dr solo," *IEEE Access*, vol. 7, pp. 51 782–51 789, 2019.

[43] J. Gaspar, R. Ferreira, P. Sebastião, and N. Souto, "Capture of uavs through gps spoofing using low-cost sdr platforms," *Wireless Personal Communications*, vol. 115, pp. 2729–2754, 2020.

[44] D. Davidson, H. Wu, R. Jellinek, T. Ristenpart, and V. Singh, "Controlling uavs with sensor input spoofing attacks," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[45] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "RVFuzzer: Finding input validation bugs in robotic vehicles through Control-Guided testing," in *28th USENIX Security Symposium (USENIX Security 19)*, Aug. 2019.

[46] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles," in *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

[47] C. Jung, A. Ahad, J. Jung, S. Elbaum, and Y. Kwon, "Swarmbug: Debugging configuration bugs in swarm robotics," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.

[48] M. Taylor, H. Chen, F. Qin, and C. Stewart, "Avis: In-situ model checking for unmanned aerial vehicles," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021.

[49] Y. E. Yao, P. Dash, and K. Pattabiraman, "Poster: May the swarm be with you: Sensor spoofing attacks against drone swarms," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 3511–3513.