# AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks

William Halfond
Alessandro Orso
Georgia Institute of Technology

This work was supported in part by NSF awards CCR-0306372, CCR-0205422, and CCR-0209322 to Georgia Tech, and by the DHS

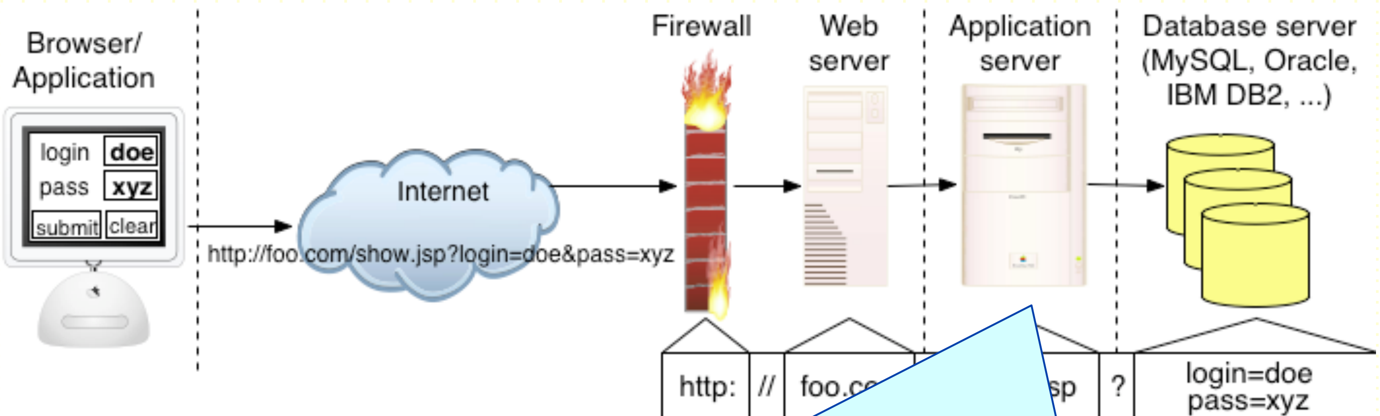
SPARC

# AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks

- David Aucsmith (CTO of Security and Business Unit, Microsoft) defined SQLIA as one of the most serious threats to web apps
- Open Web Application Security Project (OWASP) lists SQLIA in its top ten most critical web application security vulnerabilities
- Successful attacks on Guess Inc., Travelocity, FTD.com, Tower Records, RIAA, …

SPARC

# Vulnerable Application




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

# Attack Scenario

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

**Normal Usage**

¬User submits login "**doe**" and password "**xyz**"

¬*SELECT info FROM users WHERE login='**doe**' AND pass='**xyz**'*

# Attack Scenario

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

**Malicious Usage**

- Attacker submits "**admin' or 1=1 --** " and password of ""
  - *SELECT info FROM users WHERE login='**admin' or 1=1 --** ' AND pass=''*

# Background Information

"Why the obvious solutions don't work."

- Input filtering
- Stored procedures
- Defensive coding

# Presentation Outline

- Background Information
- The AMNESIA Technique
- Empirical Evaluation
- Related Work
- Conclusion

# Our Solution: 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

# Overview of the Technique

1. Identify all hotspots.
2. Build SQL query models for each hotspot.
3. Instrument hotspots.
4. Monitor application at runtime.

