4A-06

64bit カーネルメモリダンプにおける関数引数解析機能の開発

高橋　香穂†　　伊藤　孝之†　　松浦　陽平†

†三菱電機株式会社　情報技術総合研究所

1　背景と目的

Linux には、カーネルがクラッシュした際に、レジスタやスタックの内容をカーネルメモリダンプとして保存する機能がある。例外発生によるシステム停止などの障害発生時には、カーネルメモリダンプを解析し、呼び出された関数やその引数を知ることが原因解明の手がかりとなる。しかし、64bit の Linux カーネルでは、関数引数の大部分をレジスタに格納して渡す呼び出し規約を採用しており、スタックを参照してその値を知ることができず、専門家が手間と時間をかけて解析する必要がある。

そこで、カーネルメモリダンプにおいてレジスタ渡し引数の値を自動的に解析する機能を開発した。本稿では機能の設計と実装について述べる。

2　既存技術と課題

Linux のメモリダンプ解析ツールである crash コマンド[1]では、障害発生までに呼び出された関数とスタックを表示することができる。

この際、関数の引数をスタックに積んで渡すスタック渡しであれば、障害発生時にもスタックを参照して値を知ることができる。

一方、64bit カーネルでは、引数をレジスタに格納して渡すレジスタ渡しによって大部分の引数を渡す。レジスタは関数内の処理によって上書きされるため、障害発生時のレジスタ値から関数の引数の値を求めることができない。

レジスタ渡し引数の値は、デバッグ情報と組み合わせて解析することで解析可能である。ただし、デバッグ情報は DWARF[2]という専用のフォーマットで記述されており、解析には専門知識が必要となる。また、人手で引数を逐一解析する必要があるため、解析に手間と時間がかかる。

これらの課題に対し、障害解析容易化のため、自動的にレジスタ渡しの引数の値を解析し表示する機能を開発している。

Development of a module for function argument analysis using crash dumps of the x86_64 Linux kernel.
† Information Technology R&D Center, Mitsubishi Electric Corporation.

```
01 <1><29c>: 省略番号: 14 (DW_TAG_subprogram)
02    <29d>   DW_AT_external    : 1
03    <29d>   DW_AT_name        : (間接文字列、オフセット: 0x1d2): func
04    <2a3>   DW_AT_type        : <0x65>
05    <2a7>   DW_AT_low_pc      : 0x400610
06    <2af>   DW_AT_high_pc     : 0x6c
07 <2><2bd>: 省略番号: 15 (DW_TAG_formal_parameter)        第1引数
08    <2be>   DW_AT_name        : x1
09    <2c3>   DW_AT_type        : <0x65>
10    <2c7>   DW_AT_location    : 0x0 (location list)
11 <2><2cb>: 省略番号: 15 (DW_TAG_formal_parameter)        第2引数
12    <2cc>   DW_AT_name        : x2
13    <2d1>   DW_AT_type        : <0x65>
14    <2d5>   DW_AT_location    : 0x4c (location list)
                                    (A)
```

(1) 関数のデバッグ情報の一部

```
    記載位置  開始アドレス          終了アドレス          格納場所
01  0000004c 0000000000400610 0000000000400624 (DW_OP_reg4 (rsi))
02  0000005f 0000000000400624 0000000000400668 (DW_OP_reg3 (rbx))
03  00000072 0000000000400668 000000000040066c (DW_OP_reg1 (rdx))
```

(2) 引数の場所情報

図 1 デバッグ情報と場所情報の例

3　引数解析機能の概要

本稿の引数解析機能では、カーネルメモリダンプに加えてデバッグ情報を用いる。デバッグ情報とは、ソースコードと実行ファイルを紐づけるために生成される情報であり、Linux カーネルの実行ファイル(vmlinux)に含まれる。カーネル実行ファイルは軽量化のためデバッグ情報を通常含まないが、その場合でも解析時に同一バージョンのデバッグ情報付きカーネル実行ファイルを用意することで解析が可能である。引数解析機能においては、デバッグ情報(debug_info)とデバッグ情報の補助的な情報の一種である場所情報(debug_loc)の 2 種類を主に使用する。

デバッグ情報の例を図 1(1)に示す。図 1(1)はある関数のデバッグ情報の一部であり、7～10 行目がこの関数の第 1 引数、11～14 行目が第 2 引数の情報を示している。図 1(1)14 行目の(A)は、場所情報において、この引数に関する情報が記載されている位置を示す。図 1(2)は第 2 引数の場所情報であり、1 行目の点線内が(A)と一致している。場所情報は、行ごとに開始/終了アドレスと格納場所の組になっており、プログラムの実行位置と引数の値の格納場所を対応付けている。

例えばプログラムカウンタ = 400614 にて障害が発生した場合、図 1(2)1 行目の開始/終了アドレスの範囲内であるため、レジスタ RSI を参照することで引数の値を知ることができる。

