A Classification of SQL Injection Attack Techniques and Countermeasures

William G.J. Halfond, Jeremy Viegas & Alessandro Orso

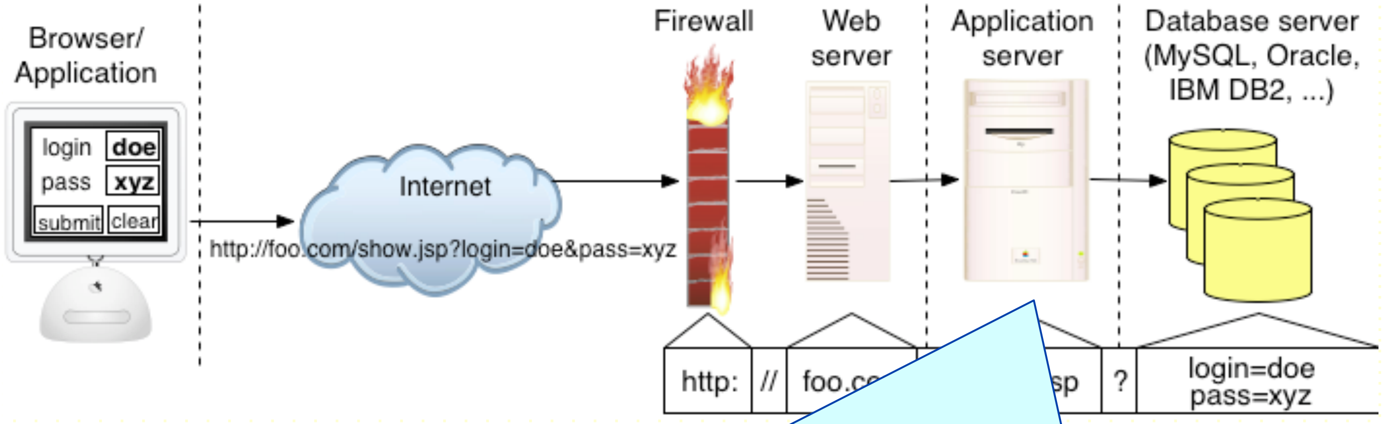
Georgia Institute of Technology

This work was partially supported by DHS contract FA8750-05-2-0214 and NSF award CCR-0209322 to Georgia Tech.

SPARC

GEORGIA TECH INFORMATION SECURITY CENTER

Vulnerable Application



```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals(""))) {
  queryString += "login='" + login + "' AND pin=" + pin ;
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);
```

Attack Scenario

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals(""))) {
  queryString += "login='" + login + "' AND pin=" + pin ;
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);
```

Normal Usage

- User submits login “**doe**” and pin “**123**”
  - *SELECT info FROM users WHERE login= `**doe**’ AND pin= **123***

Attack Scenario

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals("")))  {
  queryString += "login='" + login + "' AND pin=" + pin ;
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);
```

Malicious Usage

- Attacker submits “**admin’ --** ” and pin of “0”
  - *SELECT info FROM users WHERE login=‘**admin’ --** ’ AND pin=0*

Presentation Outline

- SQL Injection Attacks
  - Intent
  - Input Source
  - Type
- Countermeasures
- Evaluation of countermeasures
- Lessons learned

Intent

- Extracting data
- Adding or modifying data
- Performing denial of service
- Bypassing authentication
- Executing remote commands

Sources of SQL Injection

Injection through user input

- Malicious strings in web forms.

Injection through cookies

- Modified cookie fields contain attack strings.

Injection through server variables

- Headers are manipulated to contain attack strings.

Second-order injection

- Trojan horse input seems fine until used in a certain situation.

Second-Order Injection

Attack does not occur when it first reaches the database, but when used later on.

