



**Finlay Reid**

**1904629**

**CMP417**

**Engineering Resilient Systems**

## Contents

Introduction .....	3
<i>CVEs related to memory leaks allowing for Dos attacks</i> .....	4
CVE-2015-8631.....	4
CVE-2015-5292.....	4
CVE-2001-0237.....	4
CVE-2018-16807 .....	5
Recommendation/Implementation .....	5
CVE-2015-8631.....	5
Conclusion.....	8
Appendices.....	9
Basic memory leak code .....	9
Valgrind Basic leak command .....	9
Basic memory Valgrind results.....	9
Basic memory pointer code .....	12
Basic memory pointer Valgrind command .....	13
Basic memory pointer valgrind results .....	13
References .....	18

## Introduction

In the modern software development process, security remains a side focus as developers are more concerned with the release of their products. When in fact security should be one of the most pertinent and criticized aspects of the software, including its development process. A software security report was undertaken by Veracode which discovered that 75% of applications had security flaws. Of the tested applications 24 percent of those were classified as high severity vulnerabilities(Veracode, 2022). This Demonstrates the sheer volume of flaws found within today's software and why developers should be familiar with the practices related to developing secure software. Furthermore, the designer should be aware of the parts of the software that are most susceptible to security vulnerabilities and make the necessary provisions to mitigate these risks. There is much documentation available to developers detailing the best coding/security practices which should be used in the development process including the SEI CERT C Coding Standard: a Rulebook for Developing Safe, Reliable, and Secure Systems(SEI CERT C Coding Standard, 2022). This should be utilized with a resource like Waverly's Top 25 Coding Errors Leading to Software Vulnerabilities allowing developers to have knowledge related to the most common software vulnerabilities and their causes(Hladun, 2022).

The fictional scenario involved a company named Scottish glen and its recent complications with a hacktivist group. Threatening to target the company, by presumably exploiting weaknesses within company systems. Specifically, Scottish glens Kerberos network authentication system, a network authentication protocol written in C that is used to provide a common authentication and authorization solution.

A blog published by plumb.io detailed the frequency and severity of memory leaks found within software applications. 2,180 different applications were analysed over the experiment with 31% containing a memory leak(Mägi, 2022). These errors within the software can lead to the application being exploited through varying attacks including Dos attacks, privilege attacks, and RCE. Overall 4 Cve's were recorded in which Kerberos could be exploited through a memory leak allowing for a Denial of service attack to occur.

## *CVEs related to memory leaks allowing for Dos attacks*

The following security faults are all related to memory leaks found within the Kerberos software in varying versions, files, and plugins. Memory leaks occur when the author of the code forgets to free up memory resources after the block of memory is no longer required within the program. Causes of memory leaks can often be attributed to two circumstances, one is error conditions and indecision concerning which area of the program is responsible for releasing the memory. These types of security faults give way for malicious users to exploit the developer's mistakes, particularly through a form of attack called denial of service. Dos exploits at their core look to disrupt the service, application, etc from functioning as it was intended, a few of the exploits work by exhausting all available memory.

### *CVE-2015-8631*

Multiple memory leaks were discovered in `kadmin/server/server_stubs.c` in `kadmind` in MIT Kerberos 5 (aka `krb5`) before 1.13.4 and 1.14.x before 1.14.1. This flaw enables malicious users to complete a denial of service attack through a request specifying a NULL principal name. Repeating these requests will eventually cause `kadmind` to exhaust all available memory, essentially causing the Kerberos software to fail. The ticket was created on Wednesday 27<sup>th</sup> of January 2016 and the recommended fix includes upgrading your `krb5` packages/updating Kerberos to the newest version. With a base score of 6.5 ranking it as a medium-level security vulnerability, the flaw does not have the highest priority however it has increased by 2.5 from its initial metric scoring of 4.0 in the previous `cvss` version. Discovered by Simo Source, the security flaw can be mitigated by initialising a few variables including `client_name` and `server_name` then releasing them in the cleanup handler (NVD - CVE-2015-8631, 2022).

