M-088

バッファオーバーフロー発生要因の特定方法

A method for identifying the cause of buffer overflow

岩村　誠†　　冨士　仁†
Makoto Iwamura　Hitoshi Fuji

１．背景

ネットワークに接続される端末数の増加により，端末に潜む脆弱性が，ネットワーク経由で悪用されるケースが増えている．こうした脆弱性の中でも，境界チェックを怠ることにより発生するバッファオーバーフローは，サービス停止（DoS：Denial Of Service）攻撃，さらには不正侵入の手段として悪用されるため，対策が急務である．

このような端末の脆弱性がもたらす問題が深刻化する中，適用されていない修正パッチを効率的に調査するツールや，修正パッチを自動的に適用するツール等も開発されており，OS やアプリケーションのパッチマネジメントは改善されつつある．しかし，パッチマネジメントの改善によってもたらされるものは，あくまで既知脆弱性の対処を迅速に行えることであって，昨今，問題視されている 0-day Exploit と呼ばれる未知脆弱性を狙う攻撃には効果がない．

そうした状況を踏まえ，バッファオーバーフローの脆弱性に関しては，CPU や OS，コンパイラなどの開発・実行環境で対策を行うアプローチが広まりつつある．これにより，既知脆弱性は勿論，未知脆弱性を狙ったバッファオーバーフロー攻撃による不正侵入から端末を守ることが可能となってきた．

２．従来技術とその問題点

２．１従来技術

開発・実行環境でのバッファオーバーフロー対策は，これまで様々なアプローチで研究されてきた．ここでは，そのうちの２つを紹介する．

・ SSP(Stack Smashing Protector) [1)]

拡張されたコンパイラにより，ソースコードを再コンパイルすることで，局所変数とフレームポインタの間にカナリアと呼ばれる値が挿入されたプログラムを生成する．バッファオーバーフローによりリターンアドレス等が書き換わっていれば，必ずカナリアの値も変更されているため，関数から戻る前（リターンアドレス等が使われる前）にカナリアをチェックすることで，攻撃を防ぐことができる．

・ Libsafe [2)]

動的リンカの機能を利用することで，バッファオーバーフローを引き起こしやすい標準コピー関数(strcpy など)が呼ばれる前に制御をフックし，コピー先のバッファが十分かどうかコピー前にチェックすることで，バッファオーバーフローの発生を検知する．

† 日本電信電話株式会社
ＮＴＴ情報流通プラットフォーム研究所

２．２問題点

開発・実行環境でのバッファオーバーフロー対策は，バッファオーバーフローの発生を検知すると，攻撃対象となったプロセスを強制終了させてしまうため，根本的な解決はプログラムを修正することになる．しかし，現状の開発・実行環境でのバッファオーバーフロー対策だけでは，オーバーフローが発生したことを検知できても，それが既知の脆弱性を狙われたものなのか，未知の脆弱性を狙われたものなのか，既知ならば実施すべき対処は何なのか，未知ならば修正すべき箇所はどこなのか，を判断することができない．

３．バッファオーバーフロー発生要因の特定方法

発生したオーバーフローが既知の脆弱性を狙われて生じたかどうか判断するには，バッファオーバーフローの発生要因を一意に特定する必要がある．ここでは，そのために必要な発生要因の識別情報と，その抽出方法を提案する．

３．１発生要因の識別情報

一般的にプログラム中のデータ内容は実行環境によって大きく変化しても，命令やデータ構造は実行環境が変わってもほとんど変化しない．そこで，本稿ではバッファオーバーフローの脆弱性を一意に特定するために，以下の３つをバッファオーバーフロー発生要因の識別情報とする．

(1) オーバーフローしたバッファを確保した命令
(2) オーバーフローさせた命令
(3) バッファ確保からオーバーフローまでの関数呼出履歴

バッファオーバーフローが発生した時点で，これら発生要因の識別情報を，既知脆弱性情報としてデータベースなどに格納し，再度バッファオーバーフローが発生した際に，この既知脆弱性情報と照らし合わせることで，既知脆弱性か否かの判断を行うことが可能になる．

３．２識別情報の抽出方法

識別情報の抽出は，従来のバッファオーバーフロー対策と連動して行うが，SSP はバッファオーバーフローが発生し終えた後にオーバーフローを検知するため，上記の識別情報を抽出することが困難である．これに対して Libsafe 等の標準コピー関数呼び出しをフックすることで，バッファオーバーフローを検知する技術は，バッファオーバーフローが発生する直前で検知可能で，上記の識別情報を抽出可能なため，本手法ではこの検知技術と連動して識別情報を抽出する．

(1) オーバーフローしたバッファを確保した命令

スタック上のバッファは，バッファを局所変数として宣言した関数のエントリポイントで確保される．よって，ここではバッファを確保した命令を抽出する代わりに，バッファを局所変数として宣言した関数を特定する．具体的には，バッファオーバーフローを検知した後，次のようにしてバッファを確保した関数を特定する．

