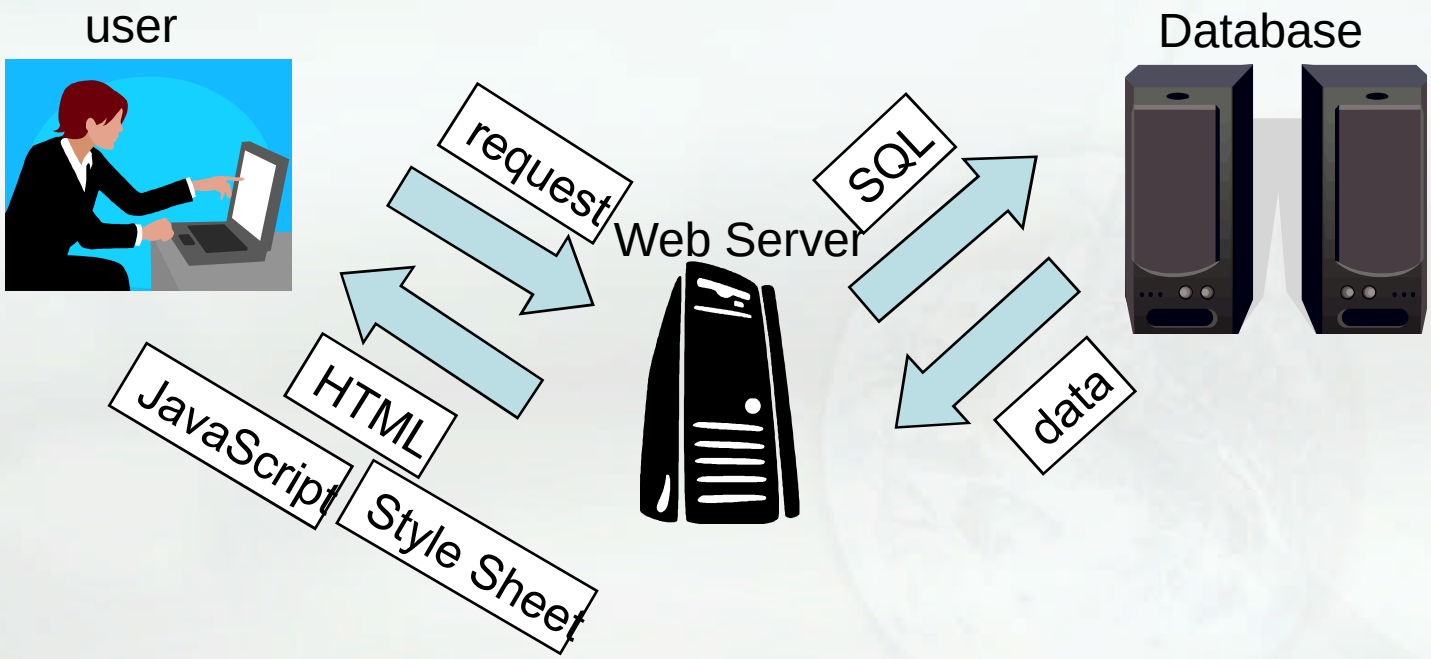
インジェクション系脆弱性を持つコードの記述が不可能なフレームワーク
Framework for making it impossible to write codes vulnerable for injection attacks

