計算機言語システム論 (12/13)

米澤研４年　佐藤秀明

概要

- David Brumley and Dawn Song. “Privtrans: Automatically Partitioning Programs for Privilege Separation”.
  - Privilege separation とは
  - 本研究のアプローチ
  - 実装の詳細
  - 実験・評価
  - 議論・その他
  - まとめ

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

特権つきのプログラム実行

- 特権がないと動かないプログラムはけっこう多い
  - setuid/getuid (e.g., ping)
  - ネットワークのデーモン（e.g., ウェブサーバ）
  - システムメンテナンスプログラム（e.g., cron)
- 特権をもつことには危険も伴う
  - 脆弱性があるとシステムの広範囲を exploit されてしまう
- 本当に必要な場合にのみ特権を与えるようにしたい

Privilege Separation

- プログラムを 2 つに分割する
  - Monitor: 特権が必要なすべての操作を行う
    - 上級の権限で動かす
  - Slave: それ以外の操作を行う
    - 一般の権限で動かす
- 普通の処理は slave が行い、特権が必要な処理のみ monitor を通じて実行する
- Monitor と slave はプロセス間通信やネットワークソケットを用いて連絡を取り合う

Privilege Separation の利点

- Monitor と slave はそれぞれ独立のプロセスで動く
  - Slave で buffer overflow が起きても、それが直接 monitor を侵すことには繋がらない
- Monitor は特権の必要な操作と slave との間に立つ
  - slave に対して、許される操作のみをインターフェイスとして提供できる
  - ユーザが自由に定義したポリシーを slave に対して強制できる

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

既存の研究

- System call interposition (Berman et al. 他多数)
  - システムコールの呼び出しを規制
  - 一般の関数呼び出しには対応していない
- Manual privilege separation (Provos et al.)
  - がんばって手作業でプログラムの分割を行った
  - いくつかの攻撃を防ぐことに成功
- Privman: a library for partitioning application (Kilpatrick)
  - ある操作の許可 / 抑止を判定する機能しかない
  - ソース中の対応するすべての操作を手動で書き換えなければならない

本研究の貢献

- Privilege separation を自動で行うツールの作成
  - ユーザが元ソースコードに少量の annotation を加える
    - 何を特権的に扱うかはユーザが自由に決定できる
    - 特権的に扱いたい関数や変数に「特権属性」の印を付ける
  - 静的な解析を用いて「特権属性」の伝搬を計算する
  - monitor と slave の 2 つのソースコードに分割される
- プログラム分割後の実行効率を改善
  - 静的・動的な解析を行い、monitor ～ slave 間の不必要な通信を抑える
- 分散環境への対応
  - Monitor と slave がそれぞれ異なるホストに存在できる

変数に対する特権属性

- 変数に対するアクセスを制限する機能だけでは不十分な場合がある
  - 例 1: 秘密鍵にアクセスする権限を持つ関数は、その鍵を外部にリークできてしまう
  - 例 2: ファイルディスクリプタそのものを返してしまうと、それがどう使用されるかについて制御できない
- 「特権属性」をもつ関数に由来するデータは、monitor の中に隠す
  - 返り値そのものを slave へは返さないようにし、代わりに返り値が格納される monitor 中のテーブルへのインデクスを返す

システム作成に必要なコンポーネント

1. 特権属性をもつ資源 (関数、変数) を検知するしくみ
   - 少ない annotation をソースコード全体に伝搬させる
   - 特権属性のついた関数の返り値にも特権属性をつける
2. monitor ～ slave 間の RPC(Remote Procedure Call) を実現するしくみ
   - データの marshaling/demarshaling
3. 特権的な関数の返り値を monitor 内に保持しておくためのしくみ
   - 以後の monitor への関数呼び出しで必要になる場合

ユーザが加える annotation

- C 言語の type qualifier として、“ priv” と “ unpriv” を用意する
  - “priv” は monitor で実行させたい関数や、その関数からの返り値で初期化される変数につける
  - “unpriv” は、変数の特権属性を失わせたいときにつける
- ツールは特権的な操作間の依存性を推論し、特権属性を必要に応じて伝搬させる
- 特権属性のついた変数や関数は monitor 内に配置される

```
int __attribute__((priv)) a;
__attribute__((priv)) void f();
```

