A Learning-based Approach to the Detection of SQL Attacks

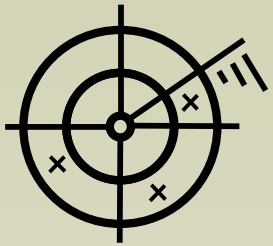
Fredrik Valeur, Darren Mutz, Giovanni Vigna

Reliable Software Group

Department of Computer Science

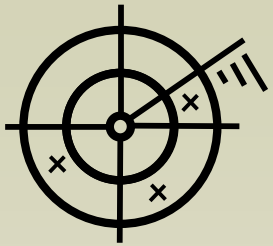
University of California, Santa Barbara

http://www.cs.ucsb.edu/~rsg



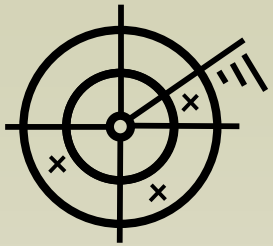
Web-based Applications

- Web applications have become pervasive
  - Use server-side execution mechanisms to access application-specific data
  - Use client-side execution mechanisms to manage user interaction
- Web applications are highly available
  - Deployed by the vast majority of companies, organizations, institutions
  - Can be reached through firewalls
- Infrastructure (Web servers, DB engines) developed by security-aware developers
- Application-specific code often vulnerable
  - Developed in-house to provide custom functionality by programmers with limited security skills
  - Developed under time-to-market pressure ("get the job done" syndrome)
- Result: Web applications are popular attack targets



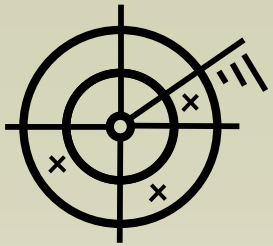
SQL-based Attacks

- SQL injection attacks
  - Unsanitized user input is used to compose an SQL query (e.g., string concatenation of user-provided parameters)
  - Attackers can provide input that contains SQL code and modifies the application behavior
  - These attacks can also be performed in two steps when DB content is used to compose SQL queries
- XSS scripting attack
  - Unsanitized data is stored in the back-end database of a web application
  - Attackers can store scripting code that will be executed in the browser of an unsuspecting user
- Data-centric attacks
  - Unchecked user input values can cause unexpected application behavior
  - Attackers provide unexpected values to trigger anomalous behavior



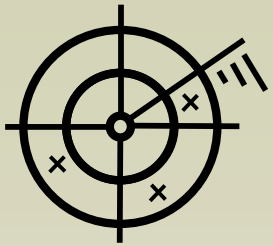
Does It Matter?

Year	Total CVE/CAN	Web-Related	Percentage
1999	1552	257	16.6 %
2000	1203	300	24.9 %
2001	1363	381	28.0 %
2002	1507	538	35.7 %
2003	956	235	24.6 %
2004	1208	318	22.2 %
Total	7861	2045	26.0 %



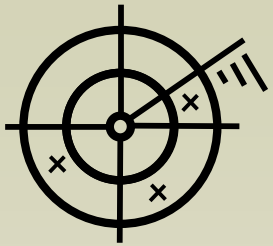
Foiling SQL-based Attacks

- Prevention
  - Access control mechanisms (difficult to "get it right")
  - Code audits (expensive and effort/expertise-intensive)
  - Pen testing (expensive and cannot keep track of fast-changing applications)
- Misuse detection (and response)
  - Snort (network traffic)
  - WebWatcher (web log entries)
  - WebSTAT (network traffic, web log entries, system calls)
- Misuse detection systems are precise and effective but...
  - These system do not analyze the actual SQL query
  - Unforeseen vulnerabilities are introduced by web-based custom applications
  - Developing signatures is time-consuming and requires security expertise



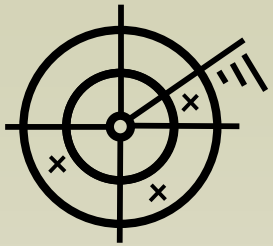
Anomaly-based Detection of SQL Attacks

- Anomaly detection relies on models of expected behavior and detects deviations from the models
- Assumption: Malicious activity generates anomalies
- Assumption: Anomalous behaviour is to be considered malicious
- Advantage: Can detect previously unknown attacks
- Approach: A multi-model, learning-based anomaly detection system to detect SQL-based attacks
  - Developed leveraging the libAnomaly framework developed at UCSB
    - http://www.cs.ucsb.edu/~rsg/libAnomaly



