Mitigating Use-After-Free Attack using Library Considering Size and Number of Freed Memory

Authors names are omitted for double blind review

***Abstract*—Use-after-free (UAF) vulnerabilities, which are abused by exploiting a dangling pointer that refers to a freed memory, execute an arbitrary code. The vulnerability is caused by bug in a program. In particular, it is contained in a large scale program such as browser. HeapRevolver [1] [2], which prohibits freed memory area from being reused for a certain period, has been proposed. HeapRevolver in Windows uses the number of the freed memory areas for prohibiting as a trigger to release the freed memory area. In other words, HeapRevolver uses the number of the freed memory areas as a threshold for releasing. However, when the size of individual freed memory areas is large, the HeapRevolver on Windows increases the memory overhead. In this paper, we propose improved HeapRevolver for Windows considering the size and number of the freed memory areas. Improved HeapRevolver enables to prohibit the reuse of the certain number of the freed memory areas at any time via the size and number of the freed memory areas as a threshold. The evaluation results show that the improved HeapRevolver can prevent attacks that exploiting UAF vulnerabilities. In particular, when the size of individual freed memory areas is small in the programs, it is effective to decrease the attack success rate.**

***Index Terms*—Security, Use-After-Free, dangling pointer, memory allocation**

I. Introduction

There has been an increase in use-after-free (UAF) vulnerability. UAF vulnerability can be exploited by referring a dangling pointer to a freed memory. A UAF-attack abuses the dangling pointer that refers to a freed memory area and executes an arbitrary code by reusing the freed memory area. The number of UAF vulnerabilities based on the investigation in [3] is shown in Figure 1. The figure shows that the number of UAF vulnerabilities has rapidly increased since 2010 [3].

In particular, it is contained a lot of dangling pointer in a large scale program such as browser, and is frequently abused. One of the reasons is that a modern browsers have a JavaScript engine inside. When a UAF-attack succeeds, an attacker has to exploit the written attack code via dangling pointer. However, if ASLR (Address Space Layout Randomization) works, it is difficult to write an attack code in the target location. Therefore, an attacker uses JavaScript to bypass ASLR. It is because it is possible to increase an attack success rate by allocating and releasing object using JavaScript and handling the heap area indirectly.

In references [1] [2], HeapRevolver, which is a novel UAF-attack prevention method that delays and randomizes the release timing of a freed memory area via a memory-reuse-prohibited library, has been proposed. UAF-attack has a feature of reusing the target memory immediately after it is released.

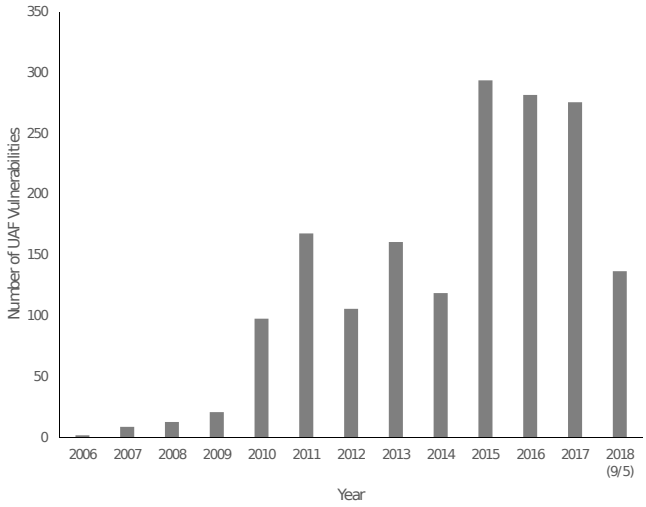

Fig. 1. Number of UAF vulnerabilities

The program applying HeapRevolver prevents reuse of the memory areas immediately after freed because it prohibits reuse of the freed memory areas for a certain period. In references [1] [2], HeapRevolver on Linux use the size of freed memory areas for prohibiting as a trigger to release the memory area. However, HeapRevolver on Windows used only the number of the freed memory areas for prohibiting as a trigger to release the memory area. Even when the number of the freed memory areas is used as a threshold, the entropy can increase and complicate UAF attacks using a large threshold and randomizing it. On the other hand, Windows does not limit the upper limit of individual memory areas size to 128 KB like Linux. Therefore, when the size of individual freed memory areas is large in the programs, there is a possibility that the memory overhead increases.

