L-032

非制御データ攻撃に対する緩和策
Mitigation Scheme against Non-control Data Attacks

李 茂新† 大塚 玲†
Li MaoXin Akira Otsuka

1. はじめに

制御フロー攻撃は，防御メカニズムが絶えず改善されているため，悪用がますます困難になっている． したがって，攻撃者の目的は，ユーザーの識別情報，構成情報，およびその他のデータを利用することに移行している．すなわちプログラムの制御フローを変更しなくても攻撃を達成できる．非制御データ攻撃を探索している，既に非制御データ攻撃に関する研究はアイデアの形成，実用化，半自動利用フレームワークの開発，そしてチューリングの完全な証明までの 4 段階を経た，これまでのところ DOP[1]攻撃などの非制御データ攻撃から防御するための有効な防御メカニズムは存在しない．

2. 非制御攻撃

Chen[2]は，非制御データ攻撃が制御データ攻撃と同じくらい有害であることを最初に提案してから，徐々に注目を集めている．

2.1 非制御データ攻撃の進化

文献[3]では，データ指向のエクスプロイト(Data-Oriented Exploits, DOE)の全体的な枠組みを体系的に提案している．文献に提案されているデータフローステッチング技術（Data-flow Stitching）は，非制御データ攻撃のプロセス全体を自動的に実装し，セキュリティ関連のデータフローフラグメントを自動的にフィルタ処理し，セキュリティデータを汚染することによって，攻撃を達成する．攻撃プロセス全体は，ほとんどの ASLR[4]，CFI[5]および DEP[6]をバイパスする．

コード再利用攻撃 ROP と同様のチューリング完全性を証明した，非制御データ攻撃のチューリング完全性を初めて証明し，特定のデータや関数に依存しない攻撃フレームワークを構築している(Data-oriented Programming, DOP)[1]．メモリアドレスを漏洩しなくても ASLR をバイパスし，メモリページのパーミッションフラグを変更できるため，コードインジェクション攻撃が再び有効になる．

```
1  struct server{int *cur_max, total, typ;} *srv;
2  int quota = MAXCONN; int *size, *type;
3  char buf[MAXLEN];
4  size = &buf[8]; type = &buf[12]
5  ...
6  while (quota--) {
7    readData(sockfd, buf);       // stack bof
8    if(*type == NONE ) break;
9    if(*type == STREAM)
10       *size = *(srv->cur_max);
11   else {
12       srv->typ = *type;
13       srv->total += *size;
14   } //...(following code skipped)...
15 }
```

図 1 DOP 攻撃の対象となるプログラムの例

†情報セキュリティ大学院大学, Institute of Information Security

DOP は最小限の言語 MINDOP(Minimal Language DOP)をシミュレートできる．MINIDOP は x86 命令のもつ 6 つの基本操作に統合でき，Turing 完全性を満たすことが証明できる．従って，DOP 攻撃も Turing 完全であることが示せる．また，DOP 攻撃で使用する DOP ガジェットの入出力は，全て C 言語におけるポインタ操作を通じて行われ(図 1)，ASLR 等のアドレスランダム化によってアドレスが変化しても，攻撃に必要な関数アドレスはランダム化後のアドレス値で参照されるため，ASLR 等のランダム化手法は DOP 攻撃に対して無力である．

3. 提案アプローチ

非制御データ攻撃に関する関連資料の調査では，非制御データ攻撃が成功するために，ほぼ全ての場合でプログラム関数内のポインタの改ざんが必要であった（図 2）．そのため，仮に攻撃者が脆弱性を介して攻撃を仕掛けても，ポインタを有効に保護できれば，ポインタを有効に活用できないため，攻撃の期待を達成することは困難になると考えられる．

非制御データ攻撃に対する既存のアプローチを分析すると，ほとんどがプログラムのソースコードを分析しており，かつ，分析対象のソースコードには一定の制限があることがわかる．無制限のバイナリプログラムに対するポインタの保護方式を検討することは価値がある．

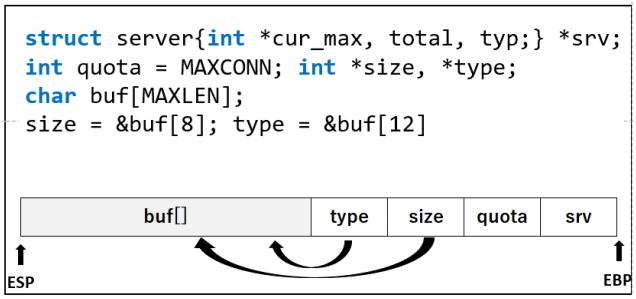

図 2 非制御データ攻撃でのポインタ改ざんの例

配列 buf がバッファーオーバーフロー等で配列境界を超えて値が入力されると，プログラムコード中の命令に依らず type, size 変数の値が変更される．

3.1 Intel Pin Tool

Pin とは，Intel Corp. が開発した動的なバイナリ分析フレームワークで，Windows や Linux プラットフォーム向けの Pin Tool と呼ばれるプログラム分析ツールが提供されている．同ツールは，実行中のプログラムの動作を監視および記録できる．プログラム実行に関数多くの重要な情報(正確性，パフォーマンス，セキュリティなど) を効果的に分析できる．