# 1 – Identify Hotspots

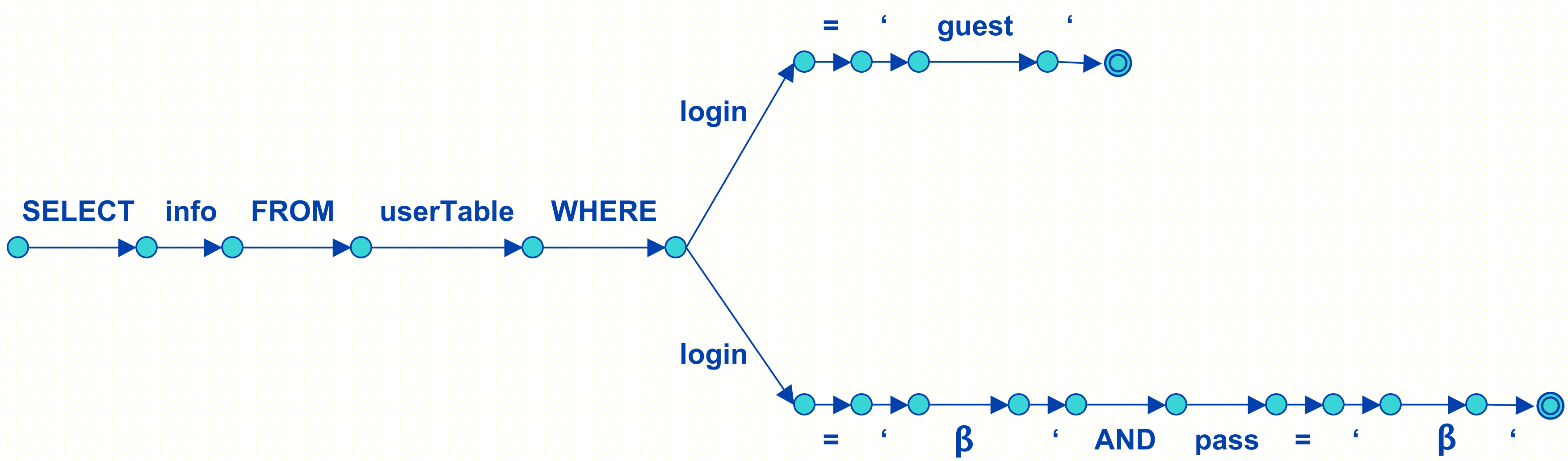
Scan application code to identify hotspots.

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

Hotspot

# 2 – Build SQL Query Model

1. Use Java String Analysis[1] to construct character-level automata
2. Parse automata to group characters into SQL tokens



