Talk: GlueZilla: Efficient and Scalable Software to Hardware Binding using Rowhammer

Ruben Mechelinck[1][0000−0002−7597−6222], Daniel Dorfmeister[2][0000−0002−2718−6007], Bernhard Fischer[2][0000−0001−9737−0056], Stefan Brunthaler[3][0000−0001−9766−4871], and Stijn Volckaert[1][0000−0001−5594−7742]

[1] Distrinet, KU Leuven, Belgium
{ruben.mechelinck,stijn.volckaert}@kuleuven.be
[2] Software Competence Center Hagenberg, Austria
{daniel.dorfmeister,bernhard.fischer}@scch.at
[3] μCSRL, CODE Research Institute, University of the Bundeswehr Munich, Germany
brunthaler@unibw.de

Abstract. Industrial-scale reverse engineering affects the majority of companies in the mechanical and plant engineering sector and imposes significant economic damages.Reverse engineering mitigations try to increase the cost involved in reverse engineering until it surpasses the cost of actual development. Although these mitigations exist, economic damage has not been impacted, indicating that they have failed to address the problem.At present, most industrial-scale reverse engineering efforts are spent on replicating hardware components since software can often be copied verbatim without any reverse engineering effort.

In this talk, we discuss GlueZilla [6], our recently published system that binds software to hardware through user-space rowhammer PUFs on commodity hardware [5,4,7,2,1,8,3]. GlueZilla relies on unclonable machine features and thereby forces counterfeiters to reverse-engineer both the hardware and the software, driving up the reverse-engineering cost.

In GlueZilla, a program has two fully functional modes of operation. In the intentional mode, GlueZilla performs the expected operations as described by the original source code, whereas in the unintentional mode, the execution differs at unsuspicious-looking junction points. For example, the program could follow conditional branches in the wrong direction, or call different targets at call sites. The unintentional mode should not exhibit obvious signs that something is wrong with the program, e.g., program crashes.

The goal of GlueZilla is to only allow execution of the intentional mode on one selected associated machine instance. To this end, GlueZilla transforms the program at compile time to exhibit the unintended behavior by default. At run time, it uses targeted rowhammer-induced bit flips at the junction points to recreate the intentional execution mode in memory, as shown in Figure 1.

GlueZilla uses rowhammer because of its unique properties.Since the rowhammer-induced bit flip pattern is unclonable, GlueZilla ensures

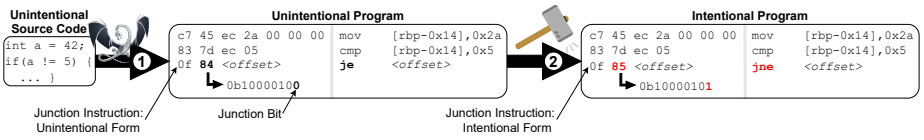

Fig. 1. At run time, GlueZilla uses rowhammer to transform junction instructions from their unintentional into their intentional form.

the intentional execution mode is only reconstructed on the associated machine. If the software runs on any other machine, including exact clones of the associated machine, the required bit flips are absent and the program remains in its unintentional mode. For the same reason, dynamic analyses are ineffective on cloned machines as the intended operations are not performed on cloned machines. Rowhammer, furthermore, allows for stealthy memory changes in the whole memory region without explicit write operations performed by the CPU. This eliminates various dynamic analysis techniques which typically rely on the CPU to intercept certain operations or code changes. Dynamic tools that modify the memory layout also interfere with GlueZilla as the junction points will no longer reside in the required rowhammer-susceptible memory locations. Additionally, the static binary is only an image of the unintentional program and lacks information about the code changes required to recreate the intentional code, rendering static binary analysis unprofitable.
The published version of GlueZilla has a few clear disadvantages. Numerous factors, such as temperature and chip aging, might undermine the reliability of bit flips. The current design does not tolerate unreliable bit flips because they might result in an incomplete transition to the intentional program form. Furthermore, Rowhammer can only flip bits in one direction, thus leaving the whole intentional program in memory throughout execution. This makes GlueZilla susceptible to memory snapshotting attacks. We will conclude this talk by discussing our ongoing work that aims to eliminate these weaknesses by using a micro-architectural attack that invalidates the in-memory copy of the program, whilst leaving its functionality intact.

Keywords: Rowhammer · Micro-architectural Attacks · Software Protection.

References

1. Cojocar, L., Razavi, K., Giuffrida, C., Bos, H.: Exploiting correcting codes: On the effectiveness of ECC memory against Rowhammer attacks. In: S&P (2019)
2. Jattke, P., van der Veen, V., Frigo, P., Gunter, S., Razavi, K.: Blacksmith: Scalable rowhammering in the frequency domain. In: S&P (2022)
3. Jattke, P., Wipfli, M., Solt, F., Marazzi, M., Bölcskei, M., Razavi, K.: ZenHammer: Rowhammer attacks on AMD Zen-based platforms. In: USENIX Security (2024)
4. Kim, J.S., et al.: Revisiting RowHammer: An experimental analysis of modern DRAM devices and mitigation techniques. In: ISCA (2020)

5. Kim, Y., et al.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ISCA (2014)
6. Mechelinck, R., et al.: Gluezilla: Efficient and scalable software to hardware binding using rowhammer. In: DIMVA (2024)
7. Orosa, L., et al.: A deeper look into RowHammer's sensitivities: Experimental analysis of real DRAM chips and implications on future attacks and defenses. In: MICRO (2021)
8. Schaller, A., Xiong, W., Anagnostopoulos, N.A., et al.: Intrinsic rowhammer PUFs: Leveraging the Rowhammer effect for improved security. In: HOST (2017)