In this paper, we propose, in addition to the number of the freed memory area, the improved HeapRevolver on Windows which use the size of the freed memory areas as a threshold. Improved HeapRevolver releases the memory area for which reuse is prohibited when the total size of the freed memory areas for which reuse is prohibited is more than the size threshold. Release is performed in FIFO (First-In-First-Out) while the number of the freed memory areas prohibiting reuse is more than the number threshold and the total size of the freed memory areas is not less than the half of the size threshold. It is necessary to prohibit the reuse of a certain

number or more of the freed memory areas to decrease the attack success rate at any time. It is possible to prohibit the reuse of a certain number or more of the freed memory areas via the size and number thresholds at any time. Even if the total size of a small number of the freed memory areas is more than the size threshold because the size of individual freed memory areas is large, via the number threshold, it is possible to prohibit the reuse of the freed memory areas of more than a certain number at any time. In addition, even if the size of individual freed memory areas is small, via the total size of the freed memory areas as a threshold, it is effective to decrease the attack success rate because it is possible to prohibit the reuse of the freed memory areas of more than a certain number.

We describe the implementation of improved HeapRevolver and the results of evaluations. We evaluated attack success rate and memory overhead of browser applying improved HeapRevolver. The evaluation results show that improved HeapRevolver enables to prohibit more memory area than HeapRevolver using only the number threshold, and is effective to decrease the attack success rate.

The contributions of this paper are as follows:

1. We show the design and implementation of improved HeapRevolver on Windows. The implementation of improved HeapRevolver uses the total size of individual freed memory areas as a threshold in addition to the number threshold. Improved HeapRevolver obtain the size of a freed memory area via *HeapSize()* function.

2. The evaluation results show that improved HeapRevolver enables to prohibit more memory area than HeapRevolver using only the number threshold. Consequently it is effective to decrease the attack success rate.

II. Design of HeapRevolver

A. Concept of HeapRevolver

We describe HeapRevolver [1] [2], which prohibits the freed memory areas from being reused for a certain period. The freed memory area is immediately reused after the memory area is released to exploit the UAF vulnerabilities. It is because the probability of reusing the target memory area without being reused by another process increases. Therefore, HeapRevolver focuses on the timing of memory reuse in UAF vulnerability attack, thereby preventing UAF vulnerability attack by reusing memory for a certain period of time.

However, if the freed memory area is not reused, the overhead of creating new memory area will be increase. To solve this problem, HeapRevolver prohibit the reuse of the freed memory area for a certain period after it is freed.

B. Overview of HeapRevolver

UAF vulnerabilities immediately reuse freed memory area after the memory area is released. Therefore, HeapRevolver prohibit freed memory area from being reused for a certain period. The conditions for reuse are as follows:

Condition 1: The total size of the freed memory area is beyond the designated size.

Condition 2: The freed memory area is merged with an adjacent

When condition 1 is satisfied, the memory area that satisfies condition 2 is released. The released memory size is at most half of the designated total size in the freed memory. Condition 1 refers to technique used in DelayFree [5], can prevent the immediate reuse of the freed memory area immediately after it is freed. However, the designated total size (threshold) in the freed memory in these techniques is constant and the threshold is 100 KB. When an attacker creates a memory area of 100 KB, the freed memory is released. Thus, an attacker can attempt to reuse a memory area by creating a memory area. Hence, UAF attacks can be attempted. HeapRevolver develop two countermeasures for this problem. First, the total size threshold of the freed memory area is set to a larger value than that in DelayFree. Second, the threshold is randomized in the designated ranges. This measure increases the threshold entropy against UAF-attacks because threshold estimation becomes more difficult. In addition, HeapRevolver releases at most only half of the freed memory area. Furthermore, by adding condition 2, a UAF attack fails if an offset of a dangling pointer to the memory area is not appropriately calculated.