# 3 – Instrument Application

## Wrap each hotspot with call to monitor.

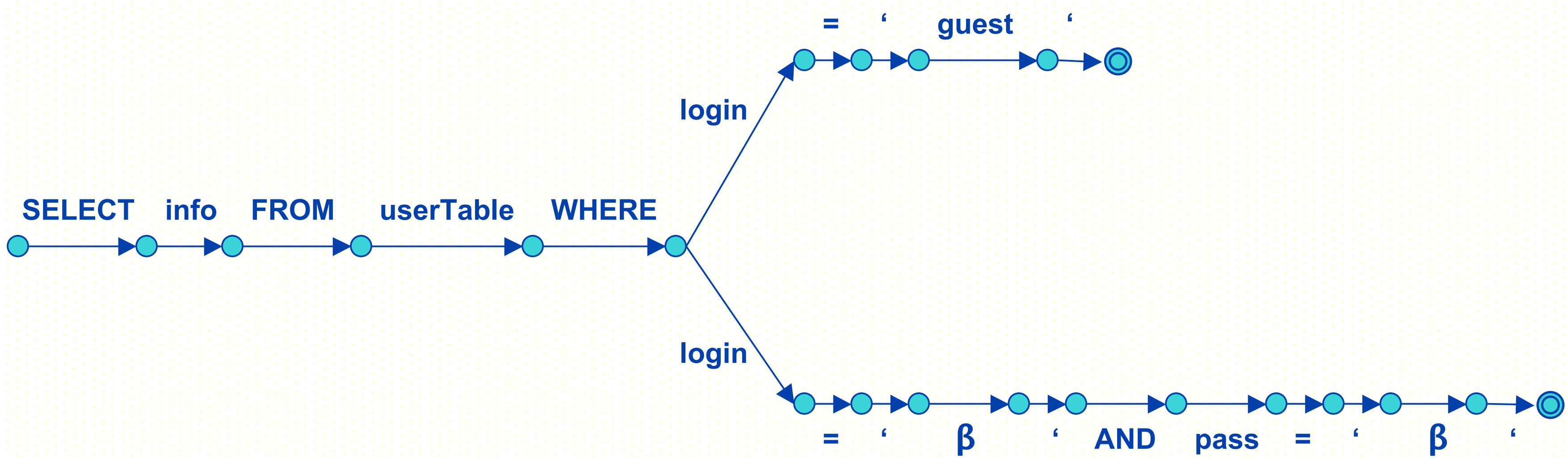
```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals(""))) {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
if (monitor.accepts (hotspotID, queryString) {
  ResultSet tempSet = stmt.execute(queryString);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

Call to Monitor

Hotspot

# 4 – Runtime Monitoring

Check queries against SQL query model.

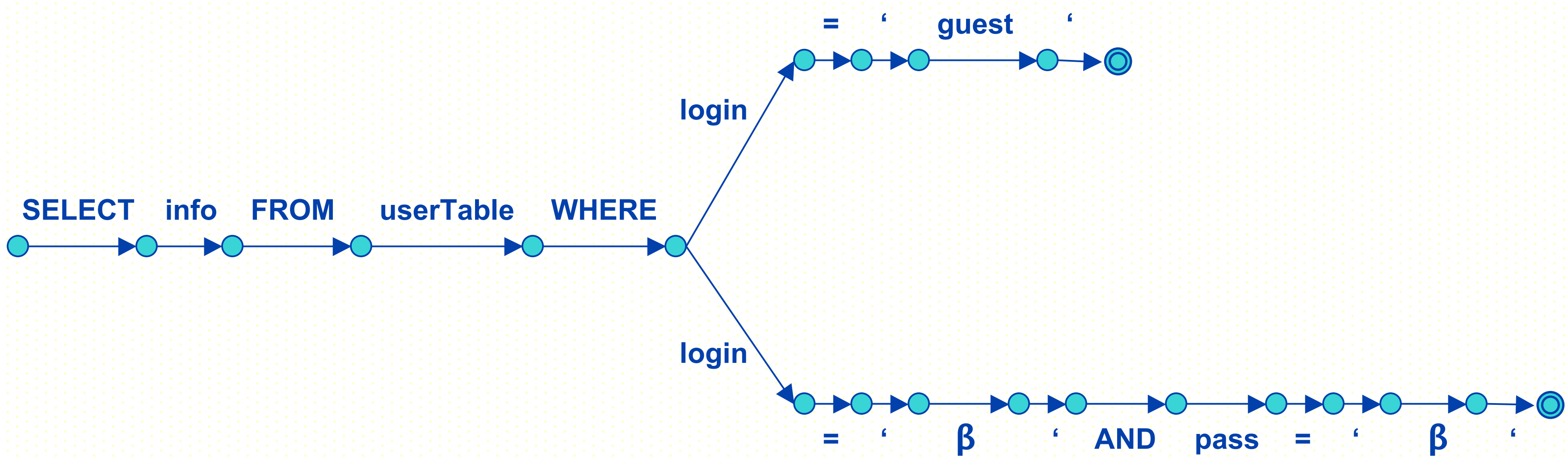



Normal Usage:

SELECT | info | FROM | userTable | WHERE | login | = | ‘ | doe | ‘ | AND | pass | = | ‘ | xyz | ‘

# 4 – Runtime Monitoring

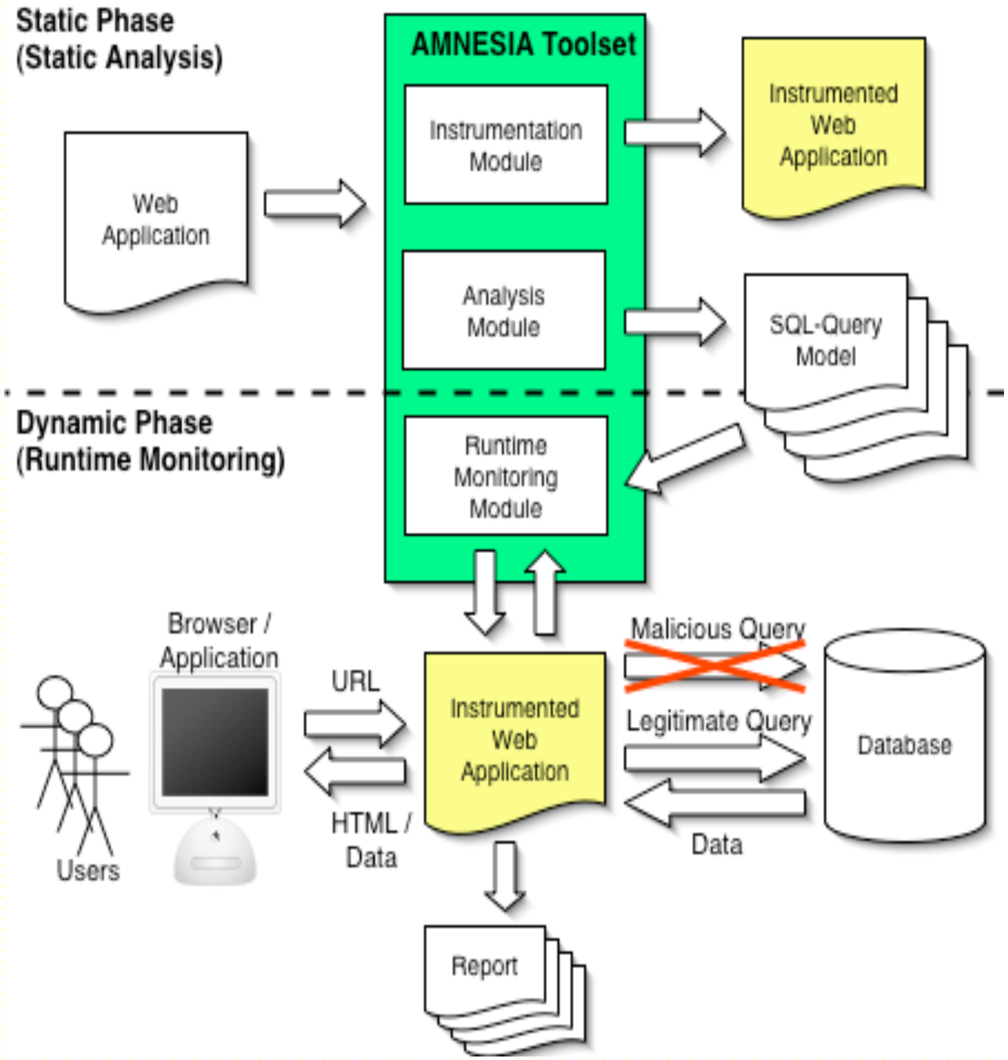
Check queries against SQL query model.




Malicious Usage:

SELECT info FROM userTable WHERE login = ‘ admin ‘ OR 1 = 1 -- ‘ AND pass = ‘ ‘

# AMNESIA Implementation



# Limitations and Assumptions

## Assumption

- Queries created by manipulating strings

## Limitations

- False positives
  - When string analysis is not precise enough
- False negatives
  - When query model includes spurious queries and an attack matches it

# Evaluation: Research Questions

**RQ1**: What percentage of attacks can our technique detect and prevent that would otherwise go undetected and reach the database?

**RQ2**: How much overhead does our technique impose on web applications at runtime?

**RQ3**: What percentage of legitimate accesses does our technique prevent from reaching the database?

# Experiment Setup

| Subject | LOC | Hotspots | Average Automata size |
|---|---|---|---|
| *Checkers* | 5,421 | 5 | 289 (772) |
| *Office Talk* | 4,543 | 40 | 40 (167) |
| *Employee Directory* | 5,658 | 23 | 107 (952) |
| *Bookstore* | 16,959 | 71 | 159 (5,269) |
| *Events* | 7,242 | 31 | 77 (550) |
| *Classifieds* | 10,949 | 34 | 91 (799) |
| *Portal* | 16,453 | 67 | 117 (1,187) |