SCIS 2008
2008/01/22 渡邉　悠

Agenda

- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

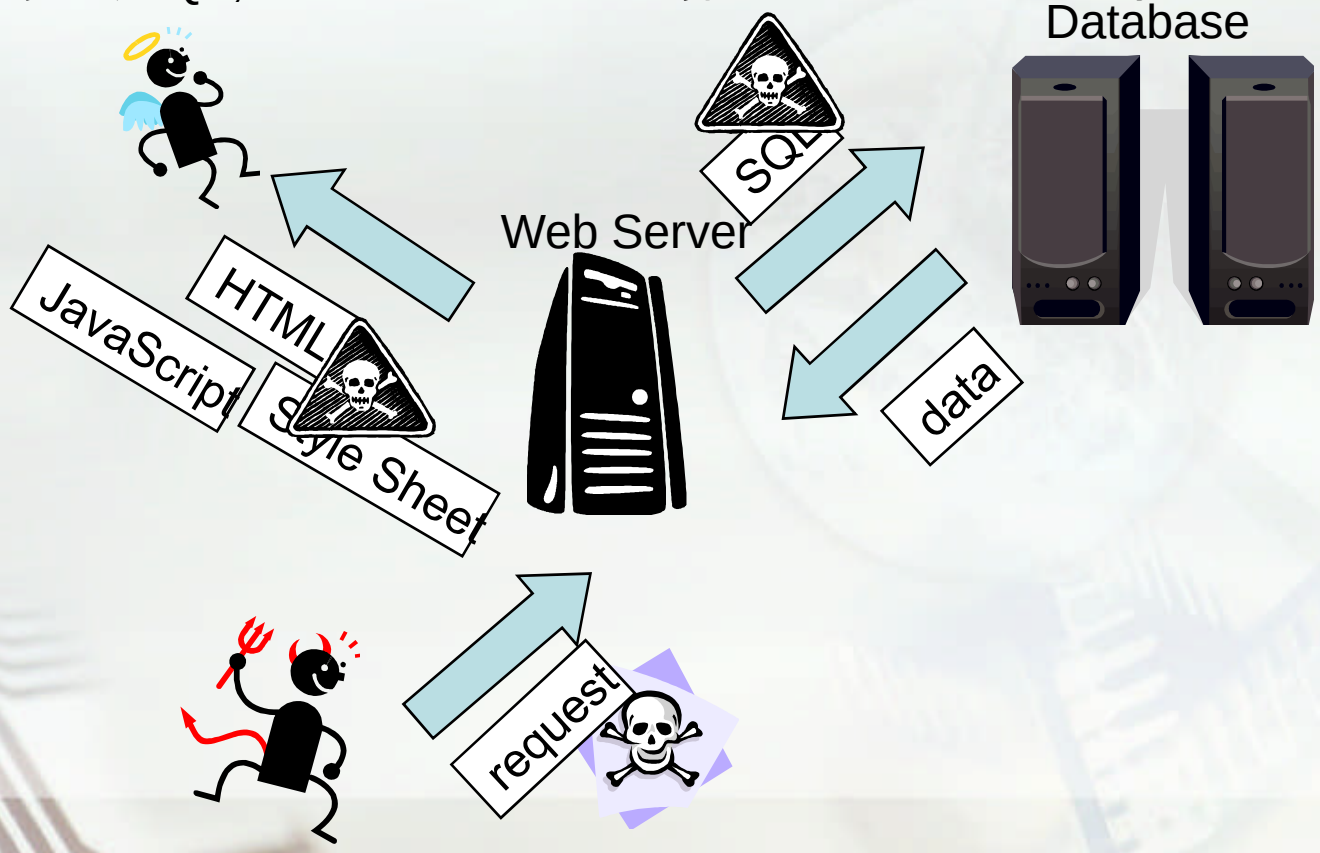
Webアプリケーションの構成



- ユーザが入力した値からプログラムが作成される
  - クライアントへレスポンス
    - HTML・JavaScript・Style Sheet
  - データベースへの問い合わせ
    - SQL

インジェクション攻撃

- プログラム作成の処理が不適切な場合、悪意のあるクライアントによってSQLやHTML が改竄される。
- 多くの場合、SQL, HTMLが文字列の加算で作成されることが原因



インジェクション攻撃

- 各種のインジェクション攻撃
  - SQL : SQL Injection[7]
  - HTML・JavaScript : Cross-site scripting
  - Shell : OS command Injection
  - XML, XQuery, etc…
  - ある言語を動的に生成するという処理は、SQL, HTMLに代わる技術が現れてもなくならない
- インジェクションの問題
  - 悪意のあるユーザがプログラムを改竄する
    - 情報の改竄
    - 情報漏洩
  - ユーザの入力が正常に処理されない
    - 正規ユーザの入力が正しく処理されず、システムが正常に機能しない
      - Ex. シングルクオートやバックスラッシュが含まれる入力が全て拒否される