### *CVE-2015-5292*

In the Privilege Attribute Certificate (PAC) responder plugin (`sssd_pac_plugin.so`) located in System Security Services Daemon (SSSD) 1.10 before 1.13.1, a memory leak is present that enables malicious users to cause a denial of service. This is achieved through exhausting all available memory on the system by making repeated requests to a Kerberized daemon application configured to authenticate using the PAC responder plug-in. The exploit was first reported on the 30<sup>th</sup> of September of 2015 and the recommended fix includes updating Kerberos to the newest version. Red hat has classified the security fault as low severity with a `cvss` score of 6.8. Furthermore, the exploit has been listed as having a low access complexity meaning the security does not require specialized knowledge to exploit the flaw (CVE -CVE-2015-5292, 2022).

### *CVE-2001-0237*

A flaw within the Kerberos service allows an attacker to facilitate a denial of service attack. The exploit works by having a connection to the Kerberos service and then disconnecting without scanning the socket, this will cause a memory leak. In order to take advantage of this over 4000 connections to the Kerberos service are made and this ensures the software will stop taking connections on port 88(Kerberos) and 464(kpasswd). The products affected by this security flaw located in the Kerberos service only currently affect the Microsoft Windows 2000 operating system. With a `cvss` score of 5.0, this type of vulnerability does not have the highest priority however, the exploit requires little to no specialized knowledge and causes a reduced performance or interruptions in resource availability (CVE -CVE-2001-0237, 2022).

### CVE-2018-16807

in the file main.bro located in scripts/base/protocols/krb/ there is a potential memory leak present which when used maliciously can be used to cause a denial of service attack. The CVE designated this exploit as having a 5.0 metric score due to its low access complexity while partial availability impact. The security flaw was first discovered by the user Maksim Shudrak and the fix was posted on the 11<sup>th</sup> of September 2018. The fix consisted of performing checks on the msg?\${service\_name} variable (NVD - CVE-2018-16807, 2022).

## Recommendation/Implementation

### CVE-2015-8631

The security faults located in the (sssd\_pac\_plugin.so could have been discovered before the product was released through the utilization of dynamic analysis. This type of analysis evaluates the program at runtime whether, on a virtual processor or real, the program is watched allowing developers to glimpse into its real-world behaviour. Essentially having additional code inserted into the program and analysing it externally allows dynamic analysis to perform checks such as if it is attempting to allocate more memory than required or if the program is trying to use out-of-bounds memory. As a small company within the energy sector theoretically, a lot of customers could rely on Scottish glen to provide their homes with energy or assist in that process meaning any delay in production through security faults could have a huge impact. Customers have the potential to be without electricity or gas for some time.

If the developers of the service utilized a secure development practice such as dynamic analysis this risk of a denial of service attack exploited through memory leakage would be mitigated. Concerning the practicality of dynamic analysis, the process is much less time-consuming than other secure coding practices. One of the advantages of this form of evaluation is the wealth of automated tools available and it can be conducted against any application. Not to mention its seamless ability to integrate with other secure coding practices, for example, the developer could utilize both an in-depth code review first then complete a static/dynamic analysis as this defense-in-depth approach is the most suitable way for developers to discover security faults. When testing for security faults related to memory leaks dynamic analysis is particularly effective when compared with other secure coding practices such as static analysis. This is because vulnerabilities such as memory leaks can be confirmed to be memory leaks at runtime, however, static analyzers can find simple cases of memory leaks but they are nowhere near as effective as their counterparts. This is apparent as memory leaks typically occur when the program's code is convoluted and the memory is free/allocated in different parts of the program. Furthermore, static analysers can only make assumptions that a piece of code is going to create a memory leak whereas previously mentioned dynamic analysis can confirm it. As Kerberos is such a popular piece of software and it was created by MIT the developers should consider a high-end dynamic analyser such as the IDRA tool suite as it integrates software life-cycle traceability, static and dynamic analysis, unit test, and system-level testing on virtually any host or target platform. Dynamic analysis like anything has its disadvantages including its automated tools often produce false positives and false negatives. Furthermore, it is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