Pin Tool は，プログラムの特定の場所にコードを挿入できる．通常は，特定の命令または関数の直前か直後にコー

ドを挿入する．たとえば，メモリリークを検出するのであれば，すべての動的メモリ割り当てを記録して検査すれば良い．

3.2 関数範囲内のポインタ保護

静的解析方法はソースコードを必要とするため，オープンソースではない商用プログラムには適用できない．バイナリプログラムに基づく動的解析法はこの問題をうまく解決できる．動的バイナリ分析フレームワーク Pin を用いれば，動的解析よって関数範囲内でポインタの宣言/参照などの命令を検出できる．ポインタ値をスタックに格納する命令が実行されたタイミングで，シャドウメモリ中にポインタ値を退避し，同関数範囲内で同スタックアドレスを参照する命令を実行する直前に，スタック上のポインタ値と退避していたポインタ値と比較してポインタ値の改ざんの有無を検知する．バッファーオーバーフロー等によるプログラム外からの改ざんが検出されれば，本来の値を復元することにより非制御データ攻撃を緩和する．

3.3 対応手順

関数範囲内のポインタを保護する具体的な手順は次の通りである．

1. 動的解析でポインタ宣言命令を検出して，当該ポインタ値が当該関数スタック範囲内であれば，そのポインタ値をシャドウメモリに格納する．
2. 関数範囲内でポインタ参照の際は，そのポインタ値とシャドウメモリ内で格納した値を比べて確認する．攻撃者が脆弱性を介してポインタデータを上書き（異常書き込み）すると，本来の値を復元する．攻撃者による非制御データへの攻撃を効果的に軽減する．
3. 当該関数の処理が完了し呼び出し元に復帰する際に，シャドウメモリ中のポインタ値を削除する．

4. 評価

4.1 実験環境

以下の実験環境で，関数範囲内のポインタ改ざん攻撃を防御する動作を確認した．

Ubuntu 18.04.01 32bit
Intel Pin 3.4
GCC Version 7.3.0

4.2 非制御データ攻撃の適用と提案動作の確認

本論文では，提案方式である関数範囲内のポインタを保護の評価のため，Stack-based Buffer Overflow を悪用する DOP 攻撃を防御することができるかどうかの実験を行った．

今回，実験には，DOP 攻撃をシミュレーションため，脆弱性があるプログラム（図３）を使用した．このプログラムは外部データによって制御フローを制御することで DOP 攻撃を構築することができる．

このプログラムを Intel Pin ツールを用いて，起動した上で，攻撃を行ったところ，防御機能対応した Pin Tool により，攻撃は成功しなかった．つまり，攻撃を防げた．

```
void fn(){

        struct server{int *cur_max,total,typ;} *srv;
        int quota = MAXCONN;int *size,*type;
        //...
        char buf[MAXLEN];
        size = (int*)&buf[8];type = (int*)&buf[12];

        while(quota--){

                gets(buf);

                if(*type == NONE) break;
                if(*type == STREAM)
                        *size = *(srv->cur_max);
                else{
                        srv->typ = *type;
                        srv->total += *size;
                }

        }
}
```

図３ 実験で使用した DOP 攻撃対象プログラム例

配列 buf がバッファーオーバーフロー等で配列境界を超えて値が入力されると，局所変数 type, size, quota, srv の値が改ざんされ、while ループの継続条件，if 文の分岐条件，ポインタ参照先メモリ値の更新値を，buf への入力値から自由に制御できる例になっている．実験では, size, type の 2 つのポインタの保護を試みた．

5. まとめ

バイナリの動的解析に基づいて実行時に，指定された命令に対して Pin Tool による操作を実行することによってプログラムの実行コンテキストを正確に保護することができることを確認した．今後，DOP 論文の実例環境で実験すると理論的な解析を重ねて，様々な攻撃を防ぐ有効な戦略になっていることを確認する．

参考文献

[1] H. Hu, S. Shinde, S. Adrian, Z. Leong Chua, P. Saxena, and Z.Liang, “Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks,” IEEE Symposium on Security and Privacy (SP’16), pp. 969-986, 2016.

[2] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, “Non- control-data attacks are realistic threats,” USENIX Association, in Proc of the 14th Usenix Security Symposium(Usenix Security’05),Vol.53, pp. 12-12. 2005.

[3] H. Hu, Z. L. Chua, S. Adrian, P. Saxena and Z. Liang, “AutomaticGeneration of Data-Oriented Exploits,” USENIX Association, inProc of the 24th Usenix Security Symposium(Usenix Security’15),pp. 177-192. 2015.

[4] S. Bhatkar, D. C. Duvarney, and R. Sekar, “Address obfuscation:an efcient approach to combat a broad range of memory error exploits,”Proceedings of Usenix Security Symposium(Usenix Security’03), pp. 105-120, 2003.

[5] N. Carlin,A. Barresi,D. Wagner, M. Payer, and T. R. Gross.“Control-Flow Bending: On the Effectiveness of Control-Flow Integrity”in Proc of the 24th Usenix Security Symposium(UsenixSecurity’15), pp.161-176, 2015.

[6] S. Andersen, “Changes to Functionality in Microsoft Windows XP Service Pack 2 Part 2: Network Protection Technologies.” Microsoft Technical Document, 2004.