Agenda

- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

攻撃を検出する技術

- 攻撃自体を検出する
  - SNORT[8]などのIDSによる侵入検出
  - 主にシグネチャベース
- 攻撃による改竄を検出する技術
  - 正常状態からの「ズレ」を検出
  - 主に学習に基づく手法（正常状態を学習） [4, 9, 11]

脆弱性を検出する技術

- ホワイトボックス的な手法
  - 脆弱性を招きやすいプログラムのパターンを検知し、間接的に脆弱性を検出
    - 動的な解析による手法：Perl, Rubyの汚染検出モード[3]
    - 静的な解析による手法:ソースコードの解析
- ブラックボックス的な手法
  - 実際にシステムに対して攻撃を行い脆弱性を検査する手法

安全なプログラムを構成する技術

- エスケープ処理
  - 問題を引き起こす危険な文字を正しく処理する
  - Ex.「¥ → ¥¥」, 「“ → ¥”」
- 記述補助ツール
  - Object-Relationalマッピング
  - Object-XMLマッピング
  - ASP.NETを利用したHTMLの作成

既存技術の問題点

- 攻撃の検出技術
  - 攻撃の検出漏れ
  - 誤検知:システムの正常な実行を妨害
  - 対症療法的な手法であり、ユーザの入力を正しく処理できていないことによるバグには無力。根本的な解決にはならない。
- 脆弱性の検出
  - 脆弱性の検出漏れ
- 安全なプログラムを構成する技術
  - 安全性の確保がプログラマ任せになる

Agenda

- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

Motivation

- 以下のような新しいフレームワークの設計
  - ユーザの入力を正しく処理できていないことによる脆弱性だけでなく、バグに対しても有効であり
  - 本質的に不確実性を含む検出技術に頼ることなく
  - システムの安全性・正しさをプログラマ任せにすることなく確保できるフレームワーク
- 方向性
  - 自由に記述したプログラムに対して検証を行うのは困難（e.g. モデル検証）[5]
  - プログラマに対して安全なツールの利用を強制することで、システムの安全性・正しさを確保する

各手法の考え方の比較

- 攻撃の検出・脆弱性の検出技術
  - プログラマに自由に記述させたプログラムに何かを施すことで、安全性を上げる手法
- 安全な開発ツール
  - プログラムを正しく、安全に記述するための技術
  - 安全性、正しさの確保はプログラマ任せ
- 提案手法
  - 安全な開発手法をプログラマが利用することを強制することで、アーキテクチャのレベルで正しさ・安全性を確保する技術

Agenda

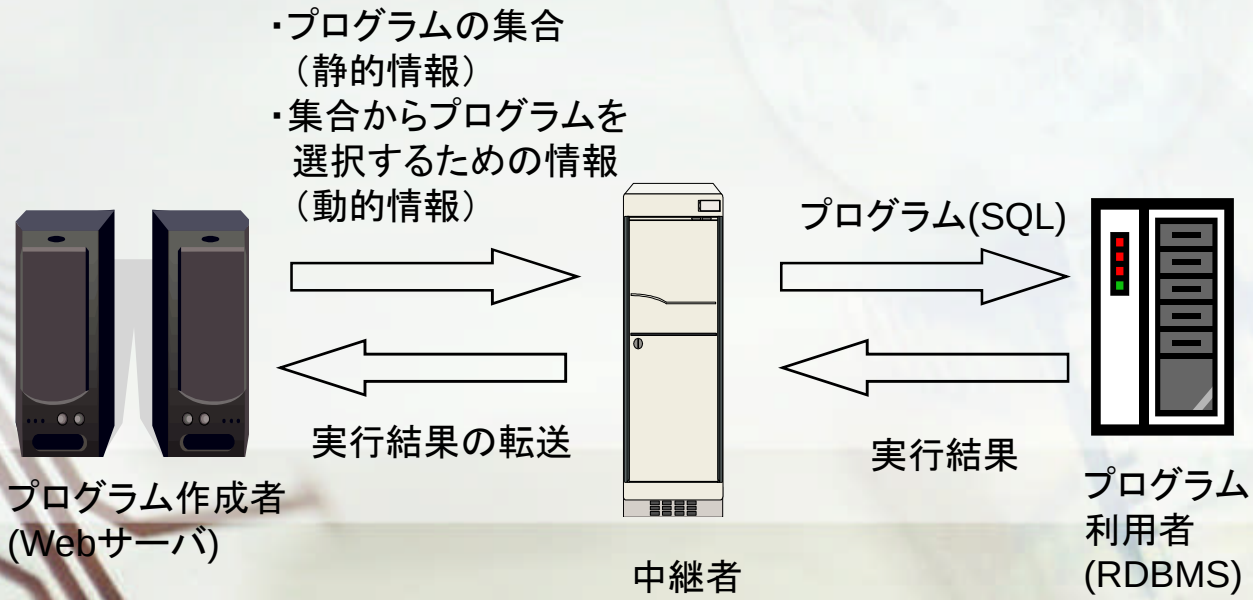
- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