Input: admin’-- ===> admin\’--

```
queryString =
 "UPDATE users SET pin=" + newPin +
 " WHERE userName=’" + userName + "’ AND pin=" + oldPin;
```

```
queryString =
 “UPDATE users SET pin=’0’
  WHERE userName= ’admin’--’ AND pin=1”;
```

Types of SQL Injection

- Piggy-backed Queries
- Tautologies
- Alternate Encodings
- Inference
- Illegal/Logically Incorrect Queries
- Union Query
- Stored Procedures

Type: Piggy-backed Queries

Insert additional queries to be executed by the database.

```
queryString = “SELECT info FROM userTable WHERE” +
              “login=‘” + login + “' AND pin=” + pin;
```

Input **`pin`** as “0; DROP database webApp”

```
queryString = “SELECT info FROM userTable WHERE
               login=‘name' AND pin=0; DROP database webApp”
```

Type: Tautologies

Create a query that always evaluates to true for entries in the database.

```
queryString = “SELECT info FROM userTable WHERE” +
              “login=‘” + login + “' AND pin=” + pin;
```

Input **`login`** as “user’ or 1=1 --”

↓

```
queryString = “SELECT info FROM userTable WHERE
               login=‘user‘ or 1=1 --' AND pin=“
```

Type: Alternate Encodings

Encode attacks in such a way as to avoid naïve input filtering.

```
queryString = “SELECT info FROM userTable WHERE” +
              “login=‘” + login + “' AND pin=” + pin;
```

Input `pin` as “0; declare @a char(20) select @a=0x73687574646f776e exec(@a)“

```
“SELECT info FROM userTable WHERE
 login=‘user' AND pin= 0;
 declare @a char(20) select @a=0x73687574646f776e exec(@a)”
```

Type: Alternate Encodings

SHUTDOWN

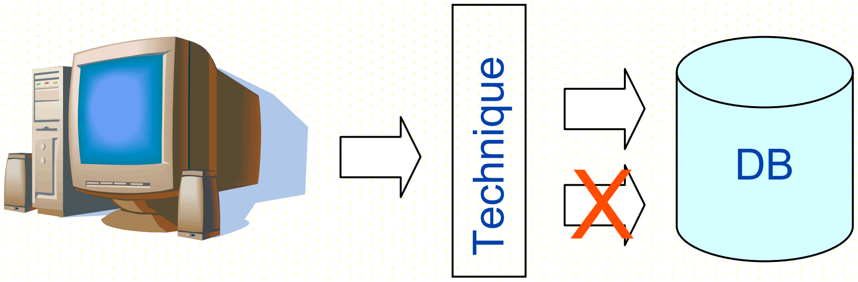
Countermeasures

Prevention

- Augment Code
- Detect vulnerabilities in code
- Safe libraries

Detection

- Detect attacks at runtime

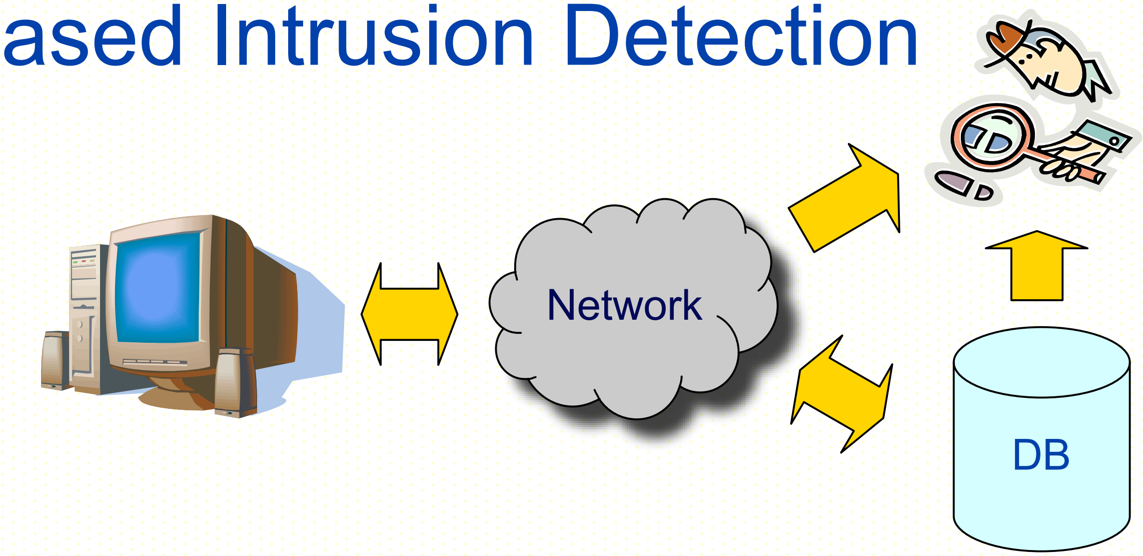


Prevention Techniques

- Defensive Coding Best Practices
- Penetration Testing
- Static Analysis of Code
- Safe Development Libraries
- Proxy Filters

