Software and Behavior Diversification for Swarm Robotics Systems

Ao Li
Washington University in St. Louis
St Louis, Missouri, USA
ao@wustl.edu

Sinyin Chang
National Taiwan University
Taipei, Taiwan
b10902087@ntu.edu.tw

Guorui Li
Washington University in St. Louis
St Louis, Missouri, USA
guorui.l@wustl.edu

Yuanhaur Chang
Washington University in St. Louis
St Louis, Missouri, USA
c.yuanhaur@wustl.edu

Nathan Fisher
Wayne State University
Detroit, Michigan, USA
fishern@wayne.edu

Thidapat (Tam) Chantem
Virginia Tech
Arlington, Virginia, USA
tchantem@vt.edu

ABSTRACT

Inspired by natural swarms, swarm robotics systems are used in safety-critical tasks due to their scalability, redundancy, and adaptability. However, their design exposes them to two primary vulnerabilities. First, their homogeneity makes them vulnerable to large-scale attacks. Second, logical flaws within swarm algorithms can be exploited, leading to mission failures or crashes. While existing studies can effectively identify these vulnerabilities using system testing and verification, they are often time-consuming and might require repetition following software updates. To this end, we propose a complementary, two-level diversification approach. The first level tackles system homogeneity through software diversification. The second level introduces algorithmic randomness to minimize the exploitability of logical flaws. By leveraging a social force model, we can ensure that the introduced randomized behaviors do not compromise safety. Our evaluations show that the performance overheads remain within acceptable limits, notably at 2% for missions characterized by self-organizing behaviors.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security.*

KEYWORDS

Diversification; Cyber-physical systems; Swarm robotics security

ACM Reference Format:
Ao Li, Sinyin Chang, Guorui Li, Yuanhaur Chang, Nathan Fisher, and Thidapat (Tam) Chantem. 2023. Software and Behavior Diversification for Swarm Robotics Systems. In *Proceedings of the 10th ACM Workshop on Moving Target Defense (MTD '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3605760.3623765

1 INTRODUCTION

A swarm robotics system is a revolutionary approach that harnesses the power of collective intelligence to perform challenging tasks efficiently. Drone swarms, for instance, have been a topic of heated discussion in recent years, especially with respect to their vast applications in logistics [1] and search-and-rescue [8].

However, certain vulnerabilities are inherent to the design nature of swarm robotics systems. The primary concern arises from the homogeneity of its implementation. Swarm robotics systems are predominantly composed of agents that are physically and behaviorally uniform. This design choice often stems from ethological models. However, homogeneity has its downsides. Attackers can easily obtain an identical copy of the common software running on the targeted systems and search it for vulnerabilities. After converting a vulnerability into an exploit [22], they can target all systems using that vulnerable software version. The second vulnerability stems from the swarm algorithm. A swarm mission is controlled by a swarm algorithm, which coordinates the actions of multiple robots. Logical flaws in a swarm algorithm can result in various failures. Existing literature [10] showcased the potential to exploit these logical flaws, leading to mission failures or even crashes.

Existing Work. Randomization [12] effectively addresses the primary vulnerability of homogeneity. However, for tackling logical flaws, current methodologies primarily rely on testing [10, 11, 24] and verification [18]. While these methods can effectively identify bugs, they often require substantial time and manual efforts, and may need to be repeated following software updates.

Our Solution. In this paper, we introduce a complementary method to existing approaches: a two-level diversification strategy designed to tackle the two types of vulnerabilities mentioned earlier. The first level, software diversification, mitigates homogeneity issues. This is achieved through compiler-generated software diversification [9]. For the second level, which we also refer to as behavior diversification or algorithm-level diversification, we introduce randomness into the agents' behaviors. Based on the observation that attacks on swarm algorithms are tailored to specific scenarios, behavior diversification increases the unpredictability of these exploitable scenarios, thereby reducing the attack surface. While software-level diversification is well understood, diversification at the algorithmic level has never been studied in previous research. A primary challenge in achieving algorithmic diversification is ensuring that randomized behaviors do not introduce new safety concerns. To address this, a social force model [7] is utilized to predict potential collisions for all candidate behaviors. Based on these predictions,

This work is licensed under a Creative Commons Attribution International 4.0 License.

MTD '23, November 26, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0256-3/23/11.
https://doi.org/10.1145/3605760.3623765

behaviors that present no risk of collisions are selected for runtime randomization.

