Computer Security Symposium 2017
23 - 25 October 2017

Windows における Use-After-Free 脆弱性攻撃防止手法

伴 侑弥[1]　山内利宏[1,a)]

概要：近年，Use-After-Free(UAF) 脆弱性攻撃が増加している．UAF 脆弱性攻撃では，解放後のメモリ領域を参照するダングリングポインタを悪用し，任意のコード実行が可能である．そこで，我々は，脆弱性を悪用した攻撃への対策として，解放されたメモリ領域の再利用を一定期間禁止することで，UAF 脆弱性攻撃を防止する手法を提案した．Windows で実現した提案手法は，再利用を禁止したメモリ領域の解放契機に再利用を禁止しているメモリ領域の個数を閾値として用いていた．本稿では，解放契機に再利用を禁止しているメモリ領域の合計サイズも閾値として導入した提案手法について述べ，評価を行った結果を報告する．

キーワード：Use-After-Free，メモリ再利用禁止，ライブラリ，動的メモリ確保

Yuya Ban[1]　Toshihiro Yamauchi[1,a)]

1　はじめに

Use-After-Free (UAF) 脆弱性攻撃が増加している．UAF 脆弱性攻撃では，解放後のメモリ領域を参照するダングリングポインタを悪用しており，任意のコード実行が可能である．**図 1** に，CVE [1] の脆弱性データベースに登録されている UAF 脆弱性の件数を示す．図 1 より，2010 年以降，UAF 脆弱性の件数が急激に増加していることが確認できる．

特に，ブラウザのような大規模プログラムは，ダングリングポインタが多く存在し，UAF 脆弱性が頻繁に悪用されている．ブラウザが内部に JavaScript エンジンを持っていることも悪用される理由の一つである [2]．攻撃者が UAF 脆弱性攻撃を達成する場合，書き込んだ攻撃コードをダングリングポインタを用いて実行する必要がある．しかし，ASLR (Address Space Layout Randomization) が有効である場合，ターゲットとなる場所に攻撃コードを書き込むことは困難である．そこで，ASLR を回避するために攻撃者は，JavaScript を利用することが多い．JavaScript を用いてオブジェクトの確保と解放を行うことで，間接的にヒープ領域を操作し，攻撃成功率を高めることが可能である．

我々は，解放されたメモリ領域の再利用を一定期間禁止することで，UAF 脆弱性攻撃を防止する手法 (以降，提案手法) を提案した ([3]–[5])．UAF 脆弱性攻撃は，オブジェクトのメモリ領域を解放した直後に，その解放したメモリ領域を再利用する特徴がある．提案手法を適用すると，解放されたメモリ領域の再利用を一定期間禁止するため，解放直後のメモリ領域の再利用を防ぐことができる．また，再利用を可能にするまでの期間をランダムにすることで，メモリ領域の再利用が可能になるタイミングの推測をより難しくする．これによって，UAF 脆弱性攻撃を高い確率で防止できる．さらに，提案手法は保護対象のプログラムを改変せずに適用できるため，プログラムへの適用は比較的容易である．

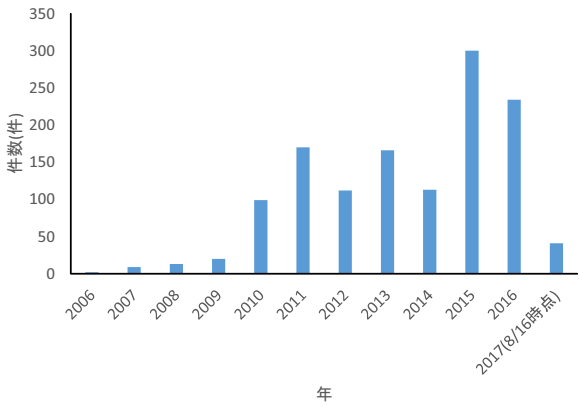

図 1　CVE に登録されている UAF 脆弱性の件数

文献 [3]–[6] において，Windows における提案手法の実

[1] 岡山大学 大学院自然科学研究科
[a)] yamauchi@cs.okayama-u.ac.jp

装方法は，再利用を禁止しているメモリ領域の解放契機に再利用を禁止しているメモリ領域の個数を用いて評価した．そこで，本稿では，Linux と同様にサイズを閾値とする手法を Windows に適用する．Windows は Linux のように個々のメモリサイズが 128KB に制限されないため，そのまま適用すると，再利用を禁止するメモリが一個となり，簡単に攻撃できる．そこで，新たに，サイズに加え，メモリ個数を閾値として用いる手法を提案する．また，提案手法について評価を行った結果を述べる．