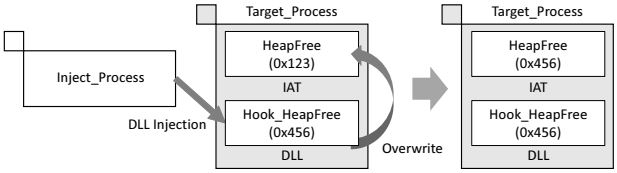

Fig. 2. Flow of hooking HeapFree() function on Windows

C. Implementation of HeapRevolver on Windows

We describe details of the realization method in Windows, because we propose improved HeapRevolver on Windows in this paper. In references [1] [2] and this paper, HeapRevolver is implemented using a dynamic link library (DLL) injection and API hook. Figure 2 shows the implementation of HeapRevolver on Windows. DLL injection is a DLL mapping method to other processes and executes DLL processing in the processes. Windows API hook is a method that hooks a Windows API call and executes a certain processing before the hooked Windows API call. HeapRevolver deployed an import address table (IAT) hook for the Windows API hook. The address of the API functions exported from DLL is stored in IAT during the loading-process time. IAT hook is a method that modifies the address of APIs in IAT to call a target function.

First, HeapRevolver maps *Hook.dll* that performs IAT hook to a target process by DLL injection. Next, *Hook.dll* overwrites the address of the *HeapFree()* function stored in IAT in the address of the *Hook_HeapFree()* function of *Hook.dll*. When the *Hook_HeapFree()* function of *Hook.dll* is called by IAT hook, the *Hook_HeapFree()* function of *Hook.dll* obtains the arguments of the *HeapFree()* function and stores them in a ring

buffer. Next, the *Hook_HeapFree()* function checks whether the number of the freed memory are beyond the threshold. If the number exceeds the threshold, the *Hook_HeapFree()* function obtains the arguments of the *HeapFree()* function and calls the *HeapFree()* function to release the freed memory area. The *Hook_HeapFree()* function calls the *HeapFree()* function until half of the threshold is released. If the number of the freed memory does not exceed the threshold, the proposed function returns without any operation. Thus, the *Hook_HeapFree()* function delays the release of the freed memory area until the number of the freed memory area exceeds the threshold.

However, the HeapRevolver implementation of Windows does not include the determination of whether or not a memory area is already merged with an adjacent memory area. This point needs to be further studied.

In references [1], [2], HeapRevolver on Windows did not manage the size of the released memory area, and use only the size threshold. In addition, even when the number of the freed memory areas is used as a threshold, the entropy can increase and complicate UAF attacks using a large threshold and randomizing it. Improved HeapRevolver explained in section III and later uses the total size as a threshold in addition to the number.

D. Problem of Previous HeapRevolver on Windows

HeapRevolver on Windows was realized and evaluated. On the other hand, the following problems exist in Windows realization. The problem is follows:

Problem 1: There is a possibility that the memory overhead increases.

On Linux, the upper limit of individual memory size is 128KB, but, on Windows, the upper limit is not limited to 128KB. On Windows, we investigated the size of individual freed memory areas that browser released when attack successes. Figure 3 shows the result of the investigation. We used Internet Explorer 10 (IE10) of Windows 7 (64bit) and attack code using CVE-2014-0322 distributed in Metasploit [6].

Figure 3 shows that about 70% of the freed memory areas from the IE10 was 128 bytes or less. On the other hand, memory areas exceeding 1 KB also existed. In particular, a maximum of 256KB per memory area was released. When the size of individual freed memory areas is large, HeapRevolver, which prohibit reuse of some freed memory areas for certain period, might increase the memory overhead. Therefore, we assumed that we have to consider the total size of the freed memory areas for which reuse is prohibited like HeapRevolver on Linux. On Windows, we can use the *HeapSize()* function of WindowsAPI to obtain the memory size.

Problem 2: There is a possibility that the reuse of a sufficient number of freed memory areas can not be prohibited.