Evaluation Results. Our evaluations on swarm drones [17] and swarm vehicles [2] indicate that the overheads are within acceptable ranges. Specifically, software diversification results in negligible runtime overhead. As for algorithm-level diversification, its impact on mission completion time varies significantly based on the type of mission. For missions characterized by more self-organizing behaviors, our method incurs only a 2% overhead. One limitation of our study is that we did not evaluate the effectiveness of our method against attacks. Future work will address this gap and further delve into optimizing the behavior randomization scheme.

2 BACKGROUND

2.1 Preliminary of Swarm Robotics Systems

Drawing inspiration from the coordination observed in natural swarms of animals like ants, swarm robotics systems capitalize on the strengths of a collection of autonomous robots that work collaboratively in a decentralized manner. These robots operate based on local interactions with the environment and neighboring robots, allowing themselves to adapt to uncertain conditions since each individual can make decisions based on its perception. Homogeneity emerges as a crucial attribute that characterizes swarm robots by uniformity [23]. Since each robot shares the same physical attributes, functionalities, and control mechanisms, this uniformity contributes to a synchronized collective behavior that enables the swarm to work seamlessly as a unit. These robots are able to exhibit consistent responses to control inputs by sharing the same communication protocols and control algorithms.

From the perspective of swarm engineering, swarm algorithm design paradigms can be categorized into behavior-based design and automatic design [3]. As a more commonly adopted paradigm, behavior-based design defines individual behaviors for autonomous agents to achieve more sophisticated collective behaviors. Among these behavior-based design methods, the finite state machine [3] (FSM) is the most widely adopted method for achieving collective behaviors in swarm robots. In this approach, each state in the machine represents a specific behavior, and the transitions between these behaviors are determined by the sensory inputs of a robot, as well as the ongoing behaviors of other robots. The behavior generated by the FSM subsequently guides the execution of the underlying controllers responsible for actuation. This paradigm distinctly separates decision-making from control into an independent layer [4], facilitating a more straightforward design for development and maintenance. The approach proposed in this paper is built upon this design paradigm.

2.2 Swarm Robotics System Security

In comparison to a single robot, a swarm robotics system introduces two primary types of vulnerabilities. The first is the increased complexity of communication between robots. The second pertains to the collective behavior that is coordinated amongst the robots.

Communication Security. In terms of communication, a few research efforts, such as LISA [5] and SCRAPS [20], address the scalability challenge of remote attestation schemes in swarm scenarios. However, [19] highlights additional pitfalls in existing communication pairing techniques, which still remain open.

Logical Flaws in Swarm Algorithm. Prior works have also investigated the potential exploitation of logical flaws in swarm algorithms to induce mission failures or crashes. SwarmFuzz [24] investigated the potential of using GPS spoofing on one drone to alter its behavior, which in turn influences other drones in the swarm. This may result in erroneous control commands in the affected drones. Such incorrect commands can force these drones to stray from their planned routes, raising the risk of collisions. Similarly, SwarmFlawFinder [10] highlights the potential for a malicious drone to create similar disruptive effects. To address such vulnerability, SwarmFuzz [24], SwarmBug [11], and SwarmFlawFinder [10] employ testing methods to identify these logical flaws or misconfigurations. In contrast to these testing-based approaches, [18] employed a model-checking approach to pinpoint discrepancies between design requirements and actual implementation, thereby uncovering logical flaws. Our method complements these previous work by providing continuous protection, consistently making vulnerabilities harder for attackers to exploit.

3 THREAT MODEL

Attacker's Goal. We consider an active attacker who aims to compromise the swarm algorithm in a swarm robotics system, ultimately disrupting its normal operation. For example, the attacker may seek to crash swarm drones during mission execution by exploiting behavioral flaws [10] or spoofing GPS signals [24].

Attacker's Knowledge and Capabilities. In line with common assumptions made in existing works [10, 22], we assume that the attacker is aware of exploitable vulnerabilities in the target swarm robotics system. This awareness could be gained either through public software vulnerability disclosures or by reverse engineering after obtaining the corresponding firmware via physical access. Furthermore, we assume that the attacker can deploy a malicious agent to disrupt the target swarm's operation by exploiting vulnerabilities in swarm algorithms.

