5A-03

64bit カーネルメモリダンプ関数引数解析機能の設計と試作

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

近年では、様々な情報システムに Linux OS が使用されている。Linux では、カーネルパニックが起きた際に出力されるカーネルメモリダンプを解析し、ソースコード中のどの関数がどの順番で実行され、障害が発生したか知ることが出来る。また、スタックを介して渡される関数引数であれば、ダンプから引数の値を取得することが可能である。しかし、Intel x86 系の CPU 向け Linux では、CPU の 64bit 化に伴いレジスタ数が増加し、関数引数の渡し方としてレジスタ渡しが用いられるようになり、解析時に引数の値を容易に取得することが出来なくなった。

そこで我々は、障害解析の容易化を目指し、レジスタ渡し引数の値をカーネルメモリダンプから自動的に解析する機能の開発に取り組んでいる。本稿では機能の設計と試作、その結果について述べる。

2　既存技術と課題

x86 Linux では、関数の引数をスタックに積んで渡す、スタック渡しが採用されている。この場合、Linux のメモリダンプ解析ツールである crash コマンド[1]を使用して、関数のスタックフレームを特定し、所定の位置から引数の値を取得することが可能である。

一方で、x64 Linux では、アプリケーションバイナリインタフェース[2]で定められた関数呼出シーケンスにより、関数引数の大部分をレジスタに格納して渡す。そのため、スタックを参照して引数の値を知ることが出来ない。レジスタに格納された引数の値は、その後の処理内容に応じて格納場所が変わるため、障害解析の専門家が命令コードを読み解き解析する必要がある。解析には後述のデバッグ情報という追加情報を使用可能だが、読解には専門知識が必要である。

これらの課題に対し、我々は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。

Implementation of function arguments tracing module for x64 Linux kernel dumps.
† Information Technology R&D Center, Mitsubishi Electric Corporation.

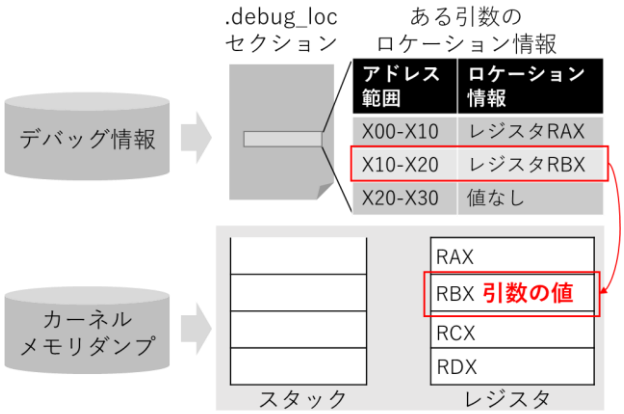

図 1 デバッグ情報を用いる引数解析のイメージ

3　設計

3.1 概要

本稿の関数引数解析機能では、スタックフレームの所定の位置から取得することが出来ないレジスタ渡し引数の値を解析するために、デバッグ情報を用いる。デバッグ情報は、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。通常、カーネル実行ファイルは軽量化のためデバッグ情報を含まないが、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。

デバッグ情報を用いる引数解析のイメージを図 1 に示す。デバッグ情報には、.debug_loc というセクションがあり、関数引数の値が格納されている場所を特定するための情報が記録されている。各引数の情報はアドレス範囲とロケーション情報の組を基本要素として構成されており、解析時のプログラム実行位置を基にロケーション情報を取得することが出来る。これにより、プログラムの実行中に値の格納場所が変化する場合でも解析が可能となっている。

ロケーション情報はデバッグ情報の記述形式である DWARF[3]に則って引数の値を取得するための方法を示している。本稿の関数引数解析機能では、ロケーション情報に基づきメモリダンプから引数の値を取得し表示する。

