VITC: 情報流解析による高安全 C コンパイラ

古瀬 淳
東京大学大学院情報理工学系研究科

1 はじめに

我々は、メモリ脆弱性を突いた攻撃を受けても、一定の安全性を保ちながら動作を継続する実行プログラムを生成する新しい C 言語のコンパイル方法を実装した。

メモリ安全な C 言語コンパイル技術を使い、不正なメモリアクセスを検知し、バッファオーバーフローを回避しつつプログラム実行を継続す方法は既に存在する。しかし、その継続後の実行が安全かどうかは、その「安全」の定義も含め、あまり議論されてこなかった。

そこで我々の枠組では、情報流解析による機密保持性を導入し、高機密性データが低機密領域に流出しないことを保証することで、通常動作時だけでなく、攻撃を受けた後の実行安全性をも保証した。

2 関連研究

2.1 高安全 C コンパイラ

C 言語プログラムでは、言語仕様自体に起因するメモリ管理の不備による、バッファー・オーバーフローなどのセキュリティーホールが度々指摘されている。この問題に対して、言語研究の分野からはメモリ安全コンパイラの提案が行われてきた [3]: これらのコンパイラでは各メモリアクセスが適正である場合にのみアクセスを許可しないよう言語仕様を拡充し、実際にコンパイラはメモリアクセスにチェックコードを埋め込んだ実行プログラムを生成する。不正なメモリアクセスがあった場合は実行を中断する。これらのコンパイラにより、多くの既存 C アプリケーションが変更無しにメモリ安全なプログラムにコンパイルできる。

エラー忘却型計算 [1] では、このアイデアをさらに押し進め、不正メモリアクセスを検知した後にもなんとか実行を継続させる。不正アクセスをそのまま実行するのは危険であるため、例えば、不正書き込みは単純に無視するなどの回避を行う。この方式は一見無謀に見えるが、多くのプログラムを案外「それとなく」実行できることが報告されている [1]。ただし、その継続された実行が安全な物であるかは議論されていない: エラー忘却型計算が行う適当な実行継続はプログラマが想定しているプログラムの意味と大きくかけ離れ、誤って機密情報を攻撃者に開示してしまう危険性がある。

2.2 情報流解析

情報流解析 [4] では主に型システムによって情報の流れを表現し、追跡することでプログラムの機密漏洩を起こす可能性を判定する。この情報流解析を実用的な言語に適用した例では Java や ML の拡張が知られている [2, 5]。それに対して、C 言語で書かれたプログラムの多くで情報漏洩が発生し、大きな問題になっているにもかかわらず、我々の知る限り C 言語での情報流解析の例は無いようである。その理由は、C 言語自身が提供するメモリ管理は不完全でかなりの部分がプログラマの責任に任されており、不正なメモリアクセスが起こりうる状況において情報流を追跡することはほぼ不可能だからである。

また、この分野の全体の傾向として、既存のソフトウェアシステムに対して情報流解析を行い、安価に高安全なプログラムを生成する研究はあまり行われていない。既存ソフトウェアは、そもそも情報機密度の概念がない言語で、情報流を意識せず書かれているため、しばしばそのままでは情報流安全なプログラムにコンパイルすることは不可能である。既存システムに対して情報流解析による安全性を提供するのであれば、コンパイラはこの点も考える必要があるだろう。

3 VITC の主張

高安全 C コンパイラのそもそもの動機は C ソフトウェアシステムの機密情報が不正に攻撃者に漏洩してしまう問題を解決することにある。C 言語ではメモリ管理が不十分であるため、不正メモリアクセスが攻撃の主要手段であり、メモリ安全コンパイラはこれを不可能とする。しかし、メモリ安全といえども機密を漏洩してしまうプログラムは存在する。よって、さらなる高安全性を C プログラムに求めるならば、第一の動機である機密漏洩自体を防止できる、情報流解析を行う高安全コンパイラが求められるはずである。

逆に、情報流解析の視点から見れば、メモリアクセスの仕様を安全にしたメモリ安全コンパイルを仮定すれば、情報流の追跡が可能となる。ただし、単純なメモリ安全性では不十分である:

```
printf("hello ");
if(h){ array[x] = 10; }
printf("world");
```

上の例では h は高機密な変数であり、その値は外部に漏れてはならない。もし h が真であった場合は配列アクセスが行われるが、もし添字 x が不正である場合、[3] で

VITC: Information Secure C Compiler
Jun FURUSE
Graduate School of Information Science and Technology, University of Tokyo

はプログラム実行が中断されてしまう。プログラムの実行中断は多くの場合(この場合、二つ目の printf が実行されるかどうかで)外部から観測可能であり、中断された場合はその事から h が真であったことが漏洩してしまう。プログラムを中断させないためには何らかのエラー処理が必要であるが、メモリアクセスは普遍的にプログラム中に存在する上、それらのうち多くの安全性は静的にはまず検証できないため、そのエラー処理をプログラマに書かせるとすれば、それは恐ろしい労力となるだろう。これを自動的に行うとすれば、エラー忘却計算が必要となる。プログラムはエラー忘却計算によって奇妙な動きをするかもしれない。しかし、情報流解析による安全性により、そのような状況でも、機密情報が攻撃者に漏れる事は防ぐことができる。