4 APPROACH

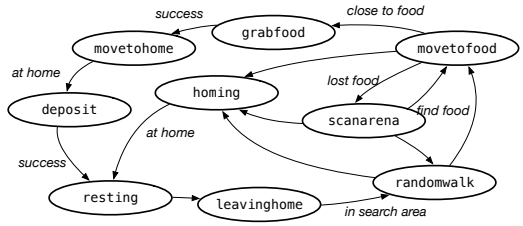


Figure 1: An example of a probabilistic finite state machine for collective foraging, adapted from [14].

This section introduces a two-layer diversification approach. Since existing randomization techniques [12] are widely recognized, the focus will be on algorithm-level diversification. This is also referred to as behavior diversification.

4.1 Behavior Diversification

The proposed approach is built upon *Behavior-based swarm algorithm*, which is the predominant design paradigm in swarm systems [3]. Within this paradigm, behaviors and their transitions are typically represented using a Finite State Machine (FSM). Figure 1 illustrates the FSM for collective foraging as described in [14], where each state represents a distinct behavior. By introducing randomness to the transitions between behaviors within the FSM, the proposed approach enhances the unpredictability of the victim agent's behaviors.

Introducing Randomness to Swarm Behaviors. Behaviors are chosen either deterministically or probabilistically, guided by predefined criteria. These decisions hinge on sensor inputs and the observed behaviors of other agents. To introduce the unpredictability in behaviors, we've integrated a randomization mechanism into the behavior decision-making process. This mechanism selects behaviors at random rather than adhering to the pre-established rules. Given that the attack strategy outlined in [10] relies on scenarios with specific behavior patterns, introducing diversified behaviors makes it harder for attackers to predict and exploit these scenarios.

Mitigating Collision Risks from Randomized Behaviors. Introducing randomized behavior poses a challenge: the selected behavior might inadvertently lead to safety issues. A solution is needed to estimate potential collisions so that these behaviors can be avoided. To this end, we utilize the social force model [7] (SFM). SFM interprets individual agent's movements as particles influenced by forces such as the desired movement direction and repulsions from other agents or obstacles. Continuously assessing these forces allows the SFM to anticipate potential collisions. Our choice of SFM is driven by two factors: its flexibility in modeling a range of swarm scenarios through parameter adjustments, and its capability for real-time computation, which is vital for real-time robotics systems. Once a candidate behavior is selected, its corresponding target position is pre-calculated using the underlying controller. This predetermined target position is then input into the SFM to assess the potential for collisions. If a collision is predicted, the behavior will be aborted and a new behavior will be re-selected.

Returning to the Previous State. After executing the diversified behavior for a predefined short period, the robot returns to its original state and resumes its mission. Subsequently, the randomization will be triggered again at configurable intervals. Such occasional behavior diversification causes disruptions in the planned mission, resulting in delays to the mission completion time. The effects on mission delays are evaluated preliminarily in §6. The analysis of the associated trade-offs is left for future research.

How does SFM predict with the presence of malicious behavior? While the predictions made by the Social Force Model (SFM) are effective only for benign behaviors, a malicious drone could deceive other drones by exhibiting counter-intuitive actions. Nevertheless, these deceptive actions wouldn't affect the predictions related to the behaviors of benign agents. In the worst-case scenario, a benign agent might incorrectly predict the adversarial drone's behavior, leading to a collision. However, if an adversarial drone's intent is to physically collide with its target, such a collision could be inevitable.

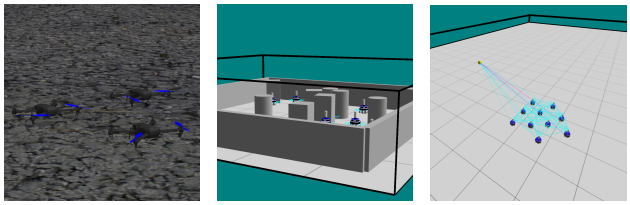
5 IMPLEMENTATION

Software Diversification. The software diversification is implemented using compiler passes in LLVM 16. Three types of passes have been implemented: *Global Data Randomization*, which handles the random reordering and padding of global variables; *Function Randomization*, which reorders the layout of functions; *Stack Element Randomization*, which randomizes the stack layout.