2 Use-After-Free 脆弱性攻撃防止手法

2.1 UAF 脆弱性攻撃

図 2 に，文献 [3] で示した例を示し，説明する．UAF 脆弱性は，プログラムのバグにより，オブジェクトのメモリ領域を解放した直後に, 再度ヒープメモリアドレスが参照できてしまう脆弱性である．この UAF 脆弱性を用いることで，任意のコード実行が可能である．解放されたヒープメモリアドレスを参照するポインタは，ダングリングポンタと呼ばれる．任意のコードを実行するためには，ダングリングポインタの参照先に任意コード実行のためのコードを書き込む必要がある．メモリ領域が解放された直後に解放されたメモリ領域と同等サイズのメモリ領域を確保することで，同じメモリ領域が再利用される確率が高くなる．このため，攻撃コードでは，メモリ領域が解放された直後に再利用を行う．ダングリングポインタを参照した際に，再利用したメモリ領域が任意コードを実行する様にコードの書き込みを行うことで，任意のコード実行が可能になる．図 2 の場合，ダングリングポインタの参照により，Addnum オブジェクトの仮想関数の実行と同様の手順で攻撃コードが実行されるため，この手順に倣ったコードを再利用したメモリ領域に書き込む．図 2 の場合の，メモリレイアウトを**図 3** に示す．

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
using namespace std;

class Addnum
{
  int num1, num2, result;
public:
  Addnum(char *arg1, char *arg2) {
    num1 = atoi(arg1);
    num2 = atoi(arg2);
    result = num1 + num2;
  }
  virtual void print() {
    printf("result = %d\n", result);
  }
};

int main(int argc, char *argv[])
{
  Addnum *addnum = new Addnum(argv[1], argv[2]);
  addnum->print();
  delete(addnum);

  char shellcode[] = "\x48\x31\xd2\x52\x48\xb8\
      x2f\x62\x69\x6e\x2f\x2f\x73\x68\x50\x48\
      x89\xe7\x52\x57\x48\x89\xe6\x48\x8d\x42\
      x3b\x0f\x05";

  char *buf = new char[24];

  memcpy(buf,"\x18\x20\x60\x00\x00\x00\x00\x00\
      x40\xe2\xff\xff\xff\x7f\x00\x00",16);