- Applications are a mix of commercial (5) and student projects (2)
- Attacks and legitimate inputs developed ***independently***
- Attack inputs represent broad range of exploits

# Results: RQ1

| Subject | Unsuccessful | Successful | Detected |
|---|---|---|---|
| *Checkers* | 1195 | 248 | 248 (100%) |
| *Office Talk* | 598 | 160 | 160 (100%) |
| *Employee Directory* | 413 | 280 | 280 (100%) |
| *Bookstore* | 1028 | 182 | 182 (100%) |
| *Events* | 875 | 260 | 260 (100%) |
| *Classifieds* | 823 | 200 | 200 (100%) |
| *Portal* | 880 | 140 | 140 (100%) |

⇒ No false negatives

⇒ Unsuccessful attacks = filtered by application

# Results: RQ2 & RQ3

- Runtime Overhead
  - Less than 1ms.
  - Insignificant compared to cost of network/database access
- No false positives
  - No legitimate input was flagged as SQLIA

# Related Work

- Require learning new API[2,8]
- Customized runtime environments and high overhead[6,9,12,10,11]
- Address only a subset of SQLIA[3,14]
- Limited by machine learning[4,13]
- Overly conservative static analysis[5,7]

# Conclusion

- SQL Injection Attacks (SQLIAs) are a serious threat to DB-based Web Applications
- This technique detects and prevents SQLIAs by combining static analysis and runtime monitoring
  - Fully automated – No human effort required
- Empirical evaluation
  - Commercial applications and real attacks
  - No false positives generated
  - Precise – No false negatives

# References


[1] [Christensen03] A. S. Christensen, A. Moller, and M. I. Schwartzbach. Precise analysis of string expressions. In Proceedings of the 10th International Static Analysis Symposium, volume 2694 of LNCS, pages 1--18. Springer-Verlag, June 2003.

[2] [Cook05] W.R. Cook and S. Rai. Safe Query Objects: Statically Typed Objects as Remotely Executable Queries. ICSE 2005

[3] [Gould04] C. Gould, Z. Su, and P. Devanbu. Static checking of dynamically generated queries in database applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04), pages 645--654, 2004.