特権操作の列に関するポリシー

- 特権操作の列がユーザのポリシーを満たすように、実行時にチェックする場合がある
  - フロー解析を用いて、操作の列に関するオートマトンを自動的に作成し、プログラムに組み込む (Chen et al.)
- 本研究ではユーザが操作の列に関してどうポリシーを指定するかまでは踏み込まない
  - monitor のソースコードにポリシーを書き加えるのは容易であろう

データの特権を格下げする

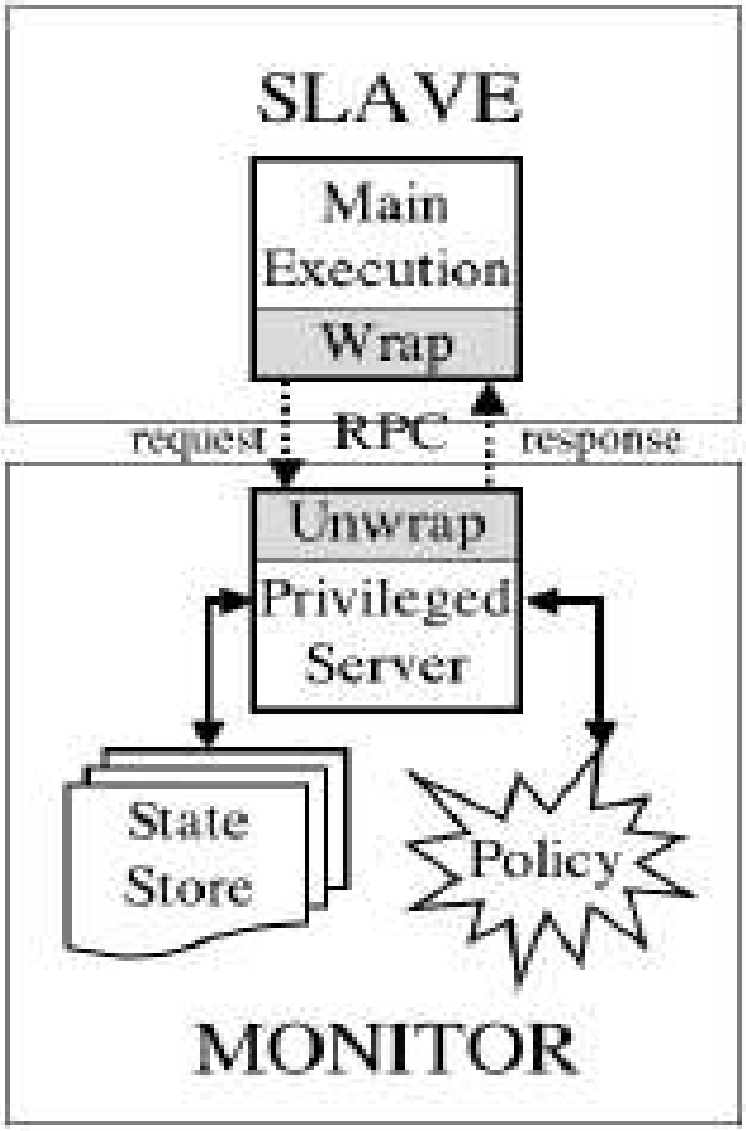
- 「特権属性」をもったデータを slave に直接渡したい場合に、”unpriv”を用いて特権属性を洗い流す
  - 例：秘密鍵と公開鍵のペアを取得する関数が公開鍵のみを slave に返す場合
    - 秘密鍵を priv にすると、秘密鍵→関数→公開鍵と属性が伝搬して公開鍵も priv になってしまう

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

実装の概要

1 Annotation の付加
   - ユーザが任意に加える
2 特権属性の伝搬
3 main 関数の最初に関数 priv_init を呼び出すように書き換える
4 特権属性を持つ関数 f の呼び出しをラッパー関数 privwrap_f に書き換える
5 monitor は slave から関数 f の呼び出しを受けたら関数 privunwrap_f を呼び出すようにする