Figure 4 shows relation between attack success rate and a threshold. This evaluation was done in reference [2], and used HeapRevolver which has only the number threshold. The

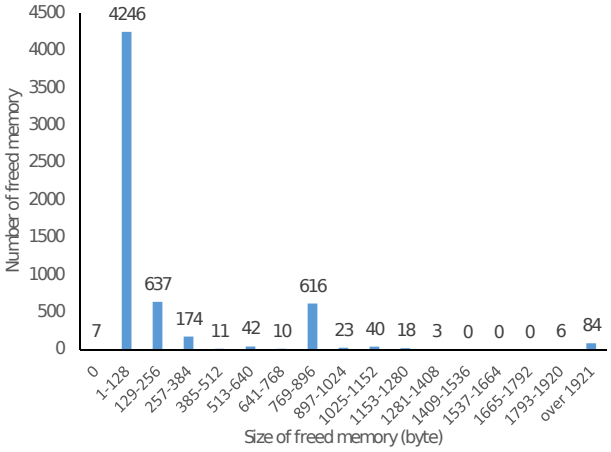

Fig. 3. Size of each freed memory area (CVE-2014-0322)

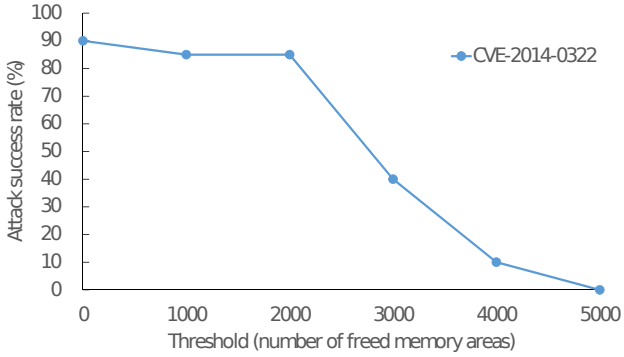

Fig. 4. Relation between attack success rate and a threshold [2]

attack code used in the environments exploited CVE-2014-0322 of IE10, and HeapRevolver were not applied to the browser when the threshold is zero.

Figure 4 shows that the attack success rate decrease according to the increase of threshold. This result indicates that it is effective to prohibit the reuse of the freed memory areas of more than a certain number to decrease attack success rate.

On the other hand, as explained above, the upper limit of individual memory size is not 128KB on Windows, and HeapRevolver has to consider the total size of the freed memory areas for which reuse is prohibited like HeapRevolver on Linux. However, if HeapRevolver on Windows uses only the total size of the freed memory areas for prohibiting as a trigger to release the memory area, the reuse of a sufficient number of the freed memory areas is not prohibited to prevent UAF-attack. It is because there is a possibility that the total size of a small number of the freed memory areas is more than the size threshold when the size of individual freed memory areas is large. Therefore, we assume that HeapRevolver has to use the number threshold because the number threshold enable to prohibit the reuse of the freed memory areas of more than a certain number at any time.

To cope with the abovementioned problems, in this paper,

HeapRevolver use the total size and number of the freed memory area as thresholds. We explained improved HeapRevolver in section III.

III. HeapRevolver considering size and number of the freed memory area

A. Concept of improved HeapRevolver on Windows

As mentioned in section II, our investigation shows that there is a possibility of releasing a large freed memory area. Owing to the possibility, on Windows, if HeapRevolver uses the number of the freed memory areas as a threshold to release the memory area and a size of individual freed memory areas is large, memory overhead will increase. On the other hand, on Windows, if HeapRevolver uses only the total size of the freed memory areas as a threshold to release the memory area, there is a problem that the reuse of a sufficient number of the freed memory areas is not prohibited to prevent UAF-attack.

To cope with the abovementioned problems, we propose improved HeapRevolver using the total size of the freed memory areas as a threshold in addition to the number. Owing to using the total size of the freed memory areas as a threshold in addition to the number, it is possible to prohibit reuse of the minimum number of memory areas at any time. In addition, improved HeapRevolver enables to prohibit more memory area than HeapRevolver using only the number threshold, and is effective to decrease the attack success rate.

