Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'makedumpfile.conf.5' command

$ man makedumpfile.conf.5

MAKEDUMPFILE.CONF(5) Linux System Administrator's Manual MAKEDUMPFILE.CONF(5)

NAME

makedumpfile.conf - The filter configuration file for makedumpfile(8).

DESCRIPTION

The makedumpfile.conf is a configuration file for makedumpfile tool.

makedumpfile.conf file contains the erase commands to filter out de?

sired kernel data from the vmcore while creating DUMPFILE using make?

dumpfile tool. makedumpfile reads the filter config and builds the

list of memory addresses and its sizes after processing filter com?

mands. The memory locations that require to be filtered out are then

poisoned with character X (58 in Hex).

FILE FORMAT

The file consists of module sections that contains filter commands. A

section begins with the name of the section in square brackets and con?

tinues until the next section begins.

"["<ModuleName>"]"

<FilterCommands>

where

"[" is the character [

"]" is the character]

<ModuleName>

is either 'vmlinux' or name of a Linux kernel module.

<FilterCommands>

is a list of one or more filter commands as described in the section FILTER COMMANDS of this manual page.

The section name indicates a kernel module name (including vmlinux) where the symbols specified in subsequent erase commands belong to. The unnamed section defaults to [vmlinux] section. However, a user can also explicitly define [vmlinux] section. The sections help makedumpfile tool to select appropriate kernel or module debuginfo file before pro? cessing the subsequent erase commands. Before selecting appropriate de? buginfo file, the module name is validated against the loaded modules from the vmcore. If no match is found, then the section is ignored and makedumpfile skips to the next module section. If match is found, then makedumpfile will try to load the corresponding module debuginfo file. If module debuginfo is not available then, makedumpfile will skip the section with a warning message.

FILTER COMMANDS

filter command

A filter command is either an erase command or a loop construct. Each erase command and loop construct must start with a new line. Each fil? ter command describes data in the dump to be erased. Syntax:

<EraseCommands>|<LoopConstruct>

where

<EraseCommands>

Described in the subsection erase command of this manual page.

<LoopConstruct>

Described in the subsection Loop construct of this manual page.

erase command

Erase specified size of a kernel data referred by specified kernel/mod? ule symbol or its member component. The erase command syntax is:

erase <Symbol>[.member[...]] [size <SizeValue>[K|M]]

erase <Symbol>[.member[...]] [size <SizeSymbol>]

erase <Symbol>[.member[...]] [nullify]

where

<Symbol>

A kernel or module symbol (variable) name that is part of global symbols /proc/kallsyms.

<SizeValue>

A positive integer value as a size of the data in bytes to be erased. The suffixes 'K' and 'M' can be used to specify kilo? bytes and Megabytes respectively where, K means 1024 bytes and M means 1024 ^ 2 = 1048576 bytes. The suffixes are not case sen? sitive.

<SizeSymbol>

A simple expression of the form <Symbol>[.member[...]] that de? notes a symbol which contains a positive integer value as a size of the data in bytes to be erased.

<Symbol>[.member[...]]

A simple expression that results into either a global kernel symbol name or its member components. The expression always uses '.' operator to specify the member component of kernel symbol or its member irrespective of whether it is of pointer type or not.

member[...]

Member or component of member in <Symbol>.

The erase command takes two arguments 1. kernel symbol name or its mem? ber components and 2. size of the data referred by argument (1) OR nul? lify keyword. The second argument size OR nullify is optional. The unit for size value is in bytes. If size option is not specified then the size of the first argument is determined according to its data type us? ing dwarf info from debuginfo file. In case of 'char *' data type, the length of string pointed by 'char *' pointer is determined with an up? per limit of 1024. The size can be specified in two forms 1. a integer value as explained above (<SizeValue>) and 2. a simple expression in the form of <Symbol>[.member[...]]] that results into base type (inte? ger) variable.

If the specified <Symbol> is of type 'void *', then user needs to pro? vide either size or nullify option, otherwise the erase command will not have any effect.

The nullify option only works if specified <Symbol> is a pointer. In? stead of erasing data pointed by the specified pointer nullify erases the pointer value and set it to '0' (NULL). Please note that by nulli? fying the pointer values may affect the debug ability of created DUMP? FILE. Use the nullify option only when the size of data to be erased is not known. e.g. data pointed by 'void *'.

Let us look at the makedumpfile.conf file from the example below which was configured to erase desired kernel data from the kernel module with name mymodule. At line 1 and 3, the user has not specified size option while erasing 'array_var' and 'mystruct1.name' symbols. Instead the user depends on makedumpfile to automatically determine the sizes to be erased i.e 100 bytes for 'array_var' and 11 bytes for 'mystruct1.name'. At line 2, while erasing the 'mystruct1.buffer' member the user has specified the size value 25 against the actual size of 50. In this case the user specified size takes the precedence and makedumpfile erases only 25 bytes from ?mystruct1.buffer'. At line 4, the size of the data pointed by void * pointer 'mystruct1.addr' is unknown. Hence the nul? lify option has been specified to reset the pointer value to NULL. At line 5, the ?mystruct2.addr_size' is specified as size argument to de? termine the size of the data pointed by void * pointer 'mys? truct2.addr'.