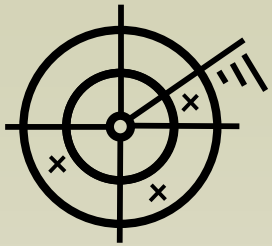
Related Work

- Specification-based anomaly detection
  - The characteristics of "normal behavior" are specified by a human expert
  - Advantage: Reliable models and few false positives
  - Disadvantage: Models can be difficult to write/derive
- Learning-based anomaly detection
  - The characteristics of "normal behavior" are automatically derived from training data
  - Advantage: Reduced expertise-intensive setup
  - Disadvantage: Incomplete, may generate false positives, may be vulnerable to mimicry attacks (e.g., Wagner's and Maxion's works)
- Data mining techniques for network traffic (e.g., S. Stolfo and W. Lee's work)
- Statistical analysis of OS audit records (e.g., D. Denning and A. Valdes)
- Sequence analysis of operating system calls (e.g., S. Forrest's approach)

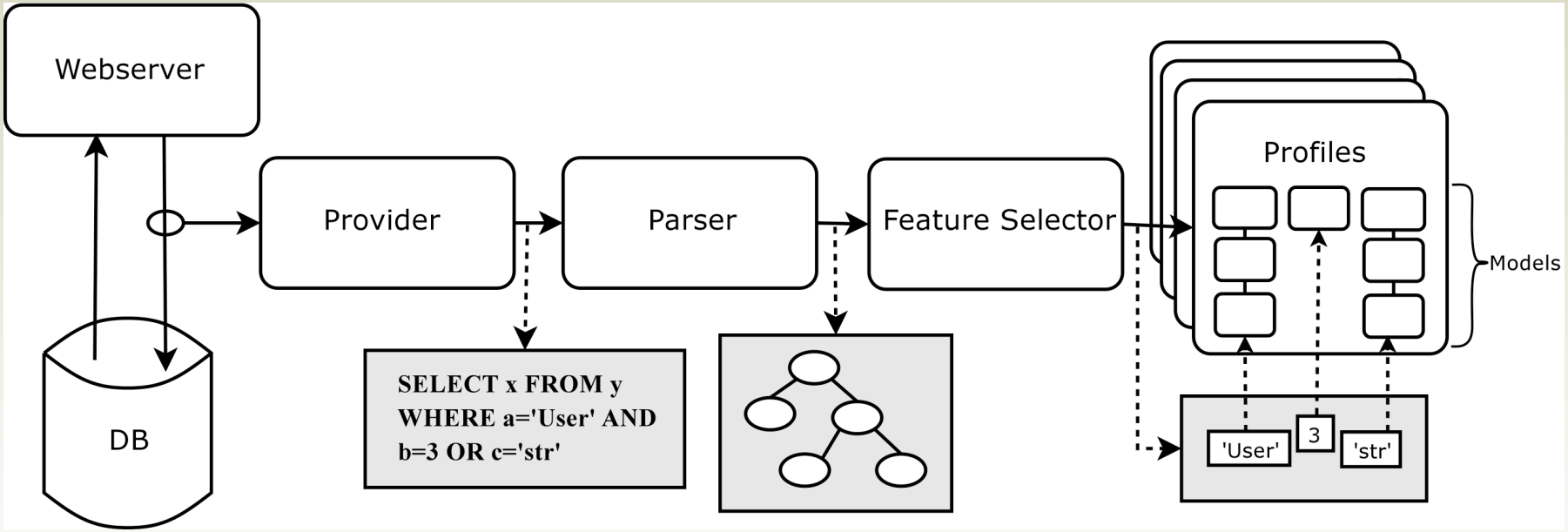


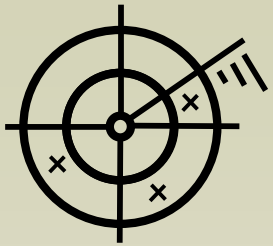
Closely Related Work

- S. Lee et al., "Learning Fingerprints for a Database Intrusion Detection System," ESORICS 2002
  - Learns structural models of acceptable SQL queries
  - Vulnerable to mimicry attacks
- Halfond et al., "Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks," ICSE Workshop on Dynamic Analysis, 2005
  - Uses static analysis to generate models of acceptable SQL queries
  - Cannot address complex code structure
- Some commercial tools provide learning-based mechanisms against SQL-based attacks (difficult to compare because details are not provided)
  - Imperva's SecureSphere