Behavior Diversification. Behavior diversification has been implemented in a selection of swarm algorithms using ARGoS [21]. To facilitate this, LLVM passes were employed to extract the names of all functions. Subsequent manual inspections were then conducted to discern the specific behaviors associated with each function, ensuring precision in the extraction process. The utilized social force model is built upon a widely recognized derivative within the field of robotics [15] with manual customized parameters. For effective monitoring and ensuring real-time responsiveness, the state-checking interval is configured to be $0.5s$, allowing for timely adjustments in the behavior of the swarm if necessary.

6 EVALUATION

This section aims to address two primary questions: 1) What are the performance impacts of software-level diversification? and 2) How much delay does algorithm-level diversification introduce to mission completion?



(a) PX4 in Gazebo. (b) Diffusion in ARGoS3. (c) Diffusion in ARGoS3.

Figure 2: Evaluation scenarios.

Experimental Setup. We conducted two sets of experiments in two simulated environments, as depicted in Figure 2, to evaluate software diversification and algorithm diversification, respectively.

Software Diversification. The first environment is set up in Gazebo, using the PX4 Autopilot [17] as the control software for drones. To emulate swarm behavior, we chose an open-source fleeting swarm algorithm based on MAVROS [16] to coordinate three drones in the simulation. The control software stacks were executed on a Raspberry Pi 3B+ with 1GB RAM.

Algorithm Diversification. The second environment is based on ARGoS3 [21], a simulation engine that supports the ground-based marXbot robot [2]. Among the built-in missions in ARGoS3, we selected five to run using both the original algorithm and the modified one. We then compared the mission completion times across the two versions to evaluate the delays introduced by our methods. These experiments were conducted on MAC OSX with a 2.6 GHz 6-Core Intel Core i7 and 16GB 2667 MHz DDR4 RAM.

Software Diversification Overhead. Table 1 presents the overheads introduced by instrumentation on software diversification. Observing the computational time associated with the trajectory control algorithm, the overheads span a range of 1.36% to 4.55%.

Table 1: Software Diversification Overhead

	Time (ms)	Memory (Bytes)	Code Size (Bytes)
Baseline	0.614020	421.108M	0.0512M
Function	0.622344 (+1.36%)	427.788M (+1.59%)	0.0541M (+5.7%)
Global Variable	0.638285 (+3.95%)	427.788M (+1.59%)	0.0545M (+6.4%)
Stack Padding	0.641954 (+4.55%)	427.788K (+1.59%)	0.0549M (+7.2%)

Notably, across all three diversification variants, the memory overhead for the controller remains consistent at 1.59%. Furthermore, the post-instrumented code size overhead is approximately 7%.

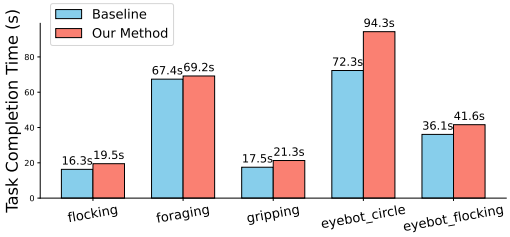


Figure 3: Evaluation on behavior diversification.

Behavior Diversification Overhead. Figure 3 presents the mission completion time for both before and after the adoption of behavior randomization. We observe that there is a 2% to 30% increase in mission completion time. This overhead largely hinges on the specific type of mission. For instance, in the *foraging* mission, which exhibits the least overhead, all robots collaboratively engage in foraging. Their behaviors in this context are inherently random. In contrast, for the *eyebot_circle* mission, the robot's tasks are more deliberate, defined as a sequence of behaviors in a set order. In such scenarios, randomized behavior is more prone to disrupt the mission's progress. A more in-depth exploration of this trade-off is reserved for future work.

7 DISCUSSION

Currently, individual behaviors are manually extracted, which is not scalable for more complex swarm algorithms. In the future, we aim to explore the possibility of automatically extracting behaviors directly from the source code. This could potentially be achieved through static analysis techniques that parse program semantics [18]. Furthermore, behavior randomization has the potential to modify the workload of a task, thus affecting its real-time performance. With this challenge in mind, our future work will focus on the co-design of behavior randomization and real-time adaptation. This can be achieved by utilizing timing analysis [13] to optimize randomization policy as well as system-level support for dynamic workload [6].

8 CONCLUSION

In this paper, we introduce a novel two-level diversification approach as a moving-target defense strategy to address vulnerabilities in swarm robotics systems. Our approach integrates both software and algorithm diversification. To mitigate potential unsafe behaviors from algorithm diversification, we utilize the social force model to predict and minimize collision risks. Our evaluations on swarm drones and vehicles demonstrate that the introduced diversification methods incur an acceptable level of performance overhead, rendering them suitable for practical deployment.