Example:

Assuming the following piece of code is from kernel module 'mymodule':

```
struct s1 {
    char *name;
    void *addr1;
    void *addr2;
    char buffer[50];
};
struct s2 {
    void *addr;
    long addr_size;
};
```

```
/* Global symbols */
char array_var[100];
struct s1 mystruct1;
struct s2 *mystruct2;
int foo()
{
    ...
    s1.name = "Hello World";
    ...
}
makedumpfile.conf:
[mymodule]
erase array_var
erase mystruct1.buffer size 25
erase mystruct1.name
erase mystruct1.addr1 nullify
# Assuming addr2 points to 1024 bytes
erase mystruct1.addr2 size 1K
erase mystruct2.addr size mystruct2.addr_size
EOF
```

Loop construct

A Loop construct allows the user to traverse the linked list or array elements and erase the data contents referred by each element.

```
for <id> in {<ArrayVar> |
            <StructVar> via <NextMember> |
            <ListHeadVar> within <StructName>:<ListHeadMember>}
    erase <id>[.MemberExpression] [size <SizeExpression>|nullify]
    [erase <id>...]
    [...]
endfor
```

where

<id> Arbitrary name used to temporarily point to elements of the list. This is also called iteration variable.

<ArrayVar>

A simple expression in the form of <Symbol>[.member[...]] that results into an array variable.

<StructVar>

A simple expression in the form of <Symbol>[.member[...]] that results into a variable that points to a structure.

<NextMember>

Member within <StructVar> that points to an object of same type that of <StructVar>.

<ListHeadVar>

A simple expression in the form of <Symbol>[.member[...]] that results into a variable of type struct list_head.

<StructName>

Name of the structure type that can be traversed using HEAD variable <ListHeadVar> and contains a member named <ListHeadMember>.

<ListHeadMember>

Name of a member in <StructName>, of type struct list_head.

<MemberExpression>

A simple expression in the form of [.member[...]] to specify a member or component of an element in <ArrayVar>, <StructVar> or <StructName>.

<SizeExpression>

Size value in the form of <SizeValue>, <id>[.MemberExpression] or <Symbol>[.member[...]].

The for loop construct allows to iterate on list of elements in an array or linked lists. Each element in the list is assigned to iteration variable <id>. The type of the iteration variable is determined by that of the list elements. The entry specified after 'in' terminal is called LIST entry. The LIST entry can be an array variable, structure variable/pointer or a struct list_head type variable. The set of erase commands specified between for and endfor, will be executed for each element in the LIST entry.

If the LIST entry specified is an array variable, then the loop will be executed for each array element. The size of the array will be deter? mined by using dwarf information.

If the LIST entry specified is a structure variable/pointer, then a traversal member (<NextMember>) must be specified using 'via' terminal. The for loop will continue until the value of traversal member is NULL or matches with address of the first node <StructVar> if it is a circu? lar linked list.

If the LIST entry is specified using a struct list_head type variable, then within terminal must be used to specify the structure name <StructName> that is surrounding to it along with the struct list_head type member after ':' which is part of the linked list. In the erase statement <id> then denotes the structure that the list_head is con? tained in (ELEMENT_OF).

The below example illustrates how to use loop construct for traversing Array, linked list via next member and list_head.

Example:

Assuming following piece of code is from kernel module 'mymodule':

```
struct s1 {
    struct *next;
    struct list_head list;
    char private[100];
    void *key;
    long key_size;
};
/* Global symbols */
struct s1 mystruct1;
static LIST_HEAD(s1_list_head);
struct s1 myarray[100];
void foo()
{
    struct s1 *s1_ptr;
    ...
```

```
    ...
    s1_ptr = malloc(...);
    ...
    ...
    list_add(&s1_ptr->list, &s1_list_head);
    ...
}
makedumpfile.conf:
[mymodule]
# erase private fields from list starting with mystruct1 connected via
# 'next' member:
for mys1 in mystruct1 via next
    erase mys1.private
    erase mys1.key size mys1.key_size
endfor
# erase private fields from list starting with list_head variable
# s1_list_head.
for mys1 in s1_list_head.next within s1:list
    erase mys1.private
    erase mys1.key size mys1.key_size
endfor
# erase private fields from all elements of the array myarray:
for mys1 in myarray
    erase mys1.private
    erase mys1.key size mys1.key_size
endfor
EOF
```

In the above example, the first for construct traverses the linked list through a specified structure variable mystruct1 of type struct s1. The linked list can be traversed using 'next' member of mystruct1. Hence a via terminal has been used to specify the traversal member name 'next'.

The second for construct traverses the linked list through a specified

struct list_head variable s1_list_head.next. The global symbol s1_list_head is a start address of the linked list and its next member points to the address of struct list_head type member 'list' from struct s1. Hence a within terminal is used to specify the structure name 's1' that can be traversed using s1_list_head.next variable along with the name of struct list_head type member 'list' which is part of the linked list that starts from s1_list_head global symbol.

The third for construct traverses the array elements specified through a array variable myarray.

SEE ALSO

makedumpfile(8)