プログラム生成に対する考察

- プログラム（SQL, HTML, etc…）は静的な情報と動的な情報から構築される
  - 静的な情報
    - SQL, HTMLを構築するプログラム（C#, Java）に埋め込まれた情報
    - 生成されるプログラムの集合を定義
  - 動的な情報
    - ユーザが入力したパラメータ
    - プログラムの集合からプログラムを選択
  - インジェクションの原因
    - 生成されるプログラムが、静的情報によって適切に制限されていない。動的情報によって多くのことが可能である
      - プログラム作成のための適切なツールを設計し、その利用を強制することでシステムの安全性を確保する

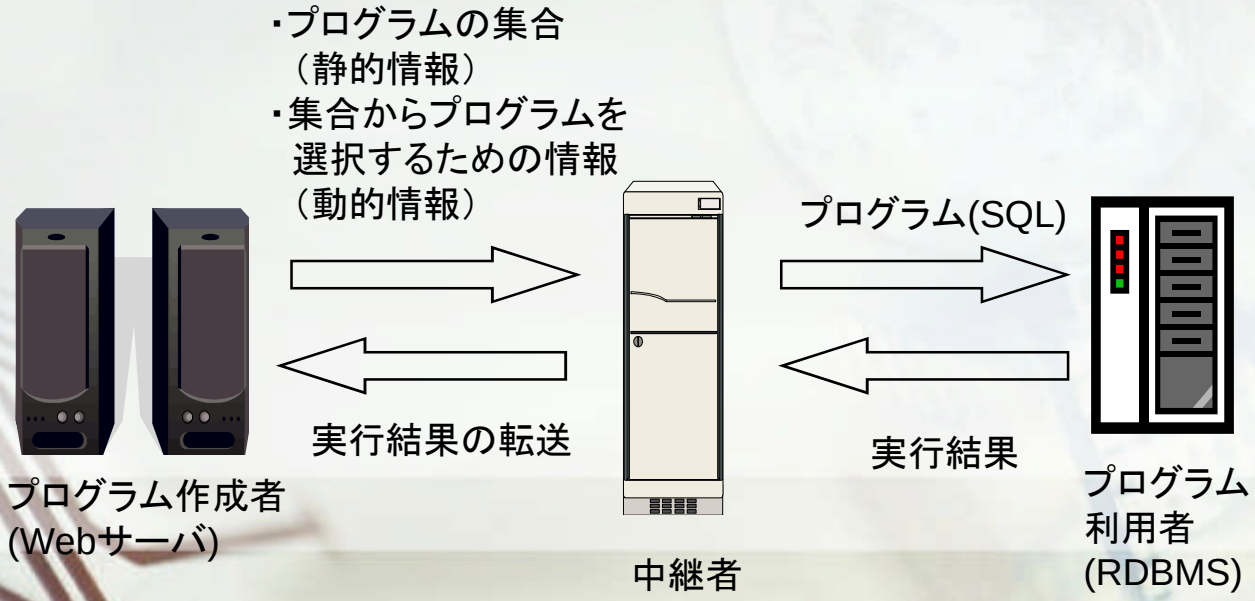
アーキテクチャ

- 「プログラム作成者」の実装には立ち入らずに、システムの安全性を確保する
- 中継者をシステムに導入