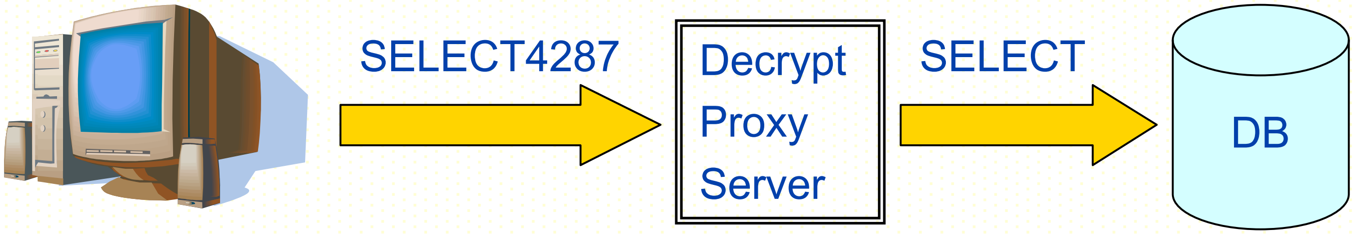
Detection Techniques

- Anomaly Based Intrusion Detection



Detection Techniques

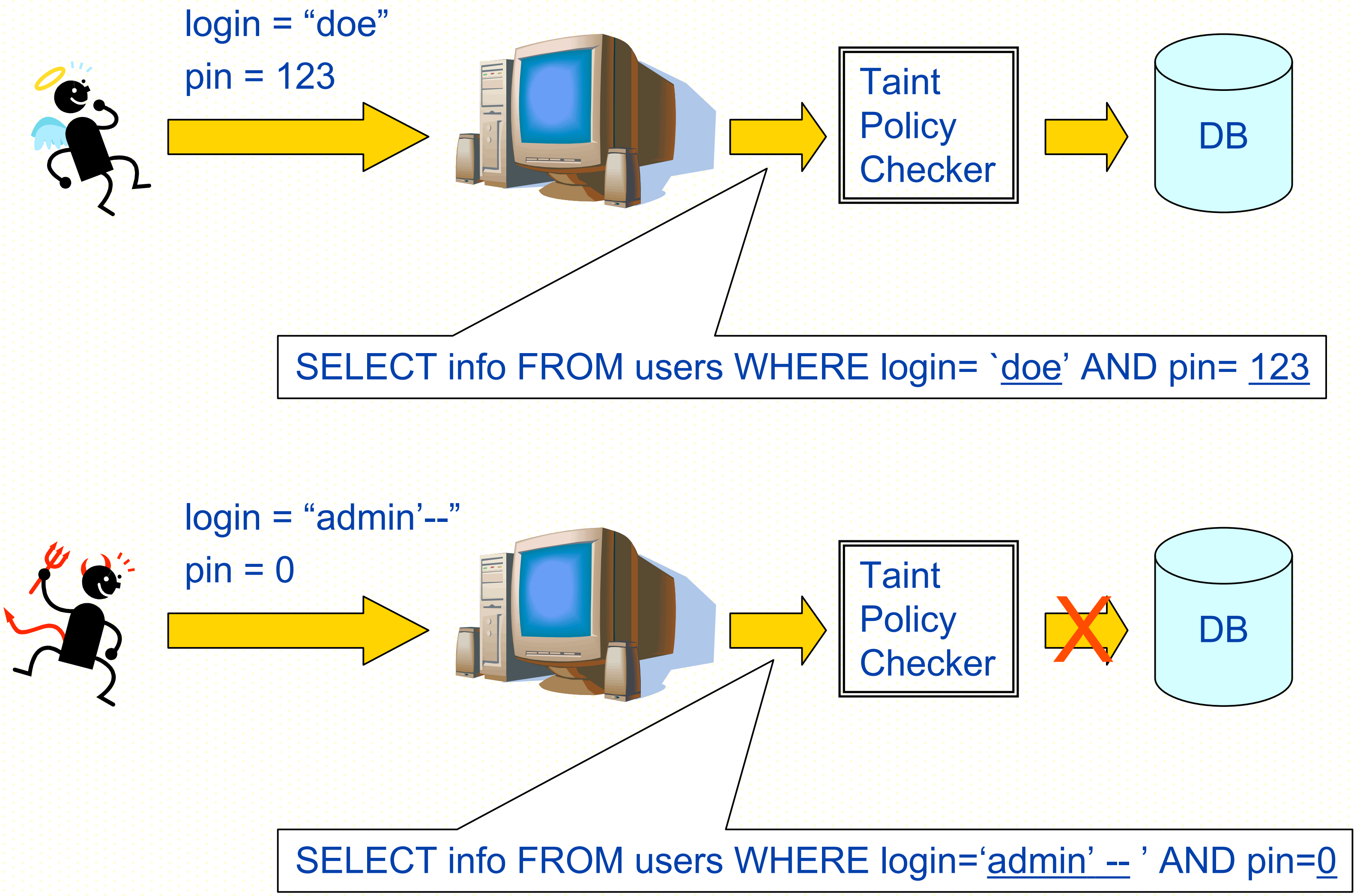
- Anomaly Based Intrusion Detection
- Instruction Set Randomization