B. Proceeding flow of HeapRevolver on Windows

Figure 5 shows proceeding flow of *Hook_HeapFree()* function of improved HeapRevolver which use the size and number of the freed memory areas as a threshold.

1) *Hook_HeapFree()* function is called.
2) Via *HeapSize()* function, the size of memory area for which reuse is prohibited can be obtained. In addition, update variables *HeapSizeTotal*, which is the total size of memory areas for which reuse is prohibited, *HeapNumTotal*, which is the number of memory areas for which reuse is prohibited.
3) Enqueue arguments to the queue. The queue is realized with a ring buffer.
4) Check whether the total size of the freed memory area for which reuse is prohibited is not less than the size threshold, which is named *size_threshold*.
   (A) If the total size of the memory areas for which reuse is prohibited is not less than the *size_threshold*; the process proceeds to process 5).
   (B) If the total size of the memory area for which reuse is prohibited is less than the *size_threshold*; the processing of the *Hook_HeapFree()* is finished.
5) Check whether the memory area for which reuse is prohibited is more than half *size_threshold*.
   (A) If it is more than half the *size_threshold*; the process proceeds to process 6).
   (B) If it is not more than half the *size_threshold*; the process proceeds to process 10).

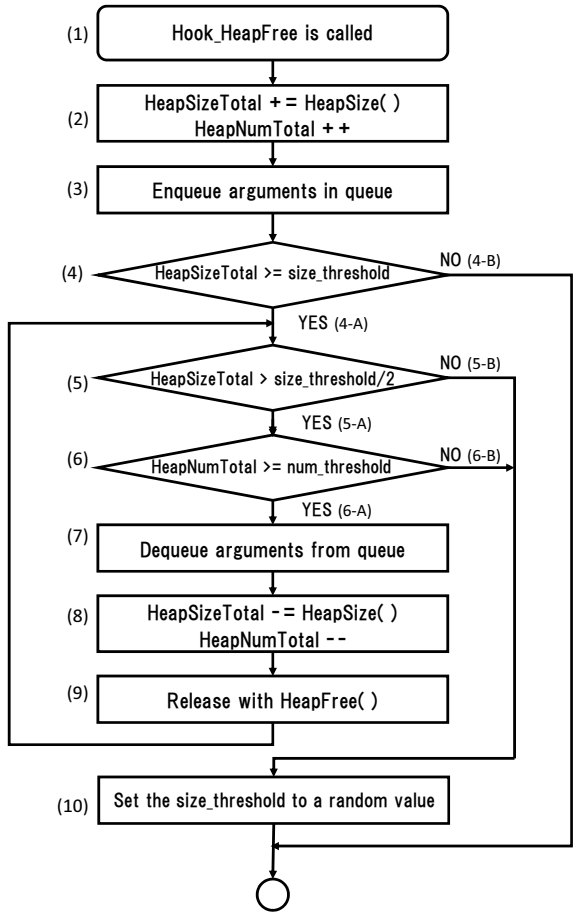

Fig. 5. Proceeding flow of *Hook_HeapFree* function

6) Check whether the number of memory areas prohibiting reuse is not less than the number thresholds, which is named *num_threshold*.
   (A) If the number of memory areas for which reuse is prohibited is not less than the *num_threshold*; the process proceeds to process 7).
   (B) If the number of memory areas for which reuse is prohibited is less than the *num_threshold*; the process proceeds to process 10).
7) Dequeue arguments from the queue.
8) Obtain size of the memory area which is released; update the variables *HeapSizeTotal* and *HeapNumTotal*.
9) Release the memory area of the dequeued arguments via the original *HeapFree()*; the process proceeds to process 5).
10) Set the size threshold to a random value within the specified range; finish the processing of the *Hook_HeapFree()*.