  - プログラム作成者とプログラム利用者の間にコンポーネントを挟み、特定のツールを利用しなければプログラムが生成できないようにする。
  - 中継者がプログラム作成者から、プログラムの集合（静的情報）とその集合からプログラムを選択するための情報を受け取りプログラムを作成、プログラム利用者に受け渡す
  - プログラム作成者がプログラム利用者に直接アクセスできないように、システムのアーキテクチャを設定する



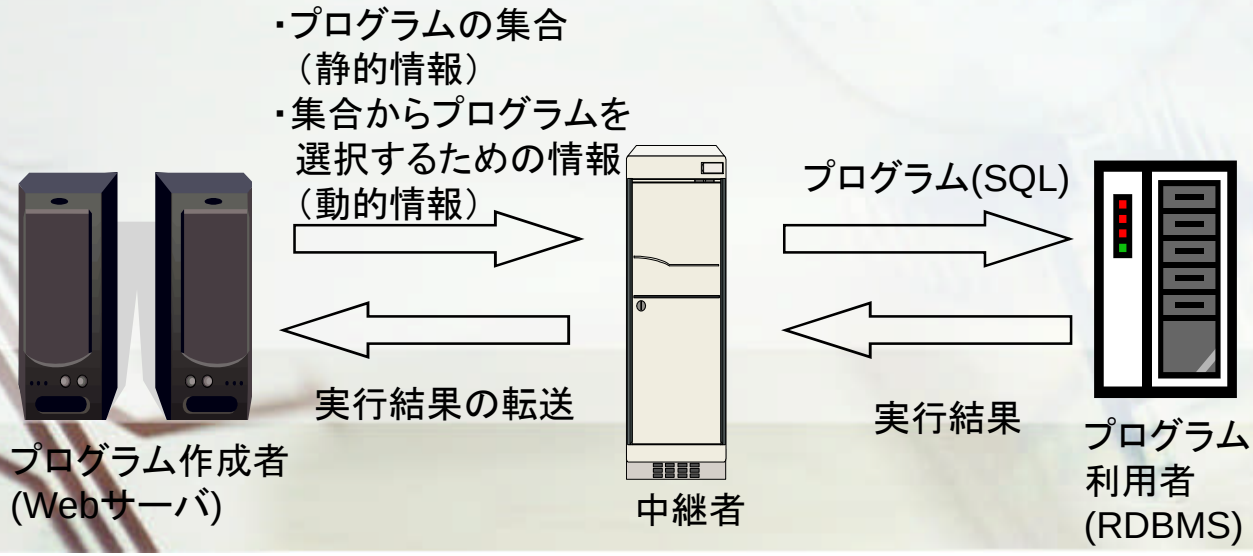
中継者に求められる性質

- システムが利用する「プログラムの集合」の定義が意図通りに行えること
  - もし開発者が意図していないプログラムが集合に含まれていると、開発者が意図していないプログラムがユーザの入力によって作成されてしまう
- 開発効率を低下させないこと
  - システム中で利用するプログラムの集合が効率的に定義できなくてはならない