  addnum->print(); // dangling pointer

  delete[] buf;
  return 0;
}
```

図 2 UAF 脆弱性を含むサンプルコード

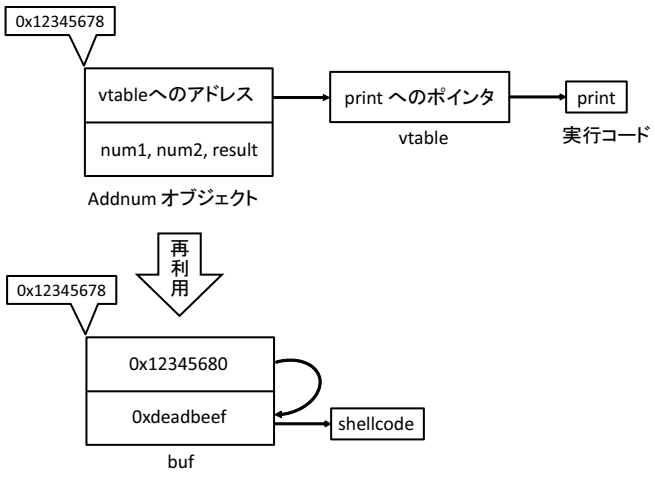

図 3 UAF 攻撃の際のメモリレイアウト

2.2 メモリ再利用禁止による Use-After-Free 脆弱性攻撃防止手法

2.2.1 考え方

文献 [3] で提案したメモリ再利用禁止による Use-After-Free 脆弱性攻撃防止手法について説明する．UAF 脆弱性攻撃におけるメモリ再利用のタイミングは，オブジェクトの解放直後である．解放直後に再利用する理由は，解放した領域を別の処理に再利用させずに攻撃コードを書き込む領域として再利用する確率を上げるためである．そこで，提案手法は，UAF 脆弱性攻撃におけるメモリ再利用のタイミングはオブジェクトの解放直後であることに着目し，一定期間メモリを再利用させないことで UAF 脆弱性攻撃を防止する．

しかし，メモリ領域を再利用させない場合，新たなメモリ領域を確保する処理が必要となるため，オーバヘッドが増加する．この問題に対処するため，提案手法では，解放した領域の再利用禁止を一定期間のみとしている．

2.2.2 実現方式

UAF 脆弱性攻撃は，オブジェクトの解放直後にメモリを再利用するため，提案手法は，解放したメモリ領域の再利用を一定期間禁止する．再利用を可能とする条件を以下に示す．

(条件 1) 解放したメモリ領域の合計サイズが閾値以上である．

(条件 2) 解放したメモリ領域が前後の領域と結合している．

(条件 1) を満たしたとき，(条件 2) を満たすメモリ領域を最大で指定した閾値の半分までを解放する．(条件 1) は，DeleyFree [7] と同様の条件であり，メモリの解放直後にその解放したメモリ領域を再利用されることを防ぐこと

が可能である．しかし，DeleyFree は，閾値の設定が 100 KB で固定されているため，閾値で設定したサイズのメモリを確保され，解放された場合，UAF 攻撃を達成される可能性がある．そこで，提案手法では，閾値に設定する合計サイズを大きくすることでエントロピーを増加させ，攻撃を困難にしている．また，閾値を指定する範囲内でランダム化することで，より UAF 脆弱性攻撃を困難にする．さらに，解放したメモリ領域の半分までを再利用可能にすることで，一部のメモリ領域の再利用を遅らせ，より UAF 攻撃の達成を困難にする．

これに加え，(条件 2) を追加する．これにより，ダングリングポインタを参照するメモリ領域までのオフセットを算出しなければ UAF 攻撃は成功しないようにし，UAF 攻撃をさらに困難にしている．

2.2.3 Windows における提案手法の実現方式

文献 [4][5][6] の評価に用いた提案手法は，DLL (Dynamic Link Library) インジェクションと Windows API フックにより実現されている．**図 4** に提案手法の実現方式を示す．DLL インジェクションとは他プロセスに DLL をマッピングし，マッピングした DLL の処理を他プロセス内で実行させる手法である．Windows API フックとは，Windows API の呼び出しをフックし，任意の処理を実行させる手法である．Windows API フックには，IAT (Import Address Table) フックを使用している．IAT には，プロセスのロード時に，DLL からインポートされる API の関数のアドレスが格納される．IAT フックは，IAT に格納された API の関数のアドレスを任意の関数のアドレスに書き換えることで，API をフックする手法である．

