Demo: Discovering Faulty Patches in Robotic Vehicle Control Software

Hyungsub Kim, Muslum Ozgur Ozmen, Z. Berkay Celik, Antonio Bianchi, and Dongyan Xu
Purdue University
{kim2956, mozmen, zcelik, antoniob, dxu}@purdue.edu

Video 1: https://youtu.be/TWK5lFPlLB4 **Video 2**: https://youtu.be/htnWzS4hoCs

This demo is based on PATCHVERIF [2], an automated patch verification framework for robotic vehicle (RV) software (e.g., ArduPilot and PX4). Faulty patches that we target occur for three main reasons: (*i*) partially fixing a bug, i.e., the buggy behavior still happens in another context (e.g., different location), (*ii*) patching an incorrect behavior but hurting another correct behavior, and (*iii*) adding new functionality but introducing a bug. The faulty patches can cause two types of bugs: (1) memory-safety bugs that violate memory access rules and (2) logic bugs that make the RV software behave incorrectly without any memory access violations [1], [3], [4]. PATCHVERIF mainly aims to discover logic bugs caused by faulty patches. Discovering such faulty patches is challenging, as there is (*i*) no clear definition of an RV's "correct behavior" and (*ii*) a huge "input space" of RV software.

We explain a faulty patch (Listing 1) discovered by PATCHVERIF. ArduPilot supports a pivot turn for vehicles. When a vehicle is near a corner, it decreases its speed, turns towards the next waypoint, and continues the navigation. The `desired_speed` at line 2 determines the vehicle's ground speed, and `g2.wp_nav.get_speed()` returns a limited speed so that the vehicle slows down before reaching a corner, as shown in Figure 1-(a). Yet, `g2.wp_nav.get_desired_speed()` at line 3 always returns the constant value set by `WP_SPEED` configuration parameter, even when the vehicle is near the corner. As a result, the vehicle deviates from the planned navigation path due to the high speed at the corner, as shown in Figure 1-(b). The deviated vehicle might collide with another vehicle in the opposite lane.

To discover such a faulty patch, PATCHVERIF conducts the following steps: (1) Identifying the RV's physical states that are affected by the patch and the environmental conditions that impact the patch by analyzing the removed/added code and developers' comments in the patch, e.g., the patch code in Listing 1 changes the RV's *speed*, and *wind* affects the patch (i.e., *speed*); (2) Inferring the inputs that affect the identified RV states and environmental conditions, e.g., `WP_SPEED` configuration parameter and `WIND_SPEED` environmental factor change the RV's *speed*; (3) Finding inputs triggering the patch code, e.g., adding waypoints and setting `AUTO` mode trigger the patch code in Listing 1; (4) Mutating the identified inputs to generate test cases that are highly likely to make the incorrectly patched software deviate from normal behaviors, e.g., PATCHVERIF assigns a large value to `WP_SPEED` to make the vehicle deviate from the planned navigation path due to the high speed at the corner, as shown in Figure 2a; (5) Concluding that the patch is faulty because the patched software shows

Symposium on Vehicles Security and Privacy (VehicleSec) 2023
27 February 2023, San Diego, CA, USA
ISBN 1-891562-88-6
https://dx.doi.org/10.14722/vehiclesec.2023.23015
www.ndss-symposium.org

```
1  void Mode::navigate_to_waypoint() {
2  -   float desired_speed = g2.wp_nav.get_speed();
3  +   float desired_speed = g2.wp_nav.get_desired_speed();
```

Listing 1: A faulty patch [5] removes line 2 and adds line 3.

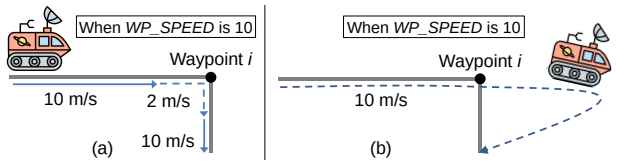
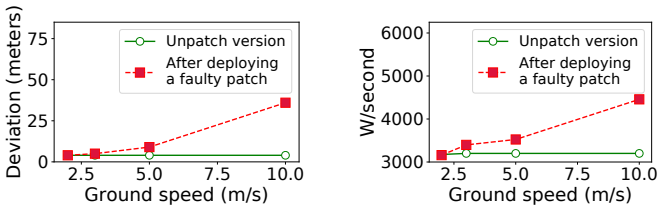


Fig. 1: (a) Normal behavior during a pivot turn and (b) abnormal behavior after deploying a faulty patch.



(a) Deviation from a path. (b) Battery consumption.

Fig. 2: Abnormal behaviors after deploying the faulty patch.

huge increases in position error and battery consumption, as shown in Figure 2.

We used PATCHVERIF to analyze 1,000 patches. As a result, PATCHVERIF discovered 115 previously unknown bugs.

Demonstration Plan. We provide videos to demonstrate two faulty patches discovered by PATCHVERIF. The first one is detailed above (Listing 1). The second one makes an RV navigate too close to a static object and eventually causes the RV to get stuck near the object. Our project website includes (1) details of the discovered 115 bugs and (2) previews of the demo videos: **https://github.com/purseclab/PatchVerif**.

REFERENCES

[1] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Policy-based Discovery and Patching of Logic Bugs in Robotic Vehicles," in *Proceedings of the Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, 2022.

[2] H. Kim, M. O. Ozmen, Z. B. Celik, A. Bianchi, and D. Xu, "PatchVerif: Discovering Faulty Patches in Robotic Vehicles," in *Proceedings of the USENIX Security Symposium*, 2023.

[3] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2021.

[4] H. Kim, M. O. Ozmen, Z. B. Celik, A. Bianchi, and D. Xu, "PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[5] "Fix speed nudge," https://tinyurl.com/x9hwzvfx, 2023.

ACKNOWLEDGMENT

This research is partially funded by the Secure Systems Research Center (SSRC) at the Technology Innovation Institute (TII) and ONR under Grants N00014-20-1-2128. The views expressed are those of the authors only.