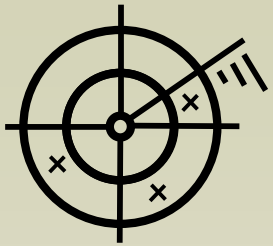
Architecture





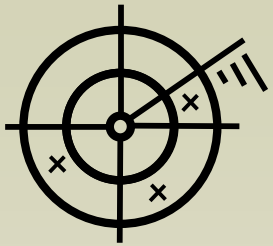
Models and Profiles

- Model: set of procedures used to evaluate a certain feature of an SQL query
  - Single feature: string length
  - Multiple features: relationship between field values
  - Series of queries: time delay between queries
- Profile: association of a model with one or more attributes of a specific query
  - Example: string length model for the *user* attribute of the query used during login



Training

- Models can operate in one of two modes
  - Training
  - Detection
- During training, profiles are established during a two-step training phase
- First phase: captures profiles
- Second phase: determines anomaly thresholds
  - Highest anomaly score is recorded
  - Thresholds set to a value *x%* higher than the highest anomaly score

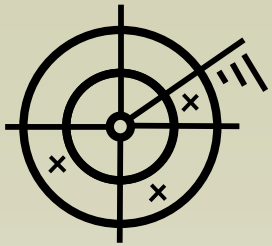


Detection

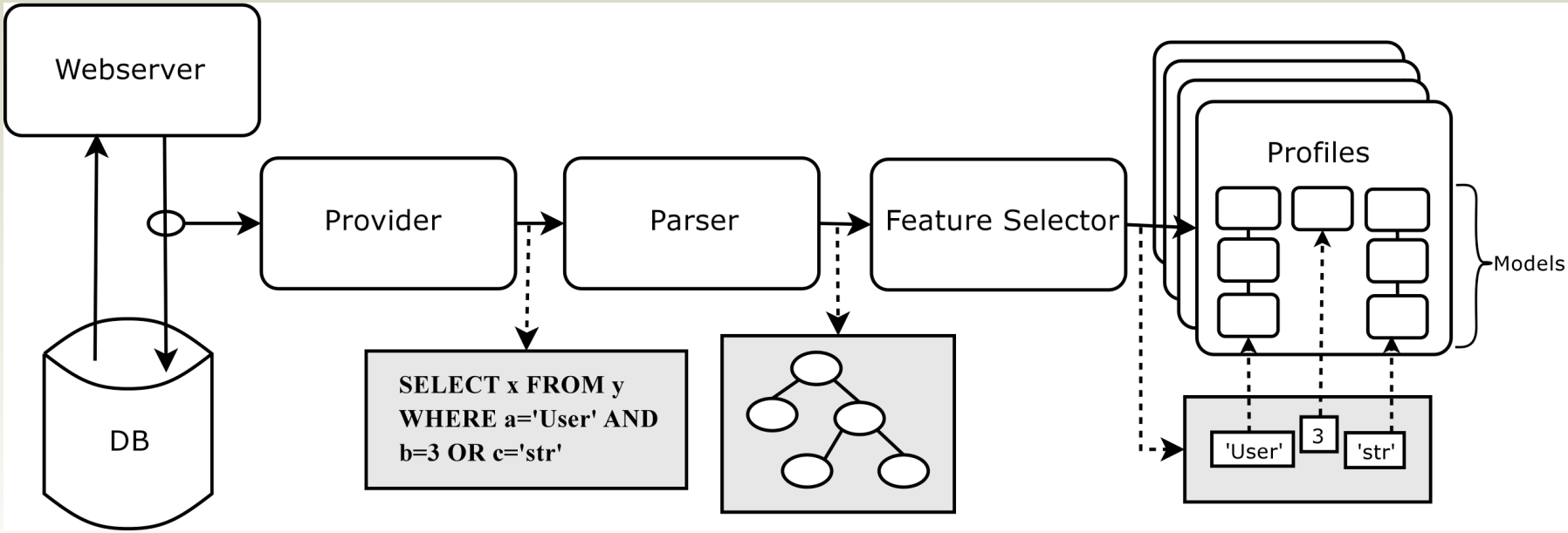
- A model assigns a probability value *p* to a query or an attribute of a query, given an established profile
  - *p = 0 means anomalous*
- The anomaly score of a query is determined by composing the results of the applicable models