The specific vulnerability CVE-2015-5292 as previously mentioned is a security fault related to the class of developer mistakes known as memory leaks. Found within the Privilege Attribute Certificate (PAC) responder plugin (sssd\_pac\_plugin.so) located in System Security Services Daemon (SSSD) 1.10 before 1.13.1 this fault could have been avoided through dynamic analysis. In order to view this security vulnerability the source code of Kerberos was downloaded from its official site and viewed in a text editor.

```
krb5_pac pac; // krb5_pac is actually a pointer type
...
kerr = krb5_pac_parse(kcontext, sssdctx->data.data, // allocation of the pac pointer from input data
                     sssdctx->data.length, &pac);
...
kerr = krb5_pac_verify(kcontext, pac, // verification of the pac content
                      req->ticket->enc_part2->times.authtime,
                      req->ticket->enc_part2->client, key, NULL);
...
// no reference to pac
return 0; // memory leak!
```

Figure 1 - Memory leak caused in the plugin

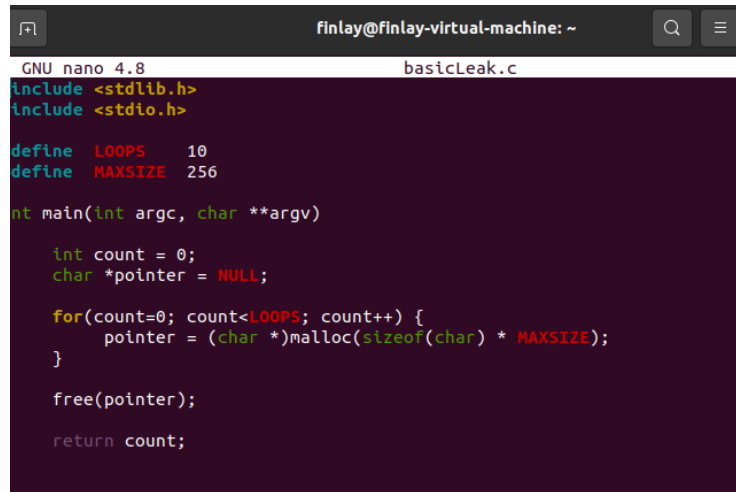
Shown in [Figure 1](#) is the piece of code that is responsible for causing the memory leak as there is no reference to the pac variable. As Kerberos is written in C, memory must always be deallocated which can be accomplished using the free() function or since krb5\_pac is a pointer type specifying the pointer is null. This vulnerable piece of code reportedly leaked gigabytes of memory after several days(1GB/day on average for 10 authentications per second) making it ideal for malicious users attempting to disrupt the service through a denial of service attack. [Figure 2](#) presents the additions necessary to fix the memory leak by freeing up the memory by specifying the pointer is null.

```
// deallocate pac - fixes memory leak reported in bug ...
+   krb5_pac_free(kcontext, pac);
+   pac = NULL;
+   // check result of the verification
+   if (kerr != 0) {
+       ...
+   }
```

Figure 2 - Fix for memory leak

To demonstrate the capabilities of dynamic analysis the software Valgrind was utilized. This program is used for several things including memory debugging, memory leak detection, and profiling. First, the Valgrind software was downloaded on a virtual machine running the latest version of Ubuntu using the typical installation commands on Debian derivatives. After setting up the Valgrind software, two basic c programs were created with memory leaks present within the code.

**Figure 3** displays the code, there are 10 allocations of size MAXSIZE, and all except the last is lost. If there is a pointer that is pointed to the memory it cannot be recovered in runtime. To fix this the free() function can be used in the loop.



```
finlay@finlay-virtual-machine: ~
GNU nano 4.8 basicLeak.c
#include <stdlib.h>
#include <stdio.h>

define LOOPS 10
define MAXSIZE 256

nt main(int argc, char **argv)

    int count = 0;
    char *pointer = NULL;

    for(count=0; count<LOOPS; count++) {
        pointer = (char *)malloc(sizeof(char) * MAXSIZE);
    }

    free(pointer);

    return count;
```