実装の概念図


SLAVE
Main
Execution
Wrap
request
RPC
response
Unwrap
Privileged
Server
State
Store
Policy
MONITOR

priv_init の役割

- 次のいずれかを行う
  - Monitor を fork して slave の特権レベルを落とす
  - 既に存在している monitor に接続する
- priv_init から返るときにはすべての初期化が終了している
  - 特権属性を持つデータを格納するテーブルの allocate など

privwrap の役割

1. 引数を marshal する
2. 引数と、実行時の特権状態を保持するベクタを monitor に送る
3. Monitor からの反応を待つ
4. 結果を demarshal する
5. slave に適切な結果を返す

privunwrap の役割

1. Monitor に送られてきた引数を demarshal する
2. この関数呼び出しが、操作の列に関するポリシーを満たしているかどうかをチェックする
3. 特権属性を持つデータの実体を、テーブルを引いて取得する
4. 所望の関数を実行する
5. もし返り値が特権属性をもっていたら、その結果をテーブルに格納し、slave に返す値はテーブルへのインデクスとする
6. 返り値を marshal して slave に返す

関数呼び出しのオーバヘッド削減 (1)

- 関数の呼び出しが特権的な呼び出しになるかならないかは、引数や返り値の属性によって変わる
- 特権的な関数呼び出しは monitor との通信オーバヘッドがかかるのでなるべくしない
  - 下の例で、f2 への最初の呼び出しは privwrap_f に変換されるが、次の呼び出しはそのまま

```
int __attribute__((priv)) a;
int  b = 0;
f(a);
f(b);
```

関数呼び出しのオーバヘッド削減 (2)

- 特権的な呼び出しが本当に必要かどうかは実行時でないとわからない場合がある
- 実行時の各引数・返り値の特権情報を保持するベクタを補助的に用いる
  - もし引数・返り値すべてが特権属性を持っていなかったら、privwrap は monitor と通信せず、local に関数を呼び出す
  - ベクタを改ざんされても、関数呼び出しが remote になるか local になるかの違いだけで、危険はない

関数呼び出しのオーバヘッド削減 (3)

- ベクタを利用した例

```
int __attribute__((priv)) a;
int b = 0;
if(some expression) b = a;
b = f(arg1, arg2);
```

→

```
int __attribute__((priv)) a;
int b = 0;
int privvec_f[3] =
  {UNPRIV, UNPRIV, UNPRIV};
if(some expression)
  {privvec_f[0] = PRIV; b = a;}
b = privwrap_f(arg1, arg2, privvec_f);
```

RPC とラッパー関数

- 主要な用途に対する privwrap や privunwrap はあらかじめ用意されている
  - ファイルの open やソケットの作成など
  - 対象のソースコードから自動的に privwrap や privunwrap を作成することはしない
    - ポインタ引数の扱いが難しいので
  - RPC のライブラリを提供しているので、ユーザ独自のラッパーが必要になったとしても簡単に書ける

特権的なデータの保持

- 特権属性を持つデータは monitor の外には漏らさない
  - monitor 内部のハッシュテーブルで管理する
- slave に対しては、データの実体の代わりにテーブルのインデクスを返す
  - インデクスから元のデータを復元するのはまず無理であり、安全

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

実験

- 様々なプログラムに対して本研究で作成したツールを適用した

name	src lines	# user annotations	# calls automatically changed
chfn	745	1	12
chsh	640	1	13
ping	2299	1	31
thttpd	21925	4	13
OpenSSH	98590	2	42
OpenSSL	211675	2	7

