平成 26 年度 卒業論文概要			
所　　属	機械情報工学科	指導教員	光來 健一
学生番号	11237009	学生氏名	植木 あずさ
論文題目	LLVM の中間表現を用いた IDS オフロードの開発支援		

1　はじめに

近年、インターネットを通じた不正アクセスが増加しており、不正アクセスを検知するために侵入検知システム（IDS）が用いられている。IDS は攻撃を検知すると管理者に対して通知を行うが、IDS 自体が攻撃を受けると不正アクセスの検知ができなくなってしまう。このような事態を防ぐために IDS オフロードと呼ばれる手法が提案されている。この手法は監視対象システムと IDS を別々の仮想マシン（VM）で動作させることで、監視対象システムに侵入されたとしても IDS への攻撃を防ぐことができる。その一方で、オフロードした IDS は監視対象 VM のメモリを解析して、OS 内の情報を取得する必要がある。しかし、監視対象 VM のメモリへのアクセスは煩雑であり、IDS の開発者の負担となる。

本研究では IDS の開発に LLVM を用い、LLVM の中間表現を変換することで IDS のオフロードを支援するフレームワーク LLView を提案する。

2　IDS オフロード

VM を用いた IDS オフロードは、監視対象システムを VM 上で動作させ、IDS を監視対象 VM とは別の VM（IDS-VM）上で動作させることで、IDS のセキュリティを向上させる手法である。図 1 は IDS オフロードを用いる際のシステム構成である。IDS-VM では IDS のみを動作させ、監視対象 VM 内のシステムの監視のみを行うため、外部からの攻撃を受ける可能性を低くすることができる。そのため、監視対象システムに侵入されたとしても、IDS は攻撃を受けて無効化されることはなく、正常に監視を継続することができる。

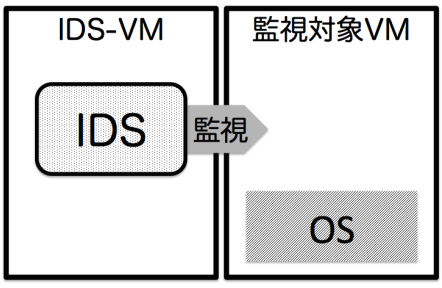

図 1　IDS オフロードのシステム構成

IDS-VM にオフロードされた IDS は監視対象 VM の外から監視対象システムに関する情報を取得することで監視を行う。そのために、IDS は監視対象 VM のメモリを解析して、OS 内の情報を取得する必要がある。例えば、監視対象システムの稼働時間を取得したければ、図 2 のように OS 内の変数 jiffies を基に計算する必要がある。監視対象 VM のメモリを解析するには、まず、参照したいデータのアドレスを監視対象 VM 内の物理的なアドレスに変換する。次に、IDS-VM から監視対象 VM のメモリにアクセスできるようにアドレス変換を行う。このような処理を監視対象 VM 内のデータにアクセスする箇所すべてにおいて手作業で行うのは膨大な労力を必要とし、IDS の開発者の負担となる。

```
t = jiffies / HZ;
```

図 2　OS 内のデータを用いたシステム稼働時間（秒）の取得

そこで、既存の IDS に変更を加えずにオフロードできるようにするためのシステムである Transcall [1] が提案されている。Transcall は監視対象 VM 内の OS をエミュレーションして、OS 内のデータへの煩雑なアクセスを IDS の代わりに行う。そのため、OS の機能を用いる IDS をそのまま動作させることができる。しかし、Transcall の開発には IDS をオフロードに対応させる以上に膨大な労力を必要とする。Transcall では、OS 内のデータにアクセスする際のアドレス変換を 13,000 カ所以上で行っている。その上、Transcall は OS のバージョンごとに開発する必要がある。そのため、Transcall を用いる場合でも IDS オフロードを実現するための労力は大きい。

3　LLView

本研究では IDS の開発に LLVM を用い、LLVM の中間表現を変換することで IDS のオフロードを支援するフレームワーク LLView を提案する。LLView は、IDS が監視対象 VM の OS 内のデータにアクセスしている箇所に自動的にアドレス変換を行うコードを挿入する。これにより、開発者が IDS をオフロードに対応させる際の負担を軽減する。LLView を Linux のソースコードに適用することで、Transcall を半自動生成することが可能となる。

3.1　LLVM の中間表現

LLVM はコンパイラ作成のために必要な機能を提供するコンパイラ基盤である。LLVM はまず、C 言語などのプログラムをコンパイルして中間表現を生成する。次に、中間表現を特定の CPU の機械語に変換する。この時にプログラミング言語や CPU とは独立に最適化を行う。

図 3 は図 2 の C 言語のプログラムをコンパイルして生成された中間表現である。この中間表現では、 1 行目の load 命令で OS 内の 64 ビット整数の変数 jiffies を一時変数の %1 に読み込んでいる。 2 行目の udiv 命令では、その値を HZ の値として定義されている 100 で割り、一時変数の %2 に格納している。 3 行目の store 命令では、その値を 64 ビット整数の変数 t に格納している。

```
%1 = load i64* %jiffies
%2 = udiv i64 %1, 100
store i64 %2, i64* %t
```