Figure 3 – basicLeak.c code

**Figure 4** presents a snippet of the results from running the command shown in the Valgrind Basic leak command appendix. The full results of this command were piped to a text file and can be seen in the Basic memory Valgrind results appendix. This shows that there is a memory leak present within the program.



```
HEAP SUMMARY:
  in use at exit: 2,304 bytes in 9 blocks
  total heap usage: 10 allocs, 1 frees, 2,560 bytes allocated

Searching for pointers to 9 not-freed blocks
Checked 69,944 bytes

2,304 bytes in 9 blocks are definitely lost in loss record 1 of 1
  at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64.so)
  by 0x10919D: main (in /home/finlay/basicLeak)

LEAK SUMMARY:
  definitely lost: 2,304 bytes in 9 blocks
  indirectly lost: 0 bytes in 0 blocks
  possibly lost: 0 bytes in 0 blocks
  still reachable: 0 bytes in 0 blocks
  suppressed: 0 bytes in 0 blocks

ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
finlay-virtual-machine:~$
```

Figure 4 – Results from Valgrind for basicLeak.c

Another program was created this time implementing pointers that cause a memory leak which can be seen in the appendix Basic memory pointer. Both of these examples are similar to the one found in CVE-2015-8631 demonstrating that dynamic analysis tools such as Valgrind would discover and provide further information related to the fault. Allowing developers of Scottish glen to implement a fix, therefore, preventing any potential denial of service attack.

## Conclusion

In conclusion, using a secure coding practice such as dynamic analysis would have discovered the memory leak security faults found within the Kerberos application. Discovering these vulnerabilities would have enabled the developers to implement their fix for this, therefore, mitigating the risk posed by a potential denial of service attack that utilizes these memory leaks. Dynamic analysis is extremely effective, particularly for finding memory-related faults with its easy integration into testing suites, and its ability to demonstrate insecurity makes it the perfect practice to be used by the development team. When compared with other secure coding practices, this is less time-consuming as it's completely automated and works well with other practices, as Dynamic and static analysis techniques are most powerful when used in tandem. If the memory leaks are exploited through a denial of service attack it could potentially heavily disrupt Scottish glens services costing them a substantial amount of money. An example, in the utilization of dynamic analysis, includes the safety demonstration of medical devices to the FDA(Knuth, D, unknown).



## Appendices

### Basic memory leak code

```
#include <stdlib.h>
#include <stdio.h>

#define LOOPS    10
#define MAXSIZE  256

int main(int argc, char **argv)
{
    int count = 0;
    char *pointer = NULL;

    for(count=0; count<LOOPS; count++) {
        pointer = (char *)malloc(sizeof(char) * MAXSIZE);
    }

    free(pointer);

    return count;
}
```

### Valgrind Basic leak command

```
valgrind --leak-check=full --log-file=basic.txt -v ./basicLeak
```

*Figure 5 - Valgrind command used for the basic c file*

### Basic memory Valgrind results

=2905== Memcheck, a memory error detector

==2905== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==2905== Using Valgrind-3.15.0-608cb11914-20190413 and LibVEX; rerun with -h for copyright info

==2905== Command: ./basicLeak

==2905== Parent PID: 2015

==2905==

--2905--

--2905-- Valgrind options:

--2905-- --leak-check=full

--2905-- --log-file=basic

--2905-- -v

--2905-- Contents of /proc/version:

--2905-- Linux version 5.13.0-35-generic (buildd@ubuntu) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #40~20.04.1-Ubuntu SMP Mon Mar 7 09:18:32 UTC 2022

--2905--

--2905-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-ssse3-avx-avx2-bmi-f16c-rdrand