REFERENCES

[1] Amazon. How amazon is building its drone delivery system, Aug 2022.
[2] Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *2010 IEEE/RSJ IROS* (2010), IEEE, pp. 4187–4193.
[3] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence* (2013).
[4] Brooks, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation 2*, 1 (1986), 14–23.
[5] Eldefrawy, K., Rattanavipanon, N., Tsudik, G., and Carpent, X. Lightweight swarm attestation: a tale of two lisa-s.
[6] Gog, I., Kalra, S., Schafhalter, P., Gonzalez, J. E., and Stoica, I. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems* (2022), pp. 453–471.
[7] Helbing, D., and Molnar, P. Social force model for pedestrian dynamics. *Physical review E 51*, 5 (1995), 4282.
[8] Innovators, T. Drone swarm technology for search and rescue missions: Saving lives in emergency situations, May 2023.
[9] Jackson, T., Salamat, B., Homescu, A., Manivannan, K., Wagner, G., Gal, A., Brunthaler, S., Wimmer, C., and Franz, M. Compiler-generated software diversity. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats* (2011), 77–98.
[10] Jung, C., Ahad, A., Jeon, Y., and Kwon, Y. Swarmflawfinder: Discovering and exploiting logic flaws of swarm algorithms. In *2022 IEEE Symposium on Security and Privacy (SP)* (2022), pp. 1808–1825.
[11] Jung, C., Ahad, A., Jung, J., Elbaum, S., and Kwon, Y. 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* (New York, NY, USA, 2021), ESEC/FSE 2021, Association for Computing Machinery, p. 868–880.
[12] Larsen, P., Homescu, A., Brunthaler, S., and Franz, M. Sok: Automated software diversity. In *2014 IEEE Symposium on Security and Privacy* (2014), IEEE, pp. 276–291.
[13] Li, A., Liu, H., Wang, J., and Zhang, N. From timing variations to performance degradation: Understanding and mitigating the impact of software execution timing in slam. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), IEEE.
[14] Liu, W., and Winfield, A. F. Modeling and optimization of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research 29*, 14 (2010), 1743–1760.
[15] Luber, M., Stork, J. A., Tipaldi, G. D., and Arras, K. O. People tracking with human motion predictions from social forces. In *2010 IEEE international conference on robotics and automation* (2010), IEEE, pp. 464–469.
[16] mavros_swarm github. https://github.com/Jaeyoung-Lim/mavros_swarm. Accessed: 2023-07-07.
[17] Meier, L., Honegger, D., and Pollefeys, M. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)* (2015), IEEE.
[18] Merlo, E., Pinciroli, C., Panerati, J., Famelis, M., and Beltrame, G. Automated extraction and checking of property models from source code for robot swarms. In *Proceedings of the 4th International Workshop on Robotics Software Engineering* (2023), RoSE '22, p. 47–54.
[19] Ozmen, M. O., Farrukh, H., Kim, H., Bianchi, A., and Celik, Z. B. Short: Rethinking secure pairing in drone swarms.
[20] Petzi, L., Yahya, A. E. B., Dmitrienko, A., Tsudik, G., Prantl, T., and Kounev, S. SCRAPS: Scalable collective remote attestation for Pub-Sub IoT networks with untrusted proxy verifier. In *31st USENIX Security Symposium*.
[21] Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., et al. Argos: a modular, multi-engine simulator for heterogeneous swarm robotics. In *2011 IEEE/RSJ IROS* (2011), IEEE, pp. 5027–5034.
[22] Schiller, N., Chlosta, M., Schloegel, M., Bars, N., Eisenhofer, T., Scharnowski, T., Domke, F., Schönherr, L., and Holz, T. Drone security and the mysterious case of dji's droneid. In *Network and Distributed System Security Symposium (NDSS)* (2023).
[23] Yang, L., Yu, J., Yang, S., Wang, B., Nelson, B. J., and Zhang, L. A survey on swarm microrobotics. IEEE.
[24] Yao, Y. E., Dash, P., and Pattabiraman, K. Swarmfuzz: Discovering gps spoofing attacks in drone swarms. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2023), pp. 366–375.