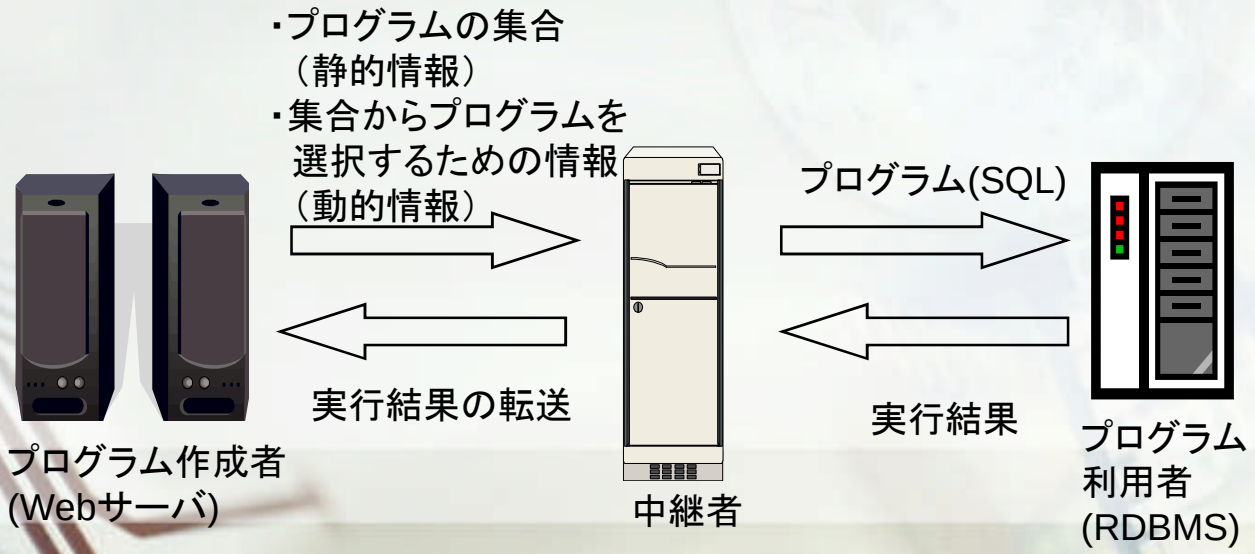
中継者に求められる性質

- プログラムの集合の定義が静的であること
  - 本手法の基本アイディアはプログラムの集合の定義（静的な情報）を利用して、生成されるプログラムを適切に限定することでインジェクションを防ぐというものである
  - そのためプログラムの集合の定義がユーザの入力から動的に作られていてはならない。
- インジェクション系の問題が入り込む余地がないこと
  - ユーザの入力（動的な情報）によって行える処理を最小限にする
  - 集合からプログラムを選択するためのインターフェイスをできる限り単純化する
  - 非常に大きな「プログラムの集合」を定義できてはならない