提案手法は，まず，DLL インジェクションにより，IAT フックを実施する DLL (以降，Hook.dll) をターゲットプロセスにマッピングする．Hook.dll は，IAT に格納されている HeapFree 関数のアドレスを自身が用意した Hook_HeapFree 関数のアドレスに書き換える．書き換えた先の Hook_HeapFree 関数では，HeapFree 関数の引数を取得し，リングバッファに引数を格納する．その後，解放したメモリ領域の合計値が閾値以上になった場合，リングバッファから引数を取り出し，本来の HeapFree 関数を呼び出し，解放したメモリの半分までを再利用可能とする．

我々はこの手法を Windows で実現した．ただし，前後のメモリ領域と結合したかの判定は未実装であり，残された課題となっている．

また，文献 [4][5][6] で用いた提案手法では，解放したメモリ領域のサイズ取得は，処理が煩雑になるため行っておらず，代わりに解放したメモリ領域の個数を閾値として利用している．個数を閾値として利用する場合でも，合計サイズで閾値を設定する場合と同様に，個数を大きくすることやランダム化することで，エントロピーが増大し，攻撃を困難にできると我々は考えている．

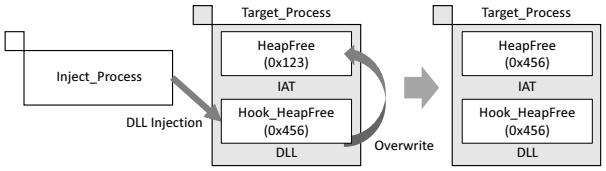

図 4 Windows における提案手法の実現方式

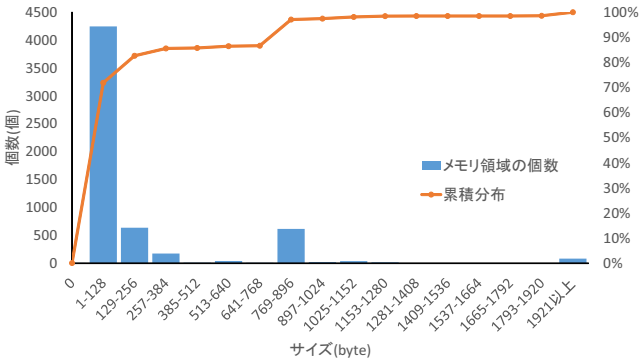

図 5 解放されるメモリ領域のサイズ (CVE-2014-0322)

3 メモリ領域の個数とサイズを考慮した提案方式

3.1 ブラウザが解放するメモリ領域のサイズ調査

文献 [6] では，個数を閾値に使用していたが，本稿ではサイズを閾値に使用した提案手法を Windows で実現する．提案手法が再利用を禁止する個々のメモリ領域のサイズは Windows API の HeapSize 関数を使用することで，取得が可能である．このため，HeapSize 関数を使用して，個々のメモリ領域のサイズから再利用を禁止しているメモリ領域の合計サイズを算出して用いる．

そこで，実際にブラウザが攻撃コードを読み込み，攻撃が成功する際にブラウザが解放する個々のメモリ領域のサイズを把握するため，調査を行った．調査結果を**図 5** に示す．調査には，Windows 8(64bit) の Internet Explorer 10(IE10) と， Metasploit[8] で配布されている CVE-2014-0322 を使用する攻撃コードを用いて調査を行った．

図 5 より，CVE-2014-0322 の攻撃コードを読み込み，攻撃が成功する際に IE10 が解放する個々のメモリサイズの約 70%が 128byte 以下である．一方で，1KB を超えるサイズのメモリも存在した．具体的には，最大一つで 256KB のメモリ領域が解放されていた．

3.2 考え方

3.1 節の評価より，大きなサイズのメモリ領域が解放される可能性があることがわかった．このため，サイズのみを閾値に使用した提案手法の場合，閾値を超えるサイズのメモリ領域を再利用禁止にすると，他の再利用を禁止しているメモリ領域が解放される問題がある．この問題に対処するため，サイズだけでなく個数も考慮した手法を提案する．

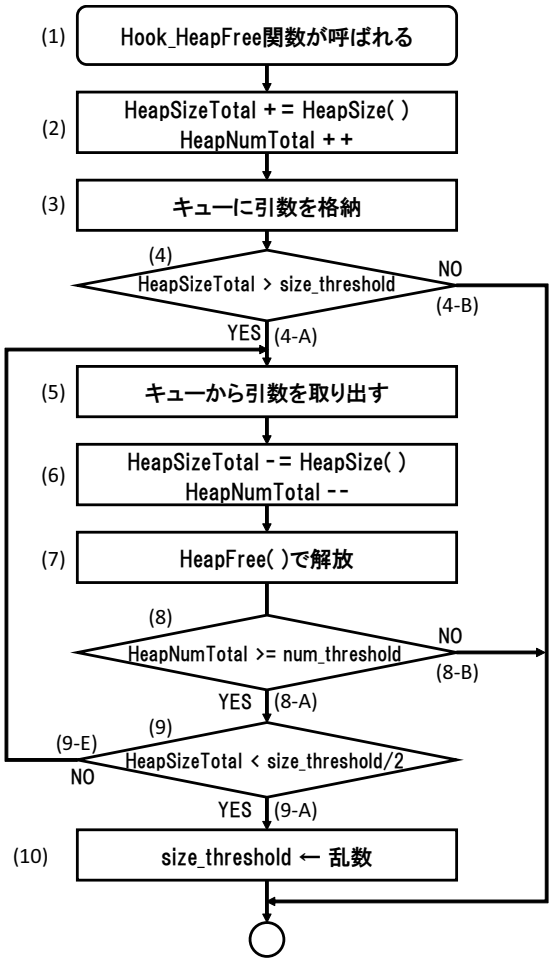

図 6 再利用を禁止しているメモリ領域の個数とサイズを考慮した処理の流れ

合計サイズのみをメモリ領域を解放する閾値として用いた場合，閾値よりも大きなサイズのメモリ領域が再利用禁止された場合，再利用を禁止している全てのメモリ領域が解放される．このため，大きなサイズのメモリ領域を再利用禁止しても一定期間解放を行わないようにする必要がある．合計サイズに加え，個数を閾値として併用することで，常に個数の閾値以上のメモリ領域を再利用禁止にしておくことができる．これにより，解放されたメモリ領域のサイズが指定した閾値以上であった場合でも，個数の閾値を指定しておくことで，すべてのメモリ領域が解放されることを防ぐことが可能である．

3.3 メモリ領域の個数とサイズを考慮した提案方式の詳細