TABLE I
EVALUATION OF RANDOMIZED THRESHOLD (CVE-2014-0322)

Threshold		
Number	Range of size	Attack success rate
Without improved HeapRevolver		90%
500	100 KB–500 KB	80%
	500 KB–1 MB	15%
	1 MB–1.5 MB	0%
1,000	100 KB–500 KB	80%
	500 KB–1 MB	35%
	1 MB–1.5 MB	0%
1,500	100 KB–500 KB	80%
	500 KB–1 MB	25%
	1 MB–1.5 MB	0%
2,000	100 KB–500 KB	70%
	500 KB–1 MB	20%
	1 MB–1.5 MB	0%
2,500	100 KB–500 KB	0%
	500 KB–1 MB	0%
	1 MB–1.5 MB	0%

IV. EVALUATION

A. Evaluation Environment

We used a computer with Intel Core i5-4590 (3.30 GHz) and 8 GB main memory and Core i7-3770 (3.40 GHz) and 4 GB main memory for the evaluation. OSs and versions used in the evaluations are Windows 8 (64 bit) and Windows 7 (64bit). To show the attack success rate and memory overhead of the improved HeapRevolver, the following experiments were performed: we evaluated the attack success rate against actual UAF vulnerability attack using HeapRevolver with the size of the thresholds variously. In addition, we evaluated the memory overhead using three browser benchmarks [7]–[9] in order to evaluate the performance in the browser which is the main target in the real world.

B. Evaluation of HeapRevolver

1) Evaluation of Success rate: We evaluated using Internet Explorer 10 (IE10) on Windows 8 (64bit). Various size thresholds were used. We experimented on whether or not UAF attacks using real attack codes distributed in Metasploit [6] could be prevented. We tried 20 times for each threshold, and measured the attack success rate. We evaluated the attack success rate by comparing with the state applied improved HeapRevolver and not. The number threshold used for the evaluation is 500, 1,000 1,500, 2,000 and 2,500. These number are half of each threshold of HeapRevolver used in the evaluation of [2]. The size threshold was randomized, respectively, in the range of 100 KB–500 KB, 500 KB–1 MB, and 1 MB–1.5 MB. The evaluation results are shown in Table I.

As shown in Table I, when the range of size threshold is 100 KB–500 KB and the number threshold is 2,000 or less, the attack success rate is high. However, when the range of size threshold is 100 KB–500 KB and the number threshold is 2,500, the attack success rate is 0%. Although the number threshold is possible to prohibit reuse of more than the number thresholds, the range of 100 KB–500 KB is impossible to prohibit reuse of more than the number thresholds. Therefore, the range of 100 KB–500 KB is insufficient to prevent UAF-attack. It is clear that it is important to prohibit reuse of sufficient number of the freed memory areas via the number threshold. It is because decreasing attack success rate is more important than suppression of memory overhead

When the range of size threshold is 500 KB–1 MB and number threshold is at the most 2,000, attack success rate can be suppressed. It is estimate that sufficient number of the freed memory areas are prohibited reuse in many time when the range of size threshold is 500 KB–1 MB.

Compare with number threshold and attack success rate in the range of 100 KB–500 KB, it is estimate that the number of prohibiting reuse of the freed memory areas is around 2,500 when the range of size threshold is 1 MB–5 MB.

For the abovementioned, the number threshold is important to decrease attack success rate. In addition, it is found that there are cases where it is possible to achieve both decreasing attack success rate and suppressing memory overhead via the size threshold. It is possible to keep all the freed memory area from being released via the number thresholds when memory area of larger than threshold is released.

2) Evaluation of Memory consumption: We performed an experiment to evaluate the memory consumption of HeapRevolver on Windows via browser benchmark with IE11 of Windows 7. We used Performance Monitor, which is preinstalled on Windows, to measure virtual memory usage of browser process. In this evaluation, the number threshold used is 1,500 and 2,500. In addition, these number thresholds have three different size thresholds, which are the randomizing range of 100 KB–500 KB, 500 KB–1 MB and 1 MB–1.5 MB. We ran three types of browser benchmark, namely, Octane, SunSpider, and Kraken. We measured the memory usage of the browser and calculated the maximum value and the average value of the memory usage during execution of the browser benchmark. Using the measurement results, compared with applying improved HeapRevolver and not, we evaluated memory overhead for each threshold. Table II shows evaluation results. Values in parentheses in the table are overhead values.