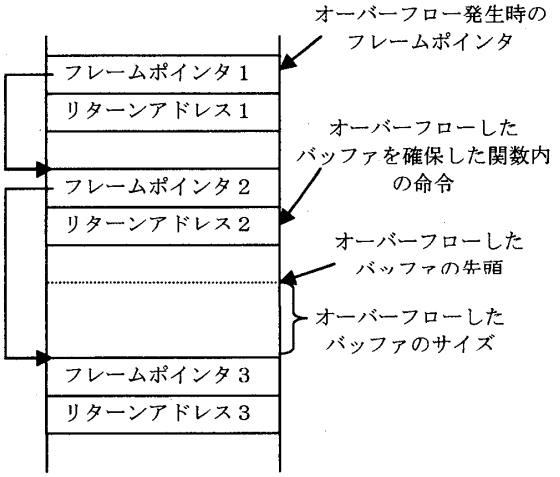

図１　オーバーフロー発生時のスタック内容

まず図１に示すように，オーバーフロー発生時のフレームポインタ１からオーバーフローしたバッファの先頭をまたぐまでスタックトレースを行う．最終的にたどりつくフレームポインタ３の直前のフレームポインタ２に近接するリターンアドレス２は，オーバーフローしたバッファを確保した関数内の命令を指している．このリターンアドレスとコンパイラ等が出力するデバッグ情報を照らし合わせることで，オーバーフローしたバッファを確保した関数を特定する．

またヒープ上バッファに関しては，ヒープメモリ確保用関数がバッファを確保した命令となる．そこで，本手法では事前にヒープメモリ確保用関数をフックしておき，ヒープ確保用関数が呼ばれる度に，スタックトップまでの関数呼出履歴を保管しておく（図２参照）．

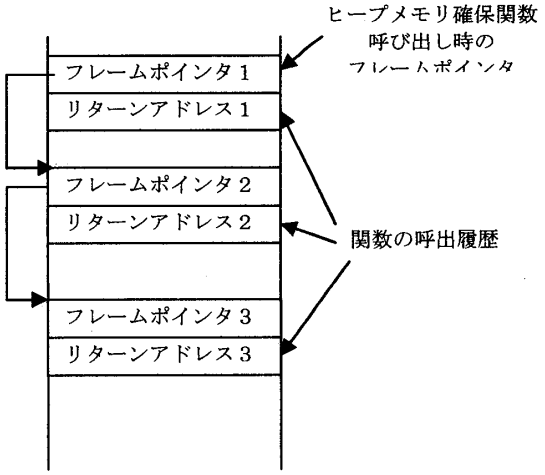

図２　ヒープメモリ確保用関数呼出時のスタック内容

そして，バッファオーバーフローの発生を検知した後に，オーバーフローしたバッファに対応する関数呼出履歴を抽出し，これをヒープ上バッファを確保した命令として扱う．

(2) オーバーフローさせた命令

バッファオーバーフローを検知した際に，実行中のフック関数から対応する標準コピー関数を求める．この標準コピー関数がオーバーフローさせた命令となる．

(3) バッファ確保からオーバーフローまでの関数呼出履歴

バッファオーバーフローを検知した際に，実行中のフック関数から，バッファを確保した関数までスタックトレースを行い，それぞれのフレームポインタに隣接するリターンアドレスを抽出し，これらのリターンアドレスとコンパイラ等が出力するデバッグ情報を照らし合わせることで，関数の呼び出し履歴をもとめる．

４．まとめ

端末の脆弱性がもたらす問題が深刻化する中，様々なバッファオーバーフロー対策が研究されてきた．しかし，従来のバッファオーバーフロー対策では，その発生要因を特定することはできなかった．そこで本稿では，バッファオーバーフローの発生要因を一意に特定するために必要な識別情報を決定し，それら識別情報の抽出方法を提案した．これによって，既知の脆弱性を狙われたのか，未知の脆弱性を狙われたのか，既知ならば実施すべき対処は何なのか，未知ならば修正すべき箇所はどこなのか，を容易に判断できるようになった．

今後は，これまでに発生したバッファオーバーフローの脆弱性を調査し，本手法で用いた識別情報によりバッファオーバーフローの発生要因を一意に特定できるか評価を行っていく．

また，ある脆弱性が修正された結果，本手法では別々に識別される複数の脆弱性が，まとめて解決する場合も考えられる．これら解決した脆弱性に関しても，既知脆弱性として取り扱うべきと考えられるため，開発者に修正方法等を既知脆弱性情報に反映させ，その修正方法を踏まえて既知脆弱性か否かを判断する方法についても，検討を進めていく．

５．参考文献

1) Hiroaki Etoh. GCC extension for protecting applications from stack-smashing attacks.http:// www.research.ibm.com/ trl/ projects/ security/ ssp/.

2) A. Baratloo, N. Singh, and T. Tsai, December 1999.Libsafe: Protecting Critical Elements of Stacks. http:// www.research.avayalabs.com/project/ libsafe/ doc/ libsafe.pdf.

3) 岩村 誠, 柏 大, “バイナリプログラムにおけるバッファオーバーフロー攻撃検知法と攻撃痕跡抽出法”, 情報処理学会研究報告 2004-CSEC-24, pp187-192, 2004