Detection Techniques

- Anomaly Based Intrusion Detection
- Instruction Set Randomization
- Dynamic Tainting
- Model-based Checkers

Dynamic Tainting



SELECT info FROM users WHERE login= `doe’ AND pin= 123

SELECT info FROM users WHERE login=‘admin’ -- ’ AND pin=0

Model-based Checkers: AMNESIA

Basic Insights

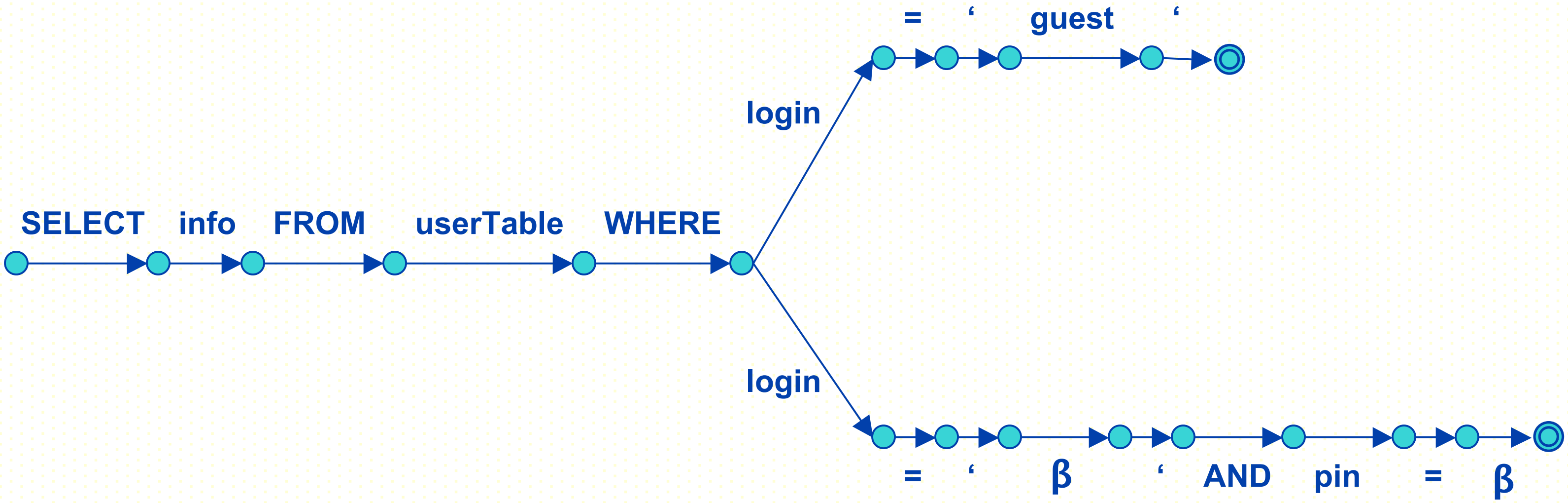
1. Code contains enough information to accurately model all legitimate queries.
2. A SQL Injection Attack will violate the predicted model.