メモリ領域を解放する閾値にメモリ領域の個数とサイズを併用した提案方式における Hook_HeapFree 関数の処理流れを図 **6** に示し，以下で説明する．

(1) Hook_HeapFree 関数が呼ばれる．
(2) HeapSize 関数を用いて，再利用を禁止するメモリ領域のサイズを取得．変数 HeapSizeTotal(再利用を禁止しているメモリ領域の合計サイズ) と変数 HeapNumTotal(再利用を禁止しているメモリ領域の個数) を更新する．
(3) キューに引数 A を格納する．キューはリングバッファで実現している．
(4) 再利用を禁止しているメモリ領域が 閾値の size_threshold (合計サイズの閾値) より大きいか確認する．
  (A) 再利用を禁止しているメモリ領域が合計サイズの閾値を超えていた場合，処理 (5) へ進む．
  (B) 再利用を禁止しているメモリ領域が合計サイズの閾値を超えていない場合，Hook_HeapFree 関数の処理を終える．
(5) 配列から再利用を禁止している期間が最も長い引数から順番に取り出す．配列はリングバッファで実現している．
(6) 取り出した引数のメモリ領域のサイズを取得し，変数 HeapSizeTotal と変数 HeapNumTotal を更新する．
(7) (6) で取り出した引数と，オリジナルの HeapFree 関数を用いてメモリ領域を解放する．
(8) 再利用を禁止しているメモリ領域の個数が閾値の num_threshold(個数の閾値) より多いか確認する．
  (A) 最低個数より多い場合，処理 (9) へ進む．
  (B) 最低個数より少ない場合，解放処理を終了して，Hook_HeapFree 関数の処理を終える．
(9) 再利用を禁止しているメモリ領域の合計サイズが閾値の size_threshold の半分より小さいか確認する．
  (A) 合計サイズの閾値の半分より小さい場合，処理 (10) へ進む．
  (B) 合計サイズの閾値の半分より大きい場合，処理 (5) へ戻る．
(10) 合計サイズの閾値を指定した範囲内でランダムな値で更新し，Hook_HeapFree 関数の処理を終える．

4 調査と評価

4.1 目的と環境

本稿で行った調査および，評価内容を以下に示す．

(1) 攻撃成功率の評価
メモリ領域を解放する閾値にメモリ領域の個数とサイズを併用した提案方式を用いて UAF 脆弱性攻撃の攻撃成功率を評価した．また，提案方式に設定する閾値の値を変更した場合の攻撃成功率を評価し，閾値と攻撃成功率の関係を評価した．評価結果より，攻撃防止に有効な閾値の値を考察する．
(2) メモリオーバヘッドの評価
メモリ領域を解放する閾値にメモリ領域の個数とサイズを併用した提案方式を用いてメモリオーバヘッドを評価する．提案手法を導入したブラウザでブラウザベンチマークを動作させた際のメモリオーバヘッドを評

価した．また，提案方式における閾値の値を大きくするほど，メモリオーバヘッドが大きくなるため，(2) の攻撃成功率と併せて，適切な閾値とメモリオーバヘッドのバランスを考察する．

上記の調査，および評価は仮想計算機上で行った．用いた環境を**表 1** に示す．(1) の評価は，Windows 8 の IE10 ，(2) の評価は Windows 7 の IE11 を用いて評価を行った．

表 1 調査，および評価に用いた環境

仮想化ソフトウェア	VirtualBox ver. 5.1.22
ゲスト OS	Windows 8 (64bit) Internet Explorer 10
とブラウザ	Windows 7 (64bit) Internet Explorer 11

表 2 CVE-2014-0322 の攻撃成功率

		攻撃成功率
手法適用なし		90%
500	100KB–500KB	80%
	500KB–1MB	15%
	1MB–1.5MB	0%
1000	100KB–500KB	80%
	500KB–1MB	35%
	1MB–1.5MB	0%
1500	100KB–500KB	80%