--2905-- Page sizes: currently 4096, max supported 4096

--2905-- Valgrind library directory: /usr/lib/x86\_64-linux-gnu/valgrind

--2905-- Reading syms from /home/finlay/basicLeak

```

--2905-- Reading syms from /usr/lib/x86_64-linux-gnu/ld-2.31.so
--2905-- Considering /usr/lib/x86_64-linux-gnu/ld-2.31.so ..
--2905-- .. CRC mismatch (computed 0306b78c wanted 8d362b37)
--2905-- Considering /lib/x86_64-linux-gnu/ld-2.31.so ..
--2905-- .. CRC mismatch (computed 0306b78c wanted 8d362b37)
--2905-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.31.so ..
--2905-- .. CRC is valid
--2905-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-linux
--2905-- object doesn't have a symbol table
--2905-- object doesn't have a dynamic symbol table
--2905-- Scheduler: using generic scheduler lock implementation.
--2905-- Reading suppressions file: /usr/lib/x86_64-linux-gnu/valgrind/default.supp
==2905== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-2905-by-finlay-on-???
==2905== embedded gdbserver: writing to /tmp/vgdb-pipe-to-vgdb-from-2905-by-finlay-on-???
==2905== embedded gdbserver: shared mem /tmp/vgdb-pipe-shared-mem-vgdb-2905-by-finlay-on-???
==2905==
==2905== TO CONTROL THIS PROCESS USING vgdb (which you probably
==2905== don't want to do, unless you know exactly what you're doing,
==2905== or are doing some strange experiment):
==2905== /usr/lib/x86_64-linux-gnu/valgrind/../../bin/vgdb --pid=2905 ...command...
==2905==
==2905== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==2905== /path/to/gdb ./basicLeak
==2905== and then give GDB the following command
==2905== target remote | /usr/lib/x86_64-linux-gnu/valgrind/../../bin/vgdb --pid=2905
==2905== --pid is optional if only one valgrind process is running
==2905==
--2905-- REDIR: 0x4022e10 (ld-linux-x86-64.so.2:strlen) redirected to 0x580c9ce2 (???)
--2905-- REDIR: 0x4022be0 (ld-linux-x86-64.so.2:index) redirected to 0x580c9cfc (???)
--2905-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_core-amd64-linux.so
--2905-- object doesn't have a symbol table
--2905-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so
--2905-- object doesn't have a symbol table
==2905== WARNING: new redirection conflicts with existing -- ignoring it
--2905-- old: 0x04022e10 (strlen ) R-> (0000.0) 0x580c9ce2 ???
--2905-- new: 0x04022e10 (strlen ) R-> (2007.0) 0x0483f060 strlen
--2905-- REDIR: 0x401f5f0 (ld-linux-x86-64.so.2:strcmp) redirected to 0x483ffd0 (strcmp)
--2905-- REDIR: 0x4023370 (ld-linux-x86-64.so.2:mempcpy) redirected to 0x4843a20 (mempcpy)

```

```

--2905-- Reading syms from /usr/lib/x86_64-linux-gnu/libc-2.31.so
--2905-- Considering /usr/lib/x86_64-linux-gnu/libc-2.31.so ..
--2905-- .. CRC mismatch (computed ef41b1a0 wanted f854b801)
--2905-- Considering /lib/x86_64-linux-gnu/libc-2.31.so ..
--2905-- .. CRC mismatch (computed ef41b1a0 wanted f854b801)
--2905-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.31.so ..
--2905-- .. CRC is valid
--2905-- REDIR: 0x48fc4b0 (libc.so.6:memmove) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb7b0 (libc.so.6:strncpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc7e0 (libc.so.6:strcasecmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb0d0 (libc.so.6:strcat) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb810 (libc.so.6:rindex) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fdc80 (libc.so.6:rawmemchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4918d10 (libc.so.6:wmemchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4918850 (libc.so.6:wscmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc610 (libc.so.6:memcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc440 (libc.so.6:bcmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb740 (libc.so.6:strncmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb180 (libc.so.6:strcmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc570 (libc.so.6:memset) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4918810 (libc.so.6:wcschr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb6a0 (libc.so.6:strnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb260 (libc.so.6:strcspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc830 (libc.so.6:strncasecmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb200 (libc.so.6:strcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc980 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4919f80 (libc.so.6:wcsnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4918890 (libc.so.6:wscpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb850 (libc.so.6:strpbrk) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb130 (libc.so.6:index) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fb660 (libc.so.6:strlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4904bd0 (libc.so.6:memrchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc880 (libc.so.6:strcasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc400 (libc.so.6:memchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x4918960 (libc.so.6:wcslen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fbb10 (libc.so.6:strspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc780 (libc.so.6:stpncpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc720 (libc.so.6:stpcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)