Solution:

Static analysis => build query models

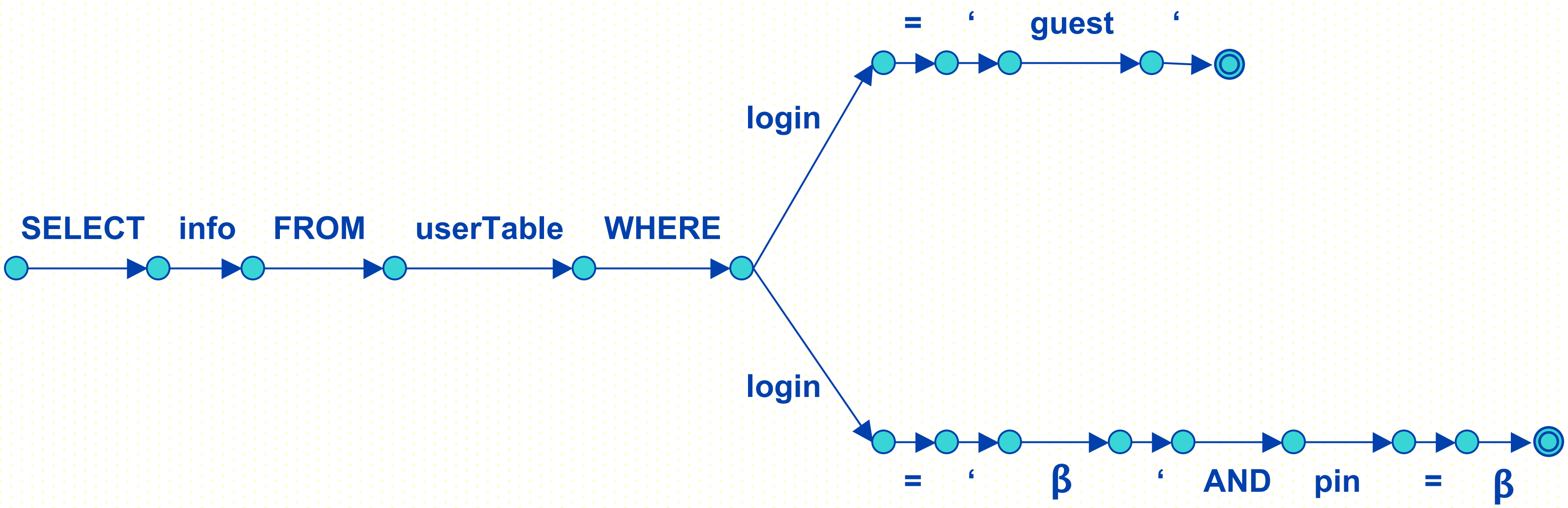
Runtime analysis => enforce models

Model-based Checkers: AMNESIA

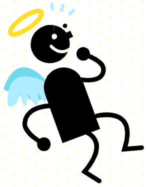


```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals("")))  {
  queryString += "login='" + login + "' AND pin=" + pin ;
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);
```