図 3　LLVM の中間表現

```
%1 = bitcast i64* %jiffies to i8*
%2 = call i8* @translate(i8* %1)
%3 = bitcast i8* %2 to i64*
%4 = load i64* %3
%5 = udiv i64 %4, 100
store i64 %5, i64* %t
```

図 4　変換後の中間表現

3.2　中間表現の変換

LLVM の中間表現では必ず load 命令を使ってメモリからデータを読み込むため、LLView では load 命令を変換することで IDS をオフロードに対応させる。監視対象 VM のメモリからデータを読み込むようにするには、IDS がメモリからデータを読み込む箇所でアドレス変換を行う必要がある。そのために、LLView はまず、中間表現の中から load 命令を見つける。次に、load 命令で読み込もうとしているデータに対してアドレス変換を行う関数を呼び出す命令を挿入する。そして、変換したアドレスを用いて監視対象 VM 内のデータを読み込むように load 命令を変更する。

図 4 は図 3 の中間表現を LLView が変換した後の中間表現である。1 行目から 3 行目でアドレス変換を行う関数を呼び出している。1 行目では、bitcast 命令を用いて、64 ビット整数の変数 jiffies へのポインタを 8 ビット整数へのポインタにキャストし、一時変数の %1 に格納している。これはどのような変数でも統一的にアドレス変換できるようにするためである。2 行目では、call 命令を用いて、%1 に格納されているポインタ（アドレス）を引数としてアドレス変換を行う translate 関数を呼び出し、返り値を一時変数の %2 に格納している。3 行目では、%2 に格納されているアドレスを元の 64 ビット整数へのポインタにキャストし、一時変数の %3 に格納している。4 行目では、%3 に格納されたアドレスを用いて load 命令を実行している。

translate 関数は別に定義されており、引数で渡された OS 内のデータのアドレスを監視対象 VM 内の物理的なアドレスに変換し、さらに IDS-VM からアクセスできるアドレスに変換して返り値として返す。この関数はアドレス変換ができない場合には、変換を行わずに元のアドレスを返す。これは、load 命令で読み込まれるデータがアドレス変換の対象となる監視対象 VM 内のデータとは限らないためである。例えば、図 2 の変数 t は IDS 内の変数であるため、アドレス変換を行う必要はない。

3.3　Pass を用いた変換

中間表現の変換は LLVM の Pass を用いて実装した。Pass は最適化のために中間表現を変換する仕組みである。LLView の Pass は、load 命令を見つけると読み込みを行おうとしている変数とその型を取得する。そして、それらの情報を用いてキャストを行う bitcast 命令と translate 関数を呼び出す call 命令を生成し、load 命令の前に挿入する。その後、変換後のアドレスからデータを読み込む load 命令を新たに生成し、元の load 命令を削除する。その際に、元の load 命令で読み込んだ値を使っているすべての命令を書き換え、新しい load 命令で読み込んだ値を使うように変更する。図 4 の例では、udiv 命令が使っている %4 が書き換えられている。

4　実験

Linux のヘッダファイルを用いて、監視対象システム内で動作中のプロセスの情報を表示する IDS を作成し、LLView を用いてオフロードに対応させた。この IDS を用いて、正常にオフロード実行ができることを確認する実験を行った。また、LLView による IDS オフロードの性能を調べるために、IDS の実行時間を測定した。実験には Intel Xeon E3-1290、32GB のメモリを搭載したマシンを使用した。仮想化ソフトウェアには Xen 4.4.0 を使用し、監視対象 VM には 2GB のメモリを割り当て、IDS-VM には残りのメモリを割り当てた。それぞれの VM では Ubuntu 14.04 を動作させた。

実験の結果、この IDS を正常に実行することができ、監視対象 VM 内のプロセス ID とプロセス名の一覧を図 5 のように取得することができた。この IDS の実行時間を 10 回計測したところ、その平均値は 1.40 秒であった。

```
root@azu-Precision-T1650:/usr/local/IDS/
plist2# ./plist2 3
0 swapper/0
1 initnit
2 kthreadd
3 ksoftirqd/0
4 kworker/0:0
5 kworker/0:0H
```

図 5　監視対象 VM 内のプロセス一覧の取得

5　まとめ

本研究では、IDS の開発に LLVM を用い、LLVM の中間表現を変換することで IDS のオフロードを支援するフレームワーク LLView を提案した。LLView は、IDS が監視対象 VM の OS 内のデータを読み込むために用いる load 命令の前に、アドレス変換を行うコードを自動的に挿入する。これにより、開発者が IDS をオフロードに対応させる際の負担を軽減することができる。Linux のヘッダファイルを用いて作成した IDS に対して LLView でオフロードに対応させ、正しく監視対象 VM 内の情報を取得できることを確認した。

今後の課題は、LLView を Linux のソースコードに適用し、既存の IDS のオフロードを可能にする Transcall を半自動生成できるようにすることである。

参考文献

[1] 飯田，光来：VM Shadow：既存 IDS をオフロードするための実行環境，第 119 回 OS 研究会 (2011).