OpenSSH

- コネクションを感知したら slave と monitor を fork し、処理を続ける
  - Slave は monitor に認証と秘密鍵の操作を要請する
  - Monitor は slave のユーザ ID とグループ ID を認証されたユーザのそれに変更する
- Annotation は以下の操作に関する 1 ヶ所づつのみ
  - 秘密鍵
  - 認証メカニズム

chfn と chsh

- password ファイルに書き込んだり、ユーザを認証したりする
  - chfn　…ユーザの finger 情報を変更する
  - chsh　…ユーザのログインシェルを変更する
- 両者とも常に上級権限のままで処理を行う
- Annotation はそれぞれ 1 ヶ所のみ
- 自動的に書き換えられたのはそれぞれ 12 、13 ヶ所

thttpd

- Web サーバ
- 80 番ポートにおける bind と accept
- annotation を加えるのに 2 時間かかった
- 4 ヶ所の annotation に対して 13 箇所の関数呼び出しが書き換えられた
  - Socket, bind, fcntl, setsockopt, close, listen, accept, poll

ping

- Raw ソケットを作るために root 権限が必要
- 1 ヶ所の annotation を 90 分かけて加えた
- 31 ヶ所の関数呼び出しがラッパー経由になった
  - Socket, setsockopt, getsockopt, ioctl, sendmsg, recvmsg, poll, getuid, bind
- 各ホストには slave を置き、信頼できるホスト 1 つだけに monitor を置けば、その 1 つのホスト以外からの ping は無視できる
  - ping を無視しなくてもよいファイヤーウォールが作れる

OpenSSL

- たくさんのサービス (slave) が 1 つの認証 (monitor) を利用するような実験をした
  - Monitor は RSA 秘密鍵の操作を行う
- 20 分かけて 2 ヶ所の annotation を加え、7 ヶ所の関数呼び出しが自動的に書き換えられた
- すべての RSA decryption は monitor 内で行われるようになった

実行効率 (1)

- ホスト間での呼び出しのローカルな呼び出しに対するオーバーヘッドは 84%
  - Software-based isolation (Wahbe et al.) という技術を使うと 1/3 まで減らすことができる
- システムコールについて、ローカル呼び出しとラッパー経由呼び出しの平均時間は 19μs vs. 88Ms
  - privman よりは効率がよいらしい

実行効率 (2)

- 特定のアプリケーションについてのテスト
  - thttpd から index.html を 1000 回ダウンロードした… オーバヘッドは 6%
  - localhost に 15 回 ping を打った… 46%
  - OpenSSL にランダムなメッセージを 1000 回 decrypt させた… 15%
- ping と OpenSSL のオーバヘッドは monitor ～ slave 間の通信によるもの
- ping のラッパー関数を ping に最適化させたらオーバヘッドは半分になった

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

自動 vs. 手動

- 自動書き換えだとアプリケーション固有の知識が得られない
  - 最適化に限界がある
- 自動書き換えのほうが圧倒的に簡単

移植性

- 出力もCのソースなので移植性は高い
  - Windowsへの適用も計画中
- ただしラッパーの実装はOSに依存する部分が大きい

様々な問題と解決策 (1)

- setuid/getuid は、想定より低い権限で動く slave の id ではなく monitor の id を操作するようにする
- slave と monitor ではファイルディスクリプタの値が異なっているので、select ではなく poll を使う
  - select では考慮すべきディスクリプタの最大値を指定するが、poll では考慮すべき個々のディスクリプタを指定する
- slave が新しいプロセスを fork したときは、その新プロセスに対応する monitor も 1 つ fork する

様々な問題と解決策 (2)

- 配列の各要素それぞれが特権属性を持っているかどうかを判定するのは難しい
  - 属性を持っているなら、テーブルから実体を引いてこなければならない
  - ファイルディスクリプタの実体は 100 未満で、テーブルへのインデクスは 100 以上である性質を利用して解決した
- ポインタの生存期間がよくわからないので、monitor 内のデータをいつ free したらいいのかわからない

- Privilege separation とは
- 本研究のアプローチ
- 実装の詳細
- 実験・評価
- 議論・その他
- まとめ

まとめ

- 自動で privilege separation を行うしくみ
  - Monitor が特権的なリソースへのアクセスを仲立ちする
  - 特権的な命令に由来するデータも守る
  - オーバヘッドを減らすためにダイナミックな情報を用いる
  - 手動で加える annotation は非常に少なくて済む
  - 実際のオーバヘッドは許容範囲