Model-based Checkers: AMNESIA

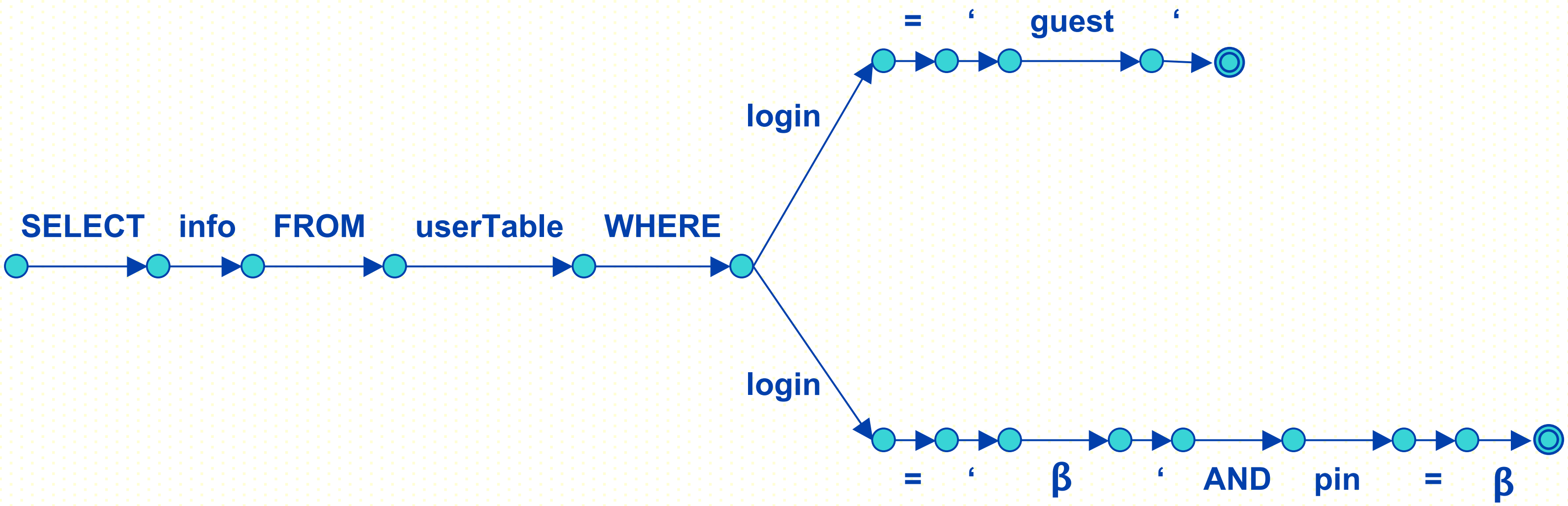


Normal Usage:



SELECT info FROM userTable WHERE login = ‘ doe ‘ AND pin = 123

Model-based Checkers: AMNESIA



Malicious Usage:

SELECT info FROM userTable WHERE login = ‘ admin ‘ -- ‘ AND pin = 0

Evaluation

- Qualitative vs. Quantitative
- Evaluate technique with respect to
  1. Injection Sources
  2. SQLIA Types
  3. Deployment Requirements
  4. Degree of automation

Summary of Results

Prevention Techniques

- Most effective: Java Static Tainting [livshits05] and WebSSARI [Huang04]
- Not completely automated
- Runner-ups: Safe Query Objects [cook05], SQL DOM [mcclure05] (Safe development libraries)
  - Require developers to learn and use new APIs
- Effective techniques automated enforcement of *Best Practices*

Summary of Results

Detection Techniques

- Problems caused by Stored Procedures, Alternate Encodings
- Most accurate: AMNESIA [halfond05], SQLCheck [su06], SQLGuard [buehrer05] (Model-based checkers)
- Of those, only AMNESIA is fully automated
- Runner-ups: CSSE [pietraszek05], Web App. Hardening [nguyen-tuong05] (Dynamic tainting)
  - Fully automated
  - Require custom PHP runtime interpreter

Conclusions and Lessons Learned

1. SQLIAs have:
   a) Many sources
   b) Many goals
   c) Many types
2. Detection techniques can be effective, but limited by lack of automation.
3. Prevention techniques can be very effective, but should move away from developer dependence.

Questions

Thank you.

SPARC

GEORGIA TECH INFORMATION SECURITY CENTER

References

V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In *Proceedings of the 14th Usenix Security Symposium*, pages 271–286, Aug. 2005.

Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In *Proceedings of the 12th International World Wide Web Conference (WWW 04)*, May 2004.

W. R. Cook and S. Rai. Safe Query Objects: Statically Typed Objects as Remotely Executable Queries. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, 2005.

R. McClure and I. Kr¨uger. SQL DOM: Compile Time Checking of Dynamic SQL Statements. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 05)*, pages 88–96, 2005.

W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In *Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, USA, Nov 2005.

Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In *The 33rd Annual Symposium on Principles of Programming Languages (POPL 2006)*, Jan. 2006.

G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In *International Workshop on Software Engineering and Middleware (SEM)*, 2005.

A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans. Automatically Hardening Web Applications Using Precise Tainting Information. In *Twentieth IFIP International Information Security Conference (SEC 2005)*, May 2005.

T. Pietraszek and C. V. Berghe. Defending Against Injection Attacks through Context-Sensitive String Evaluation. In *Proceedings of Recent Advances in Intrusion Detection (RAID2005)*, 2005.