[4] [Huang03] Y. W. Huang, S. K. Huang, T. P. Lin, and C. H. Tsai. Web application security assessment by fault injection and behavior monitoring. In Proceedings of the 11th International World Wide Web Conference (WWW 03), May 2003.

[5] [Huang04] Y. W. 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.

[6] [Kc03] G. S. Kc, A. D. Keromytis, and V.Prevelakis. Countering code-injection attacks with instruction-set randomization. In Proceedings of the ACM Conference on Computer and Communications Security, pages 272--280, October 2003.

[7] [Livshits05]

[8] [McClure05] Russell McClure and Ingolf Krüger. SQL DOM: Compile Time Checking of Dynamic SQL Statements. ICSE 05

[9] [Newsome05] James Newsome and Dawn Song. Dynamic Taint Analysis: Automatic Detection, Analysis, and Signature Generation of Exploit Attacks on Commodity Software. In Network and Distributed Systems Security Symposium. Feb 2005.

[10] [Nguyen-Tuong05] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, David Evans. Automatically Hardening Web Applications Using Precise Tainting Information. In Twentieth IFIP International Information Security Conference, May 2005.

[11] [Pietraszek05] T. Pietraszek1 and C.V. Berghe. Defending Against Injection Attacks through Context-Sensitive String Evaluation. RAID 2005

[12] [Scott02] D. Scott and R. Sharp. Abstracting application-level web security. In Proceedings of the 11th International Conference on the World Wide Web, pages 396--407, 2002.

[13] [Valeur05] F. Valeur, D. Mutz, and G. Vigna, Learning-Based Approach to the Detection of SQL Attacks, Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)

[14] [Wassermann04] G. Wassermann and Z. Su. An analysis framework for security in web applications. In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, pages 70--78, October 2004.