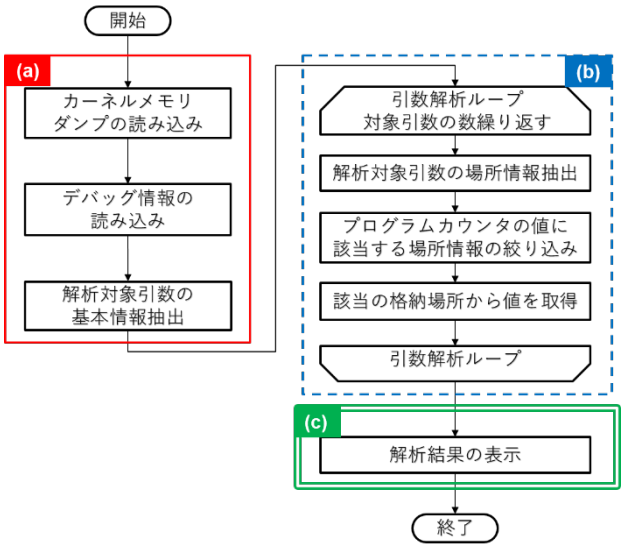

図 2　引数解析機能の処理フロー

4　引数解析機能の設計

メモリダンプとデバッグ情報を入力とする引数解析機能の処理フローを図 2 に示す。各処理の詳細を下記(a)～(c)に示す。

(a) 解析対象引数のデバッグ情報抽出

引数解析機能が前記入力データを読み込み、障害発生時のプログラムカウンタに該当する関数のデバッグ情報を抽出する。この中に解析対象となる引数の名前と場所情報内の記載位置が含まれているため、保持して次以降の処理で使用する。

(b) 引数の解析

この処理は、(a)で抽出した関数の引数の数、繰り返して実行する。各引数に該当する場所情報を抽出し、その中から障害発生時のプログラムカウンタに該当する格納場所を絞り込む。格納場所から引数の値を取得し、解析結果として保持する。

(c) 解析結果の表示

(a)で抽出した引数の名前と(b)で解析した引数の値を対応づけて表示する。解析不可能である場合はその旨を表示する。

なお、本稿では、障害発生時に実行していた関数のみを解析対象としている。障害発生以前に実行された関数についても、先述のデバッグ情報を用いた方式によって解析可能であるが、レジスタの値を復元する処理を別途設計する必要がある。

また、DWARF[2]では格納場所として数十パターンが挙げられているが、本稿ではレジスタに格納されている場合についてのみ実装し、デバッグ情報を用いた引数解析機能の実現性を確認する。

```
crash> bt
PID: 6561   TASK: ffff9b58acd3cf10  CPU: 0 COMMAND: "insmod"
 #0 [ffff9b58ae6bf990] machine_kexec at ffffffffad660b2a
 #1 [ffff9b58ae6bf9f0]   crash_kexec at ffffffffad713402
 #8 [ffff9b58ae6bfc30] do_page_fault at ffffffffadd1a925
 #9 [ffff9b58ae6bfc60] page_fault at ffffffffadd16768
    [exception RIP: map_vm_area+16]
    ( area = 0, prot = 0, pages = 0)
```

(1) 全引数解析可能

```
crash> bt
PID: 8408   TASK: ffff8bd2183e1fa0  CPU: 0 COMMAND: "insmod"
 #0 [ffff8bd1d92bb9a0] machine_kexec at ffffffff93860b2a
 #1 [ffff8bd1d92bba00]   crash_kexec at ffffffff93913402
 #8 [ffff8bd1d92bbc40] do_page_fault at ffffffff93f1a925
 #9 [ffff8bd1d92bbc70] page_fault at ffffffff93f16768
    [exception RIP: bin2hex+28]
    ( dst = 1, src = ..., count = ...)
```

(2) 一部引数解析不可能

図 3　引数解析機能の実行結果

5　引数解析機能の実装と結果

4 章にて設計した引数解析機能の一部を実装した。実装環境は Cent OS 7.5 を使用した。実装において、デバッグ情報変換のために binutils[3]の readelf コマンドを使用しており、解析した引数の表示は crash コマンド[1]のソースコードに一部追記し、表示に組み込む形で実現した。

実装した引数解析機能を使用してダンプ解析した結果を図 3 に示す。図 3(1)の最終行にて引数の解析結果を表示している。本稿では故意にカーネルクラッシュを発生させ取得したダンプを解析している。全ての引数に与えた任意の値が正しく解析されており、デバッグ情報を用いた引数解析機能の実現性が確認できた。

また、図 3(2)において、第 2 引数と第 3 引数が解析不可能であることが確認できる。これらの値は今回実装しなかった格納場所パターンのため、今後の実装追加により解析可能と考える。

6　結論

本稿では、64bit カーネルメモリダンプにおいて、関数に渡された引数の値を機械的に解析する機能の一部を開発し、その実現性を検証した。今後は、追加処理を設計、実装し、より多くのケースに対応する引数解析機能を実現する。

参考文献

[1] Red Had Software, Inc., “crash,” <https://people.redhat.com/anderson/> 2019-12-17 アクセス.

[2] DWARF Standards Committee, “The DWARF Debugging Standard,” <http://dwarfstd.org/> 2019-12-17 アクセス.

[3] Free Software Foundation, Inc., “GNU Binutils,”<https://www.gnu.org/software/binutils/> 2019-12-17 アクセス.