As shown in Table II, Octane was a small memory overhead. It was found from this result that even if you prohibit the reuse of 2,500 or more memory areas which equal to the number threshold, memory overhead is not less than range of the size threshold.

On the other hand, SunSpider and Kraken were a large memory overhead. It estimates that two reasons are exist. First, the total size of the memory area for which reuse was prohibited exceeded the range of the size thresholds. Second, the size of individual freed memory areas is larger than Octane one. Therefore, the total size of the freed memory area for which reuse was prohibited was increased, and the memory overhead was increased.

TABLE II
MAXIMUM AND AVERAGE MEMORY CONSUMPTION ON BROWSER BENCHMARKS

		Octane	SunSpider		Kraken	
		max	average	max	average	max
Without improved HeapRevolver		633.49 MB	103.31 MB	113.54 MB	213.95 MB	364.43 MB
1,500	100 KB-500 KB	638.92 MB (0.86%)	109.25 MB (5.76%)	122.21 MB (7.64%)	314.44 MB (46.96%)	664.03 MB (82.21%)
	500 KB-1 MB	622.05 MB (-1.81%)	108.31 MB (4.84%)	119.12 MB (4.92%)	317.19 MB (48.25%)	687.97 MB (88.78%)
	1 MB-1.5 MB	641.57 MB (1.28%)	111.35 MB (7.79%)	136.16 MB (19.93%)	309.55 MB (44.68%)	774.91 MB (112.64%)
2,500	100 KB-500 KB	650.42 MB (2.67%)	111.30 MB (7.74%)	127.14 MB (11.98%)	336.56 MB (57.30%)	719.22 MB (97.36%)
	500 KB-1 MB	622.05 MB (-1.81%)	112.00 MB (8.41%)	128.45 MB (13.14%)	321.45 MB (50.24%)	722.17 MB (98.17%)
	1 MB-1.5 MB	625.40 MB (-1.28%)	110.23 MB (6.70%)	124.64 MB (9.78%)	326.52 MB (52.61%)	735.15 MB (101.73%)

For these results, when the size of individual freed memory areas is small in the programs, the size threshold enables to prohibit more memory area than using only the number threshold. Therefore, it is effective to decrease the attack success rate. In addition, when the size of individual freed memory areas is large in the programs, it is difficult to decrease memory overhead. However, via the size threshold, it is a possible to keep the sufficient memory areas, which decrease the attack success rate, from being release.

V. RELATED WORK

References [10]–[16] were methods to prevent UAF-attacks. Reference [10] detects dangling pointers via static analysis and dynamic symbolic execution. References [11] and [12] add codes that detect dangling pointers in a compilation and detect UAF attacks in runtime. Reference [13] detects UAF vulnerabilities via machine learning and Typestate analysis. These approaches [10]–[13] require a source code.

References [14] and [15] shows a method that prevents VTable hijacking, which is often used for UAF-attack. By rewriting the pointer to VTable, attackers succeed UAF-attack. In order to cope with VTable hijacking, reference [14] place VTables in read-only memory, and reference [15] pin all the freed VTable pointers on a safe VTable under VTPin's control. These approaches change the VTable pointer to call only safe objects because most UAF attacks rewrite the pointer stored in VTable. However, these methods cannot handle a UAF attack that does not alter VTable. In addition, rewriting the binary of a target program beforehand is required.

Reference [16] was prevented using a method that alters library which randomizes the location of the allocated memory area. This method manages 4 free lists for each size. Freed objects were managed by 4 free lists, and released in FIFO (First-In-First-OUT). Since this method uses free lists, even if the number of managing objects increase, allocation costs fits within a certain time. On the other hand, owing to using free list and four free lists, the memory overhead increases.