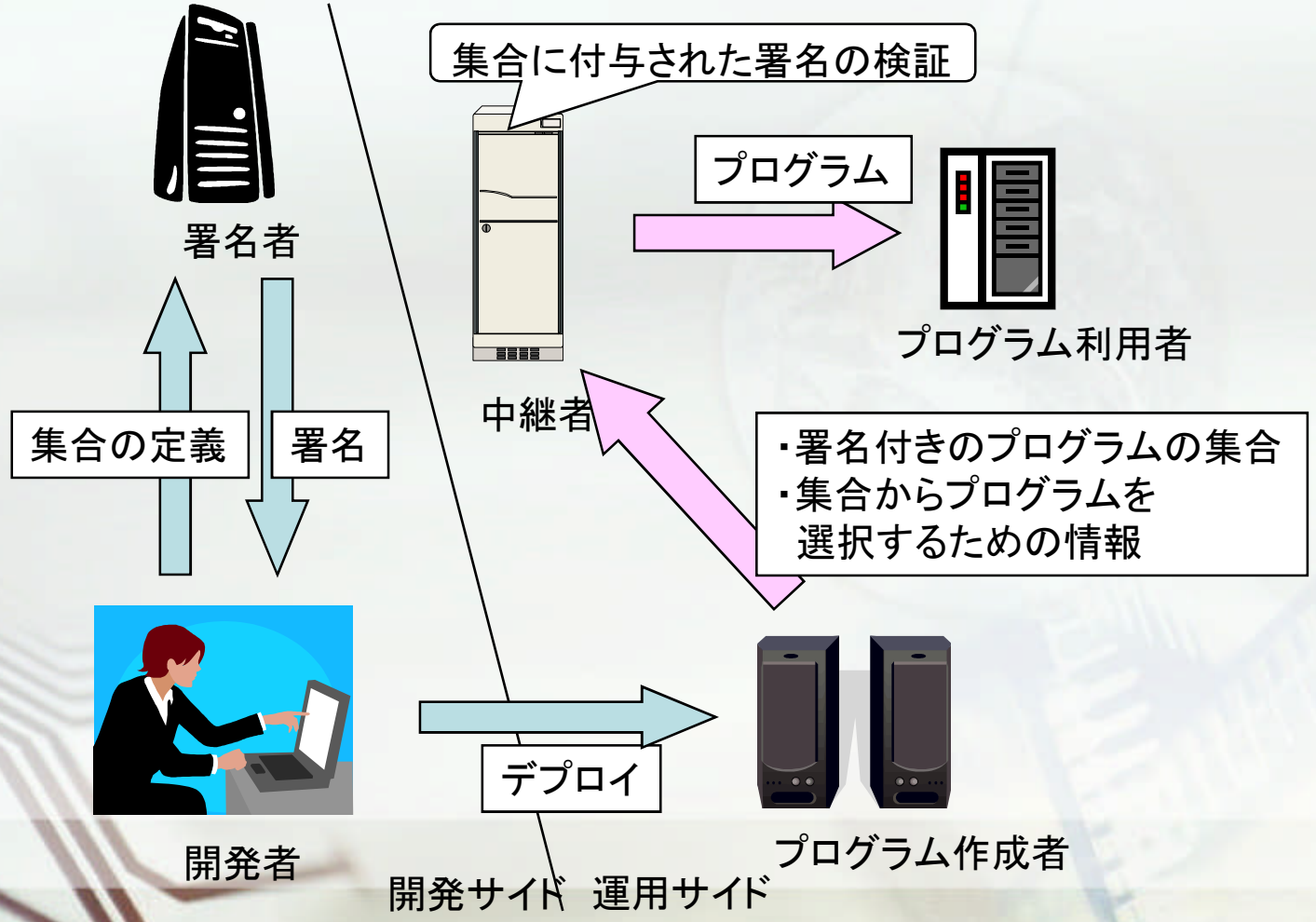


中継者に求められる性質(4/4)

- 対象とする言語(SQL, HTML)に依存する要求
  - システムが利用する「プログラムの集合」の定義が意図通りに行えること
  - 開発効率を低下させないこと
  - 動的な情報によるプログラムの選択部分がシンプルであること
- 依存しない要求
  - プログラムの集合の定義が静的であること



プログラムの集合が静的であることを保証するアーキテクチャ



Agenda

- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

特定の言語への適用

- 以下の要求を満たすように以下に「プログラムの集合」の定義とプログラムの選択のためのインターフェイスを設計するかが鍵となる
  - システムが利用する「プログラムの集合」の定義が意図通りに行えること
  - 開発効率を低下させないこと
  - 動的な情報によるプログラムの選択部分がシンプルであること
- 特に2番目と3番目の要求が競合
  - 動的に行えることを制限し、不必要なことはできなくするという要求
  - 開発者が行いたいと思っている処理は制限せず、効率よく行えるようにする
- 落とし所
  - 動的に決定できる必要のある要素を洗い出し、そうした部分のみについて、動的に決定することを許可する

SQLへの適用

- Webアプリケーションで利用されるSQLの洗い出し
- CRUD処理
  - Create ：INSERT文
    - 動的に決まる要素：挿入する値（パラメータ）
  - Read　：SELECT文
    - 動的に決まる要素：検索条件(WHERE), データの表示順(ORDER BY)
  - Update : UPDATE文
    - 動的に決まる要素：検索条件, パラメータ
  - Delete ：DELETE文
    - 動的に決まる要素：検索条件
- その他
  - ストアードプロシージャ：パラメータ