以上が我々の高安全コンパイラ VITC の基本的アイデアである。以降は簡単にではあるが、VITC の情報流型システムについて説明する。

4 VITC の型システム

メモリ安全性を仮定した C は、かなり宣言的ではあるものの関数型言語に酷似した言語である。そのため、VITC の情報流型システムは基本的に ML そのそれ [5] を踏襲する。各機密度は束を形成し、C の型に付加される。例えば $\texttt{int}^H$ は高 (High) 機密度の整数、$\texttt{int}^H\ {*}^L$ は高機密度整数を指す、値自体は低機密 (Low) なアドレスの型である。束の半順序によって型のサブタイプ関係が定義され、関数は情報流に関する多相型を持ち、HM(X)[6] で説明されている制約型システムおよび推論を適用できる。さらに、VITC では C 言語の既存ソースが情報流を意識して書かれておらず、そのままでは静的に情報流型安全なプログラムとして型検査できない場合に備えて、静的型を隠蔽し、動的検査を行うために dynamic type を導入している [7]。

C 言語で特徴となるのは型キャスト周辺での情報流型の扱いである:

$\texttt{int}^{H*L}$ `p; int x = (int)p; int* q = (int*)x;`

上ではポインタ p は $\texttt{int}^H\ {*}^L$ という型を持つとプログラマによって指定されており、コンパイラは残りの値の型を推論しようとしている。[7] では x は p の持つアドレス値が代入されるため $\texttt{int}^\alpha$ where $L \leq \alpha$ という型を持つが、p の指す中身の機密度 H のことは忘れてしまう。そのため、この値をふたたびキャストにより int *へと戻した場合、指示先の機密度が不明であるため、q = $(\texttt{int}^{\texttt{H}}\ *)$x という明示的な実行時検査が必要となり、プログラムの修正の必要と実行時オーバーヘッドが問題であった。

この問題を解決するため、我々の最新の型システムは、キャストが影響を与えるのは C 本来の型の部分のみで、情報流型は影響を受けないように改良された。上の例の x の型は、この型システムでは $\cdot^H\ \texttt{int}^\alpha$ where $L \leq \alpha$ となって、H は失われない。そのため、最後のキャストにおいても q の型は $\cdot^H\ \texttt{int}^\beta$ where $\alpha \leq \beta$ と自動的に推論できる。より一般的にはこの新しい型システムでは情報流型 τ は、C 本来の型の部分 (int など) を無視すれば、機密度 ℓ と機密度変数 α の無限リストと見なされる:

$$\lambda ::= \ell \mid \alpha \qquad \tau ::= \tau * \lambda \mid a \mid \lambda^\omega$$

ここで a は任意の τ へと具体化可能な型変数、λ^ω は一定の機密度を持つ無限リスト $\ldots * \lambda * \ldots * \lambda$ を表す。このアイデアは我々は [7] で問題となっていたキャストによる明示的動的検査の必要性を大幅に減らすことができ、既存の C アプリケーションを安価に安全化するために必要不可欠である。

5 結論

VITC コンパイラではメモリ安全 C コンパイルと情報流解析を組み合わせることで、安価に既存 C アプリケーションを安全化する手法を提案している。情報流解析されたエラー忘却計算により、VITC でコンパイルされたプログラムはメモリ脆弱性攻撃を受けた後も、プログラム中の機密情報を漏洩することなく安全な継続動作を行うことができる。

参考文献

[1] Martin Rinard et al. Enhancing server availability and security through failure-oblivious computing. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation San Francisco, CA*, December 2004.

[2] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Symposium on Principles of Programming Languages*, pages 228–241, 1999.

[3] Yutaka Oiwa, Tatsurou Sekiguchi, Eijiro Sumii, and Akinori Yonezawa. Fail-safe ANSI-C compiler: An approach to making C programs secure (progress report). In *Lecture Notes in Computer Science*, volume 2609, Fev 2003.

[4] A. Sabelfeld and A. Myers. Language-based information-flow security. In *IEEE Journal on Selected Areas in Communications, 21(1), 2003.*, 2003.

[5] V. Simonet. Flow Caml in a nutshell. In *Proceedings of the first APPSEM-II workshop*, pages 152–165, March 2003.

[6] Martin Sulzmann, Martin Odersky, and Martin Wehr. Type inference with constrained types. In *Fourth International Workshop on Foundations of Object-Oriented Programming (FOOL 4)*, 1997.

[7] 古瀬 淳. VITC: 対攻撃耐性コード生成コンパイラ. 第 4 回ディペンダブルソフトウェアワークショップ (DSW'06-2). 2006.