```

```

--2905-- REDIR: 0x48fdcc0 (libc.so.6:strchrnul) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x48fc8d0 (libc.so.6:strncasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--2905-- REDIR: 0x49e4410 (libc.so.6:___strchr_avx2) redirected to 0x483ea10 (rindex)
--2905-- REDIR: 0x48f6110 (libc.so.6:malloc) redirected to 0x483b780 (malloc)
--2905-- REDIR: 0x48f6700 (libc.so.6:free) redirected to 0x483c9d0 (free)

==2905==
==2905== HEAP SUMMARY:
==2905==   in use at exit: 2,304 bytes in 9 blocks
==2905== total heap usage: 10 allocs, 1 frees, 2,560 bytes allocated
==2905==
==2905== Searching for pointers to 9 not-freed blocks
==2905== Checked 69,944 bytes
==2905==
==2905== 2,304 bytes in 9 blocks are definitely lost in loss record 1 of 1
==2905==   at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==2905==   by 0x10919D: main (in /home/finlay/basicLeak)
==2905==
==2905== LEAK SUMMARY:
==2905==   definitely lost: 2,304 bytes in 9 blocks
==2905==   indirectly lost: 0 bytes in 0 blocks
==2905==   possibly lost: 0 bytes in 0 blocks
==2905==   still reachable: 0 bytes in 0 blocks
==2905==   suppressed: 0 bytes in 0 blocks
==2905==
==2905== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

```

## Basic memory pointer code

```

#include <stdlib.h>
#include <stdint.h>

struct _List {
    int32_t* data;
    int32_t length;
};
typedef struct _List List;

List* resizeArray(List* array) {
    int32_t* dPtr = array->data;
    dPtr = realloc(dPtr, 15 * sizeof(int32_t)); //doesn't update array-
    >data
    return array;
}

```

```

int main() {
    List* array = calloc(1, sizeof(List));
    array->data = calloc(10, sizeof(int32_t));
    array = resizeArray(array);

    free(array->data);
    free(array);
    return 0;
}

```

## Basic memory pointer Valgrind command

```

finlay@finlay-virtual-machine:~$ valgrind --leak-check=full --log-file=basicpointer.txt -v ./basicLeakpointer

```

## Basic memory pointer valgrind results

==3144== Memcheck, a memory error detector

==3144== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==3144== Using Valgrind-3.15.0-608cb11914-20190413 and LibVEX; rerun with -h for copyright info

==3144== Command: ./basicLeakpointer

==3144== Parent PID: 2015

==3144==

--3144--

--3144-- Valgrind options:

--3144-- --leak-check=full

--3144-- --log-file=basicpointer.txt

--3144-- -v

--3144-- Contents of /proc/version:

--3144-- Linux version 5.13.0-35-generic (buildd@ubuntu) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #40~20.04.1-Ubuntu SMP Mon Mar 7 09:18:32 UTC 2022

--3144--

--3144-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-ssse3-avx-avx2-bmi-f16c-rdrand

--3144-- Page sizes: currently 4096, max supported 4096

--3144-- Valgrind library directory: /usr/lib/x86\_64-linux-gnu/valgrind

--3144-- Reading syms from /home/finlay/basicLeakpointer

--3144-- Reading syms from /usr/lib/x86\_64-linux-gnu/ld-2.31.so

--3144-- Considering /usr/lib/x86\_64-linux-gnu/ld-2.31.so ..

--3144-- .. CRC mismatch (computed 0306b78c wanted 8d362b37)

--3144-- Considering /lib/x86\_64-linux-gnu/ld-2.31.so ..

--3144-- .. CRC mismatch (computed 0306b78c wanted 8d362b37)

--3144-- Considering /usr/lib/debug/lib/x86\_64-linux-gnu/ld-2.31.so ..