	500KB–1MB	25%
	1MB–1.5MB	0%
2000	100KB–500KB	70%
	500KB–1MB	20%
	1MB–1.5MB	0%
2500	100KB–500KB	0%
	500KB–1MB	0%
	1MB–1.5MB	0%

4.2 攻撃成功率の評価

提案手法のメモリ領域を解放する閾値を様々に変えて適用した IE10 を用いて攻撃成功率を評価した．評価には，Metasploit で配布されている CVE-2014-0322 の攻撃コードを用いて，攻撃を防止できるか否かを実験した．実験は，提案手法を適用したブラウザを用いて，Metasploit で用意した攻撃コードを配置したページにアクセスする．閾値毎に 20 回試行し，攻撃成功率を求めた．評価は提案手法を適用していない状態と提案手法を適用した状態を比較し，攻撃成功率を測定した．また，提案手法における閾値の値を評価するため，個数の閾値は 500 個，1,000 個，1,500 個，2,000 個，および 2,500 個の提案手法を用いた．これらの個数は，文献 [6] の評価で用いた提案手法の閾値それぞれを半分にしたものを選択した．また，それぞれ，サイズの閾値を 100KB-500KB，500KB-1MB，および 1MB-1.5MB の範囲でランダム化した手法を用いて評価を行った．評価結果を**表 2** に示す．

表 2 より，100KB–500KB の場合，個数の閾値 2,000 以下では，攻撃成功率が高いものの，2,500 では 0 %である．これは，個数の閾値はその個数以上のメモリを再利用禁止にできるものの，100KB–500KB のメモリサイズでは個数の閾値より多くのメモリを再利用禁止にはできず，その数では攻撃成功率を低下させるのに不十分であったためと考えられる．このことから，個数の閾値で攻撃成功率を抑制できる個数を再利用禁止にすることが重要であることがわかる．これは，メモリオーバヘッドを抑制することよりも，攻撃成功率を下げることの方が重要であるためである．

500KB–1MB の場合，個数の閾値が 2,000 以下の場合でも，攻撃成功率はある程度抑制できている．これは，500KB–1MB のサイズの範囲内で，攻撃を抑制するのに十分な個数のメモリを再利用禁止にできていることが多いためであると推察できる．

1MB-1.5MB の場合，攻撃成功率が 0 %である．100KB–500KB における個数の閾値と攻撃成功率の関係から，1MB–1.5MB の場合に再利用を禁止されるメモリの個数は，2.500 程度以上であったことが推察できる．

以上のことから，個数の閾値は攻撃成功率を抑制することに重要であることがわかる．その上で，メモリサイズの閾値を導入することで，攻撃成功率の低下とメモリオーバヘッドの抑制を両立できる場合があることがわかった．また，サイズの閾値以上のメモリを確保すると，そのメモリ以外のメモリが解放されてしまう問題に，個数の閾値で対処できていることがわかる．

4.3 メモリオーバヘッドの評価

ここでは，提案手法を適用していない状態と適用した状態のブラウザにおけるメモリオーバヘッドを比較するため，ブラウザベンチマークを実行した際のブラウザプロセスの仮想メモリ使用量を Performance Monitor を用いて測定した．評価に用いた提案手法は，個数の閾値が 1,500 個と 2,500 個，それぞれサイズの閾値が 100KB から 500KB，500KB から 1MB，および 1MB から 1.5MB の範囲でランダム化する提案手法を用いた．なお，評価には Windows 7(64bit) の IE11 を使用した．ブラウザベンチマークは，Google's Octane 2.0[9]，Apple's SunSpider[10]，および Mozilla's Kraken 1.1[11] の 3 種類を用いた．3 種類のブラウザベンチマークを実行した際のメモリ使用量の推移を測定し，ブラウザベンチマーク実行中に，のメモリ使用量の最大値，実行中のメモリ使用量の平均値を算出した．これを用いて，提案手法を適用していない時の最大値と平均値と比較した際の，閾値毎のメモリオーバヘッドを評価する．評価結果を**図 3** に示す．表中の括弧内の値がオーバヘッドである．