VI. CONCLUSION

This paper proposed improved HeapRevolver on Windows considering the size and number of freed memory areas. In addition, we described its design and implementation. The *HeapSize()* function is used to obtain the memory size so that improved HeapRevolver can manage the total size of the freed memory areas. Therefore, improved HeapRevolver uses the size of freed memory areas as thresholds in addition to the number threshold. Consequently improved HeapRevolver enables to prohibit more memory area than HeapRevolver using only the number threshold, and is effective to decrease the attack success rate.

The evaluation of the attack success rate indicated that the number threshold was important to decrease attack success rate. In addition, the evaluation of memory consumption indicated that memory overhead became large when a size of individual memory area is large.

From these results, when the size of individual freed memory areas is large in the programs, improved HeapRevolver is difficult to decrease memory overhead. On the other hand, when the size of individual freed memory areas is small in the programs, improved HeapRevolver is effective to decrease the attack success rate.

REFERENCES

[1] Yamauchi, T. and Ikegami, Y., "HeapRevolver: Delaying and Randomizing Timing of Release of Freed Memory Area to Prevent Use-After-Free Attacks," The 10th International Conference on Network and System Security (NSS 2016), Lecture Notes in Computer Science (LNCS), Vol.9955, pp.219-234 (9, 2016). DOI:10.1007/978-3-319-46298-1_15

[2] Yamauchi, T., Ikegami, Y. and Ban, Y., "Mitigating Use-After-Free Attacks Using Memory-Reuse-Prohibited Library," IEICE Transactions on Information and Systems, volume E100 D, pp.2295–2306 (2017). DOI:10.1587/transinf.2016INP0020

[3] Common vulnerabilities and exposures, https://cve.mitre.org/index.html

[4] Daniel, M., Honoroff, J. and Miller, C., "Engineering Heap Overflow Exploits with JavaScript," In Proc. USENIX Workshop on Offensive Technologies (WOOT), 2008.

[5] Tang, J. "Mitigating uaf exploits with delay free for internet explorer," http://blog.trendmicro.com/trendlabs-security-intelligence/mitigating-uaf-exploits-with-delay-free-for-internet-explorer/

[6] Metasploit, http://www.metasploit.com/

[7] Octane 2.0, http://octane-benchmark.googlecode.com/svn/latest/index.html

[8] SunSpider 1.0.2 JavaScript Benchmark, https://www.webkit.org/perf/sunspider/sunspider.html

[9] Kraken JavaScript Benchmark (version 1.1), http://krakenbenchmark.mozilla.org/

[10] Josselin, F., Laurent, M., Sbastien, B., et al., "Finding the needle in the heap: combining static analysis and dynamic symbolic execution to trigger use-after-free," Proc. 6th Workshop on Software Security, Protection, and Reverse Engineering (SSPREW'16), Article No.2, 2016.

[11] Van der Kouwe, E., Nigade, V. and Giuffrida, C., "DangSan: Scalable Use-after-free Detection," Proc. Twelfth European Conference on Computer Systems, (EuroSys '17), pp.405–419, 2017.

[12] Younan, Y., "FreeSentry: Protecting Against Use-After-Free Vulnerabilities Due to Dangling Pointers," in the 2015 Network and Distributed System Security Symposium (NDSS'15), 2015.

[13] Yan, H., Sui, Y., Chen, S., et al., "Machine-Learning-Guided Typestate Analysis for Static Use-After-Free Detection," Proc. 33rd Annual Computer Security Applications Conference (ACSAC '17), pp.42–54, 2017.
[14] Chao, Z., Dawn, S., Scott, A.C et al., "VTrust: Regaining Trust on Virtual Calls," in the 2016 Network and Distributed System Security Symposium (NDSS'16), 2016.
[15] Pawel, S., Vasileios, P.K and Christiano, G., "VTPin: Practical VTable Hijacking Protection for Binaries," Proc. 32nd Annual Computer Security Applications Conference (ACSAC '16), pp.448–459, 2016.
[16] Silvestro, S., Liu, H., Crosser, C., et al., "FreeGuard: A Faster Secure Heap Allocator," Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS ' 17), pp.2389–2403, 2017.