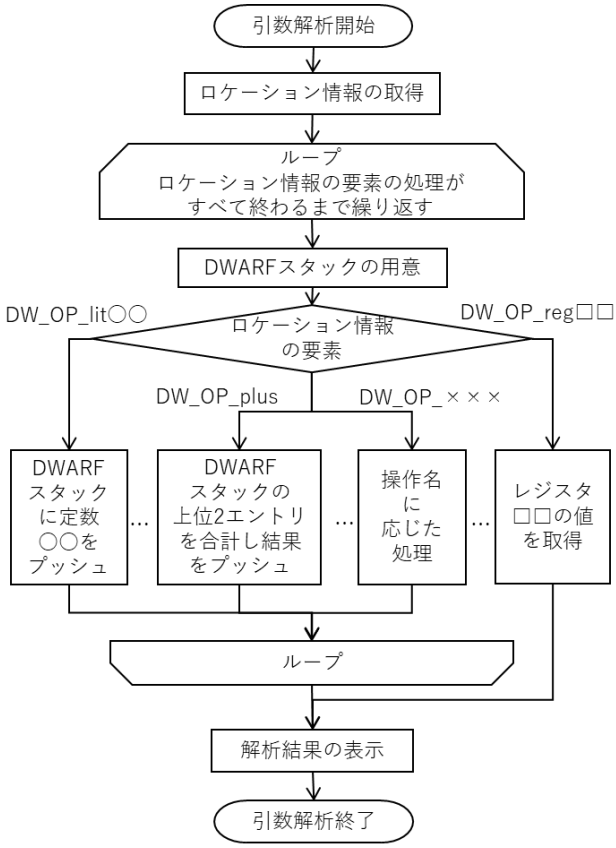

図 2 引数解析の処理フロー概要

3.2 引数解析処理

デバッグ情報を用いる引数解析では、解析に必要な情報を取得し、その内容に沿って解析を実施する。解析対象の引数ごとにその値を解析する処理のフロー概要を図 2 に示す。

始めに、引数に対応するロケーション情報をデバッグ情報から取得する。解析対象の引数について記述した情報を抽出し、解析時のプログラム実行位置を含むアドレス範囲のロケーション情報を取得することで解析に利用する。

次に、取得したロケーション情報に基づき、引数の値を解析する。ロケーション情報は 1 つ以上の要素（DW_OP_から始まる操作名）の組み合わせで表現されており、複数要素である場合は DWARF スタックと呼ばれる領域を利用して値を引継ぎながら連続で操作を実施する。そのため、解析処理内では、ロケーション情報の要素を順に処理していき、最後に DWARF スタックの最上位にある値を取得する。なお、中には引数の値がそのままレジスタにあるケースなど、単一要素である場合もあり、その場合は処理を繰り返すことなくループを抜ける。

最後に、解析結果を利用者に分かるように表示する。以上の処理を引数ごとに実施することで、自動的な関数引数解析が可能となる。

4 試作

3 章にて設計した引数解析機能の内、ロケーション情報の一部を解析することができるプロトタイプを実装した。具体的には、100 種類以上あるロケーション情報の要素の内、レジスタに値が格納されているケース（DW_OP_reg で始まる計 33 種類）に相当する解析処理を実装している。この中に含まれる要素のみで構成される引数については解析結果が、その他の引数については未対応である旨が表示される仕様となっている。

試作において、デバッグ情報変換のために binutils[4]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、表示に加える形とした。

5 結果

4 章にて述べた引数解析機能のプロトタイプを動作させた結果を示す。今回は、動作確認のため、異なる原因で意図的に発生させたダンプを 3 種類用意し、プロトタイプに入力として与えた。これらの入力の内、4 章にて述べた今回のプロトタイプで解析可能である引数の条件に合致するものは 7 つあり、すべてについて引数の値を解析可能であることが確認出来た。

なお、今回のプロトタイプでは解析対象外である引数についても、ロケーション情報に対応した解析処理を実装し組み込むことで解析可能になると考えられる。

6 結論

本稿では、CPU の 64bit 化に伴いレジスタを介して関数に渡されるようになった引数の値を自動的に解析する機能の設計と試作を実施した。今後は、追加の処理を設計、実装し、すべての引数に対応する解析機能を実現する。

参考文献

[1] Red Hat, Inc., “Crash-utility,” < https://www.redhat.com/mailman/listinfo/crash-utility> 2020-12-17 アクセス.

[2] M. Matz, J. Hubicka, A. Jaeger, M. Mitchell, “System V Application Binary Interface, AMD64 Architecture Processor Supplement, Draft Version 0.99.7,” <https://uclibc.org/docs/psABI-x86_64.pdf.> 2020-12-17 アクセス.

[3] DWARF Standards Committee, “The DWARF Debugging Standard,” <http://dwarfstd.org/> 2020-12-17 アクセス.

[4] Free Software Foundation, Inc., “GNU Binutils,” <https://www.gnu.org/software/binutils/> 2020-12-17 アクセス.