表 3 より，Octane の場合，メモリオーバヘッドは小さ

表 3 ブラウザベンチマーク実行時のメモリ使用量とオーバヘッド

閾値		Octane		SunSpider		Kraken	
個数	ランダム化する範囲	平均	最大値	平均	最大値	平均	最大値
適用なし		358.10MB	633.49MB	103.31MB	113.54MB	213.95MB	364.43MB
1500	100KB–500KB	348.86MB (-2.58%)	638.92MB (0.86%)	109.25MB (5.76%)	122.21MB (7.64%)	314.44MB (46.96%)	664.03MB (82.21%)
	500KB–1MB	355.91MB (-0.61%)	622.05MB (-1.81%)	108.31MB (4.84%)	119.12MB (4.92%)	317.19MB (48.25%)	687.97MB (88.78%)
	1MB–1.5MB	354.10MB (-1.12%)	641.57MB (1.28%)	111.35MB (7.79%)	136.16MB (19.93%)	309.55MB (44.68%)	774.91MB (112.64%)
2500	100KB–500KB	368.04MB (2.78%)	650.42MB (2.67%)	111.30MB (7.74%)	127.14MB (11.98%)	336.56MB (57.30%)	719.22MB (97.36%)
	500KB–1MB	355.91MB (-0.61%)	622.05MB (-1.81%)	112.00MB (8.41%)	128.45MB (13.14%)	321.45MB (50.24%)	722.17MB (98.17%)
	1MB–1.5MB	361.77MB (1.02%)	625.40MB (-1.28%)	110.23MB (6.70%)	124.64MB (9.78%)	326.52MB (52.61%)	735.15MB (101.73%)

い．このことから，個数の閾値である 2,500 個以上のメモリを再利用禁止にしても，そのメモリオーバヘッドがランダム化したメモリサイズの閾値の範囲内に収まっていることが推察できる．

一方，SunSpider と Kraken の場合，メモリオーバヘッドはかなり大きい．これは，ランダム化したサイズの範囲を，再利用禁止にしたメモリの合計サイズが超え，1,500 個，もしくは 2,500 個のメモリがそれぞれ再利用禁止になり，個々のメモリサイズが Octane よりも大きいため，合計の再利用禁止にしたメモリサイズが大きく，メモリオーバヘッドが大きくなってしまったものと推察できる．メモリサイズの閾値を導入しても，メモリオーバヘッドが大きくなる場合があるものの，攻撃成功率低下を優先させるために，個数の閾値を大きくすることが優先されるため，この場合は，提案方式でメモリオーバヘッドは削減できないことがわかった．

5 考察

ランダム化した個数の閾値のみの方式である文献 [6] は，閾値を大きくすることで，攻撃成功率を低下できる．しかし，個数でのみ再利用禁止にするメモリを設定するため，十分に大きな個数を閾値に設定する必要があり，メモリオーバヘッドが大きくなる問題がある．

一方，本提案方式は，まずは，メモリサイズの閾値で再利用を禁止するため，個々のメモリサイズが小さい場合，そのサイズの範囲内で個数の閾値よりも多数のメモリを再利用禁止にできる．このため，攻撃成功率低下に効果が大きいと推察できる．個々のメモリサイズが大きい場合には，メモリオーバヘッドを抑制できないものの，個数の閾値により，攻撃成功率を低下させるのに十分な個数のメモリを再利用禁止にできる．以上のことから，文献 [6] の手法に比べ，本提案方式は，生成削除する個々のメモリサイズが小さいプログラムにおいて，特に攻撃成功率低下の効果が大きい手法であるといえる．

