1- mtrace

The mtrace program displays a stack trace for programs that have either crashed or hung.

```
mtrace [-bf] [-p mpeNum] coffFile
mtrace -a address coffFile
```

Switches:

b	display a stack trace of the background task in the pe context.
f	display a stack trace of the foreground task in the pe context.
-a *address*	display the calls stack starting at the specified address.
-p *mpeNum*	specify the mpe for the trace.

Options, usage and output are described in greater detail in the following sections.

1.1- output

Each entry in the stack trace is of the form:

```
NearestLabel+offset "fileName", line n
```

For example:

```
_NuiIdle+00000052 "init.c", line 86
```

This indicates that the PC stored in thecorresponding stack frame points to an instruction in the _NuiIdlefunction at an offset of 0x52 from the start of the function. The corresponding source code line is in the file "init.c" at line 86.

The symbols and line number information are taken from the cof file specified on the command line. It is often useful to try several different cof files if the program makes use ofseveral separately loaded files. For instance, if you are trying to debug the player, you might try the player cof file, the pe cof file and the bios cof file.

While you can use the -p switch tospecify which mpe to trace, the default mpe is 3 and is usually the one wanted.

1.2- Using mtrace

There are basically four ways to run mtrace:

```
mtrace myProg.cof
```

This will display a stack trace of a program that has crashed. It uses the current value of r30 as the head of the stack frame list.

```
mtrace -b myProg.cof
```

This will display a stack trace of the background task in the pe context. It is not useful for bios applications. It uses the stored value of r30 for the background task as the head of the stack frame list. This is useful if the background task has made a call that results in a switch to foreground mode and is hung at that call. This will allow you to figure out which call the background task made that caused it to hang.

```
mtrace -f myProg.cof
```

This will display a stack trace of the foreground task in the pe context. It is not useful for

bios applications. It uses the stored value of `r30` for the foreground task as the head of the stack frame list. This could be useful if the foreground task hangs. In practice, I have never needed this. It is included for completeness.

```
mtrace -a address myProg.cof
```

This will display the calls stack starting at the specified address. The address should be a pointer to a stack frame. This can be useful if the stack gets corrupted and you want to manually examine stack frames. I've used it by starting at the base of the stack and tracing what look like valid stack frames. This requires a knowledge of how stack frames are arranged and inspection of a hex dump of the stack in order to find candidate stack frames. It has the advantage that once you do identify a valid stack frame, you can get a stack trace with symbols and file names without having to look them up in the map file.

Copyright © 2000, VM Labs, Inc., All rights reserved.

Confidential and Proprietary Information of VM Labs, Inc.

These materials may be used or reproduced solely under an express written license from VM Labs, Inc.

Merlin™, Merlin Media Architecture™, and the logo are trademarks of VM Labs, Inc. The information contained in this document is confidential and proprietary to VM Labs, Inc., and is provided pursuant to a non-disclosure agreement between VM Labs, Inc., and the recipient. It may not be distributed or copied in any form whatsoever without the express written permission of VM Labs.

The information in this document is preliminary and subject to change at any time. VM Labs reserves the right to make changes to any information described in this document.