--3144-- .. CRC is valid

```

--3144-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-linux
--3144-- object doesn't have a symbol table
--3144-- object doesn't have a dynamic symbol table
--3144-- Scheduler: using generic scheduler lock implementation.
--3144-- Reading suppressions file: /usr/lib/x86_64-linux-gnu/valgrind/default.supp
==3144== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-3144-by-finlay-on-???
==3144== embedded gdbserver: writing to /tmp/vgdb-pipe-to-vgdb-from-3144-by-finlay-on-???
==3144== embedded gdbserver: shared mem /tmp/vgdb-pipe-shared-mem-vgdb-3144-by-finlay-on-???
==3144==
==3144== TO CONTROL THIS PROCESS USING vgdb (which you probably
==3144== don't want to do, unless you know exactly what you're doing,
==3144== or are doing some strange experiment):
==3144== /usr/lib/x86_64-linux-gnu/valgrind/../../bin/vgdb --pid=3144 ...command...
==3144==
==3144== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3144== /path/to/gdb ./basicLeakpointer
==3144== and then give GDB the following command
==3144== target remote | /usr/lib/x86_64-linux-gnu/valgrind/../../bin/vgdb --pid=3144
==3144== --pid is optional if only one valgrind process is running
==3144==
--3144-- REDIR: 0x4022e10 (ld-linux-x86-64.so.2:strlen) redirected to 0x580c9ce2 (???)
--3144-- REDIR: 0x4022be0 (ld-linux-x86-64.so.2:index) redirected to 0x580c9cfc (???)
--3144-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_core-amd64-linux.so
--3144-- object doesn't have a symbol table
--3144-- Reading syms from /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so
--3144-- object doesn't have a symbol table
==3144== WARNING: new redirection conflicts with existing -- ignoring it
--3144-- old: 0x04022e10 (strlen ) R-> (0000.0) 0x580c9ce2 ???
--3144-- new: 0x04022e10 (strlen ) R-> (2007.0) 0x0483f060 strlen
--3144-- REDIR: 0x401f5f0 (ld-linux-x86-64.so.2:strcmp) redirected to 0x483ffd0 (strcmp)
--3144-- REDIR: 0x4023370 (ld-linux-x86-64.so.2:mempcpy) redirected to 0x4843a20 (mempcpy)
--3144-- Reading syms from /usr/lib/x86_64-linux-gnu/libc-2.31.so
--3144-- Considering /usr/lib/x86_64-linux-gnu/libc-2.31.so ..
--3144-- .. CRC mismatch (computed ef41b1a0 wanted f854b801)
--3144-- Considering /lib/x86_64-linux-gnu/libc-2.31.so ..

```

```

--3144-- .. CRC mismatch (computed ef41b1a0 wanted f854b801)
--3144-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.31.so ..
--3144-- .. CRC is valid
--3144-- REDIR: 0x48fc4b0 (libc.so.6:memmove) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb7b0 (libc.so.6:strncpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc7e0 (libc.so.6:strcasecmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb0d0 (libc.so.6:strcat) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb810 (libc.so.6:rindex) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fdc80 (libc.so.6:rawmemchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4918d10 (libc.so.6:wmemchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4918850 (libc.so.6:wscmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc610 (libc.so.6:mempcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc440 (libc.so.6:bcmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb740 (libc.so.6:strncmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb180 (libc.so.6:strcmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc570 (libc.so.6:memset) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4918810 (libc.so.6:wcschr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb6a0 (libc.so.6:strnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb260 (libc.so.6:strcspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc830 (libc.so.6:strncasecmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb200 (libc.so.6:strcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc980 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4919f80 (libc.so.6:wcsnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4918890 (libc.so.6:wscpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb850 (libc.so.6:strpbrk) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb130 (libc.so.6:index) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fb660 (libc.so.6:strlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4904bd0 (libc.so.6:memrchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc880 (libc.so.6:strcasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc400 (libc.so.6:memchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x4918960 (libc.so.6:wcslen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fbb10 (libc.so.6:strspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc780 (libc.so.6:stpncpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fc720 (libc.so.6:stpcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x48fdcc0 (libc.so.6:strchrnul) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)