6 おわりに

再利用を禁止しているメモリ領域の個数とメモリサイズを閾値として用いた Windows における提案方式について述べた．合計サイズと個数を併用することで，閾値よりも大きなサイズのメモリ領域が解放された場合にも，一定個数のメモリ領域の再利用を禁止することができる．

攻撃成功率の評価では，個数の閾値は攻撃成功率の抑制に対して重要であることがわかった．

メモリオーバヘッドの評価では，メモリサイズを導入した場合にも，個々のメモリサイズが大きい場合には，メモリオーバヘッドが大きくなることがわかった．

以上の結果より，個々のメモリサイズが大きい場合には，メモリオーバヘッドが抑制できないものの，生成削除する個々のメモリサイズが小さいプログラムにおいて，特に攻撃成功率低下の効果が大きい手法であることを示した．

謝辞 研究を進めるにあたって様々なコメントをして下さった岡山大学大学院自然科学研究科の上川先之氏に感謝を申し上げます．

参考文献

[1] Common vulnerabilities and exposures, available from 〈https://cve.mitre.org/index.html〉 (accessed 2016-12-20).

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

[3] 池上祐太，山内利宏：メモリ再利用を禁止するライブラリにより Use-After-Free 脆弱性攻撃を防止する手法の提案，コンピュータセキュリティシンポジウム 2014 論文集，Vol.2014，No.2，pp.567–574 (2014).

[4] 山内利宏，池上祐太：メモリ再利用禁止による Use-After-Free 脆弱性攻撃防止手法の実現と評価，情報処理学会研究報告，Vol.2015-CSEC-69，No.21，pp.1-8 (2015).

[5] Toshihiro Yamauchi and Yuta Ikegami: 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 (2016).

[6] 伴侑弥，山内利宏：Use-After-Free 脆弱性攻撃防止手法におけるメモリ解放契機の評価，情報処理学会研究報告，Vol.2017-CSEC-78，No.26，pp.1-7(2017).

[7] Tang, J.: Mitigating uaf exploits with delay free for internet explorer, available from 〈http://blog.trendmicro.com/trendlabs-security-intelligence/mitigating-uaf-exploits-with-delay-free-for-internet-explorer/〉 (accessed 2016-12-20).

[8] Metasploit, available from 〈http://www.metasploit.com/〉 (accessed 2017-2-6).

[9] Octane 2.0, available from 〈https://chromium.github.io/octane/〉 (accessed 2017-6-1).

[10] SunSpider 1.0.2 JavaScript Benchmark, available from 〈https://webkit.org/perf/sunspider/sunspider.html〉 (accessed 2017-2-6).

[11] Kraken JavaScript Benchmark (version 1.1), available from 〈https://krakenbenchmark.mozilla.org/〉 (accessed 2017-2-6).