MAZE: Towards Automated Heap Feng Shui

Hamidreza Sanaee 400200868

December 7, 2021

1 Introduction

Maze is an automated heap layout manipulation tool which is designed to avoid memory corruption vulnerabilities that can be exploited in specific heap layouts like heap overflow and use after free(UAF).

1.1 importance of AEG

The number of security vulnerabilities is increasing exceedingly and software vendors need to quickly evaluate the severity of these vulnerabilities and take allocate appropriate resources to fix them. An AEG system can demonstrate the severity of a bug by generating a working targeted exploit, which takes control of the program and executes a payload.

1.2 Problem Scope

Existing AEG solutions are effective at exploiting stack-based or format string vulnerabilities. Maze's target is to handle heap-based vulnerabilities which few of AEG solutions so far are effective on.To manipulate heap layouts, in general one has to find primitives that are able to interact with heap allocators first(Heap layout primitive analysis), and then assemble them in a specific way by (de)allocating objects of specific sizes in a specific order(Heap layout primitive assembly). MAZE is only applicable to event loop driven programs like two program which communicate with each other through TCP connection over and over again in a loop.

1.3 overview of MAZE

MAZE takes the program and proof of concept (PoC) as inputs, First step is to Extract primitives in them. Primitives are the building blocks for heap layout manipulation. Second step is to find out dependencies which these primitives have with each other. Third step is primitive semantic analysis, in this step one determines information like the number of heap (de)allocation, size of them and their target and etc. In tools like shrike and gollum this step has been done with random search and genetic algorithm which had flaws of their own like drop of success rate when facing unwanted (de)allocation in heap primitives. Maze uses a new algorithm called dig & fill which seems to address this issue. Then Maze will utilize heap primitives to manipulate PoC's layout (infered from the PoC info) to the expected layout and generate an exploit using a constraint solver.

2 Evaluation

In this part we explain Maze evaluation results.

Name	CTF	Vul Type	Final State
sword	PicoCTF '18	UAF	EIP hijack
hacknote	Pwnable.tw	UAF	EIP hijack
fheap	HCTF '16	UAF	EIP hijack
main	RHme3 CTF '17	UAF	Memory write
cat	ASIS Qual '18	Double free	Memory write
asvdb	ASIS Final '18	Double free	Memory leak
note3	ZCTF '16	Heap bof	Unlink attack
stkof	HITCON '14	Heap bof	Unlink attack
Secure-Key-Manager	SECCON '17	Heap bof	Unlink attack
RNote2	RCTF '17	Heap bof	Unlink attack
babyheap	RCTF '18	Off-by-one	Unlink attack
secret-of-my-heart	Pwnable.tw	Off-by-one	Unlink attack
Mem0	ASIS Final '18	Off-by-one	Unlink attack
quotes_list	FireShell '19	Off-by-one	Unlink attack
freenote	0CTF '15	Double free	Unlink attack
databank	Bsides Delhi	UAF	fastbin attack

Figure 1: CTF programs successfully processed by MAZE

As you can see in figure 1 first benchmark is the CTF benchmark, 23 programs were selected from ctf competition and Maze could hijack control flow for 5, leak arbitrary memory address information for 1, output exploitable memory layout for 10 of them.

Solution	Solve time(s)	Succ	POC analysis time(s)
Maze	100% in 68s	100%	922s
Shrike	25% in 300s, 60% in 3000+s	60%	Not Supported
Gollum	75% in 300s, 85% in 2500+s	85%	Not Supported

Figure 2: Evaluation results of different solutions on PHP

Target	Vulnerabilities	Average time(s)
Python	CVE-2007-4965, 2014-1912, Issue24105, 24095, 24094	100% in 118s
Perl	Issue132544, 130703, 130321, 129024, 129012	100% in 141s

Figure 3: Evaluation results on Python and Perl

We can see evaluation on real world programs (PHP, Python and Perl) in figure 2 and 3.

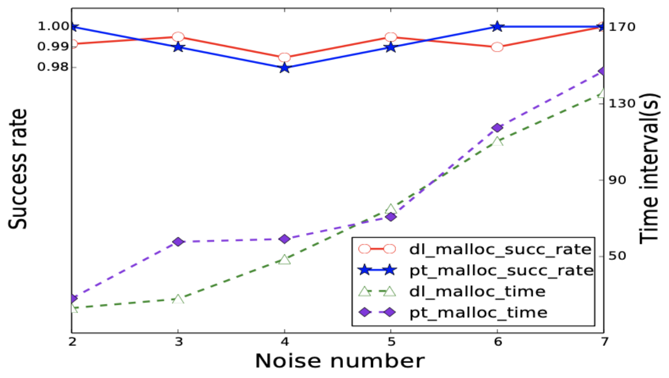

Figure 4: Influences of diffrent number of noises

Table 4 shows that success rate of Dig & Fill keeps between 98% and 100% for different number of noises which clarifies that the number of noises doesn't influence the success rate of Dig & Fill.

3 Conclusion

As stated in the paper Maze can transform PoC sample's heap layouts into expected layouts and automatically generate working exploits when possible, Maze could efficiently recognize and analyze heap layout primitives, Maze benefits of a novel Dig & Fill algorithm which assembles primitives to generate expected layout.