```

```

--3144-- REDIR: 0x48fc8d0 (libc.so.6:strncasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--3144-- REDIR: 0x49e4410 (libc.so.6:__strchr_avx2) redirected to 0x483ea10 (rindex)
--3144-- REDIR: 0x48f7b40 (libc.so.6:calloc) redirected to 0x483dce0 (calloc)
--3144-- REDIR: 0x48f6eb0 (libc.so.6:realloc) redirected to 0x483df30 (realloc)
--3144-- REDIR: 0x48f6700 (libc.so.6:free) redirected to 0x483c9d0 (free)

==3144== Invalid free() / delete / delete[] / realloc()

==3144== at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144== by 0x109215: main (in /home/finlay/basicLeakpointer)
==3144== Address 0x4a51090 is 0 bytes inside a block of size 40 free'd
==3144== at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144== by 0x1091B4: resizeArray (in /home/finlay/basicLeakpointer)
==3144== by 0x109202: main (in /home/finlay/basicLeakpointer)
==3144== Block was alloc'd at
==3144== at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144== by 0x1091EC: main (in /home/finlay/basicLeakpointer)

==3144==
==3144==
==3144== HEAP SUMMARY:
==3144== in use at exit: 60 bytes in 1 blocks
==3144== total heap usage: 3 allocs, 3 frees, 116 bytes allocated
==3144==
==3144== Searching for pointers to 1 not-freed blocks
==3144== Checked 69,960 bytes
==3144==
==3144== 60 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3144== at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144== by 0x1091B4: resizeArray (in /home/finlay/basicLeakpointer)
==3144== by 0x109202: main (in /home/finlay/basicLeakpointer)
==3144==
==3144== LEAK SUMMARY:
==3144== definitely lost: 60 bytes in 1 blocks
==3144== indirectly lost: 0 bytes in 0 blocks
==3144== possibly lost: 0 bytes in 0 blocks
==3144== still reachable: 0 bytes in 0 blocks
==3144== suppressed: 0 bytes in 0 blocks

```



```
==3144==
==3144== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==3144==
==3144== 1 errors in context 1 of 2:
==3144== Invalid free() / delete / delete[] / realloc()
==3144==   at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144==   by 0x109215: main (in /home/finlay/basicLeakpointer)
==3144== Address 0x4a51090 is 0 bytes inside a block of size 40 free'd
==3144==   at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144==   by 0x1091B4: resizeArray (in /home/finlay/basicLeakpointer)
==3144==   by 0x109202: main (in /home/finlay/basicLeakpointer)
==3144== Block was alloc'd at
==3144==   at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3144==   by 0x1091EC: main (in /home/finlay/basicLeakpointer)
==3144==
==3144== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

## References

- Cve.mitre.org. 2022. *CVE -CVE-2015-5292*. [online] Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5292>> [Accessed 22 March 2022].
- Cve.mitre.org. 2022. *CVE -CVE-2001-0237*. [online] Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0237>> [Accessed 22 March 2022].
- Hladun, I., 2022. Top 25 Coding Errors Leading to Software Vulnerabilities - Waverley. [online] Waverley. Available at: <<https://waverleysoftware.com/blog/top-software-vulnerabilities/>> [Accessed 13 March 2022].
- Ldra.com. 2022. [online] Available at: <<https://ldra.com/products/ldra-tool-suite/>> [Accessed 22 March 2022].
- Mägi, I., 2022. *Memory leaks – measuring frequency and severity | Plumbr – User Experience & Application Performance Monitoring*. [online] Plumbr – User Experience & Application Performance Monitoring. Available at: <<https://plumbr.io/blog/memory-leaks/memory-leaks-measuring-frequency-and-severity>> [Accessed 13 March 2022].
- Knuth, D., Using Dynamic Software Analysis to Support Medical Device Approval.
- Nvd.nist.gov. 2022. *NVD - CVE-2015-8631*. [online] Available at: <<https://nvd.nist.gov/vuln/detail/CVE-2015-8631>> [Accessed 22 March 2022].
- Nvd.nist.gov. 2022. *NVD - CVE-2018-16807*. [online] Available at: <<https://nvd.nist.gov/vuln/detail/CVE-2018-16807>> [Accessed 22 March 2022].
- Resources.sei.cmu.edu. 2022. SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems (2016 Edition). [online] Available at: <<https://resources.sei.cmu.edu/forms/secure-coding-form.cfm>> [Accessed 13 March 2022].
- Stackpath.com. 2022. *What is Dynamic Analysis?*. [online] Available at: <<https://www.stackpath.com/edge-academy/what-is-dynamic-analysis>> [Accessed 22 March 2022].
- Veracode. 2022. *Veracode's 10th State of Software Security Report Finds Organizations Reduce Rising 'Security Debt' via DevSecOps, Special Sprints | Veracode*. [online] Available at: <<https://www.veracode.com/press-release/veracodes-10th-state-software-security-report-finds-organizations-reduce-rising>> [Accessed 13 March 2022].
- Web.mit.edu. 2022. *Historic MIT Kerberos Releases*. [online] Available at: <<http://web.mit.edu/kerberos/dist/historic.html#krb5-1.12-src>> [Accessed 22 March 2022].