パラメータの動的な決定と
検索条件の動的な決定が必要

パラメータの動的な決定

- バインディングメカニズム
  - sql = “select * from users where id = @id”
  - sqlCmd = new SqlCommand(sql);
  - sqlCmd.Parameters.Add(“@id”, id);
  - sqlCmd.Execute();
- 解説
  - 値を代入する部分をとりあえず、@idなどとしてクエリを記述しておく
  - 実行時に「@id」に「id」の中の値を代入する。（バインディング）
    - @idがidの文字列で置換されるのではない
    - @idがidの文字列をSQL上であらわす文字列リテラルに置換される

検索条件の動的な決定

- あらゆる検索条件が動的に構築できるのは望ましくない
- 検索条件を構築する必要のある場合
  - 条件の有無が実行時に決定される場合
    - 顧客データの中から、名前・性別・生年月日などで顧客を検索。ただし、名前のみ性別のみなど、一部の条件だけで検索することも可能。
  - 条件が繰り返される場合
    - IDがfooもしくはbarもしくはhogeのものを検索
- バインディングメカニズムを拡張し、条件の有無と条件の繰り返しのみを動的に扱える手法を設計・実装[10]
  - 条件の有無の動的な決定
    - 利用しない検索条件のパラメータに、その条件を利用しないことを表すあたいをバイディングすることで条件を削除する
  - 条件の繰り返し
    - 条件の繰り返しを表す部分に、配列をバイディングすることで、条件の繰り返しをサポート

実装

- .NET FrameworkのADO.NET[1]のライブラリとして実装
  - ADO.NET : データベース接続ライブラリ。JavaのJDBC[6]のようなもの

Agenda

- インジェクション攻撃
- 既存の対策技術
- 動機 と本研究の貢献
- 研究内容
  - 脆弱なコードの記述を防ぐアーキテクチャ
  - 提案アーキテクチャの、SQLを利用するシステムへの適用
    - 設計と実装
- まとめ

まとめ

- インジェクション系の問題を防ぐための新しいアーキテクチャの提案
  - 機械学習などの技術に頼らないアーキテクチャ
  - システムの安全性・正しさをプログラマ任せにせずに保つためのアーキテクチャ
  - 基本アイディア
    - プログラム（SQL）作成者（Webサーバ）と利用者（データベース）の間に、プログラムを正しく・安全に生成するためのコンポーネントを挟むことで、安全なシステムの作成を強制する
  - 間にはさむコンポーネントが満たすべき性質の整理
- SQLを対象とした、実際のシステムの設計・実装
- 本手法の短所
  - 既存のシステムに対して導入するには、プログラムの書き換えが不可欠となる。
  - これは、特にシステムに問題がないもない場合には無駄なコストとなる

References (1/2)

- [1] ADO.NET. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/adonetanchor.asp.
- [2] Mike Andrews. The state of web security. *IEEE Security and Privacy, Vol. 4, No. 4, pp. 14–15,* JULY/AUGUST 2006.
- [3] Gunther Birznieks. Cgi/perl taint mode faq. http://gunther.web66.com/FAQS/taintmode.html.
- [4] Gregory Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware, pp. 106–113, New York, NY, USA, 2005.* ACM Press.
- [5] Gerard J. Holzmann. The model checker SPIN. *Software Engineering, Vol. 23, No. 5, pp. 279–295, 1997.*

References (2/2)

- [6] JDBC. http://java.sun.com/javase/technologies/database/.
- [7] SPI Labs. SQL injection are your web applications vulnerable? In *SPI Dynamics Whitepaper, 2006.*
- [8] SNORT. Snort.org. http://www.snort.org/.
- [9] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Network and Distributed System Security Symposium (NDSS), 2007.*
- [10] 渡邉悠松浦幹太. SQL の条件説が動的に構成されることを考慮したデータベース接続API の設計. In *Computer Security Symposium, pp. 571–576, 奈良, 2007.*
- [11] 川中真耶. 異常木検知によるjavascript injection 検知. In *Computer Security Symposium, pp. 417–422, 奈良,* 2007.