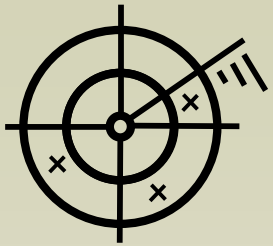
$$-\sum_{m \in Models} \log\,(1 - p_m)$$

- High anomaly score values indicate anomalous queries



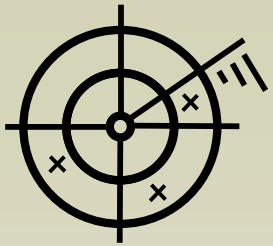
Architecture





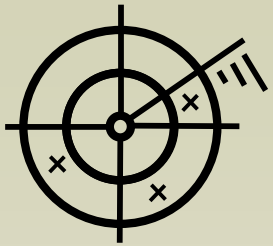
Event Provider

- Responsible for supplying the IDS with a stream of SQL queries
- Does not rely on application-level mechanisms to collect the query data
- Collects the name of the script executing the query
  - Future extensions are planned to include line number
- Implemented by modifying the system libraries that support DB connectivity



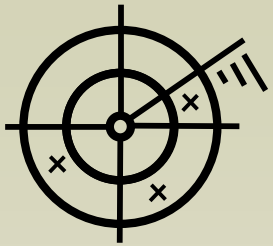
Parser

- Generates a higher-level representation of the query
- Queries are tokenized into keywords and literals
  - Literals are the only fields that should contain user input
- Tokens representing table fields are augmented with a type
- A type table is automatically generated by parsing the database schema
- Each literal's type is used to determine which models can be applied
- New, custom data types can be specified by the user to allow for better characterization (e.g., varchar can be refined to contain XML data)
- Literals' types are inferred by using simple rules
  - Comparison to a typed field
  - Insertion in a typed field of a table



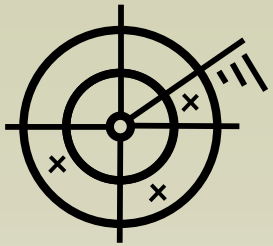
Feature Selector

- The feature selector prepares a query to be evaluated by models
- It generates a skeleton query that represents the structure of the query (i.e., all constants are replaced by place-holders)
- If models are being trained
  - The invoking script and the skeleton are used as a key to lookup the corresponding profile
  - The relevant profile is updated
- If thresholds are being determined
  - The relevant profile is recovered
  - The corresponding models are used to determine an anomaly score
  - The thresholds are updated to allow the event to fit as normal
- If detection is being performed
  - Anomaly score determined as in the threshold-learning phase
  - Queries whose anomaly scores overcome the established threshold are marked as malicious



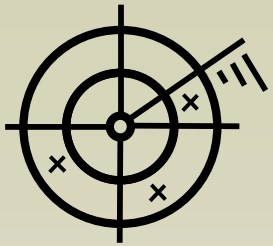
Detection Models

- String length
  - Statistically models the "normal" length for a certain parameter of a specific query (based on Chebyshev inequality)
- String character distribution
  - Statistically models the relative frequencies of characters (based on Pearson's χ^2-test)
- String prefix and suffix matcher
  - Models shared substring values at the beginning and end of strings (e.g., pathnames and extensions)
- String structural inference
  - Generates a probabilistic grammar of the parameter value (based on Stolcke and Omohundro's state-merging technique)
- Token finder
  - Models parameters that assume a finite set of values (based on Kolmogorov-Smirnov non-parametric variant)



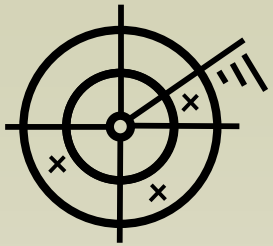
Evaluation

- We evaluated our system using an installation of the PHP-Nuke web portal system
  - Standard LAMP configuration
- Attack-free audit data was generated by
  - Manually operating the web site
  - Using custom bots that simulate user activity
- Data sets
  - Training (44035 queries)
  - Threshold learning (13831 queries)
  - False positive rate estimation (15704 queries)
- Attacks
  - Developed four different SQL-based attacks (0-day) against PHP-Nuke
  - Collected corresponding traces



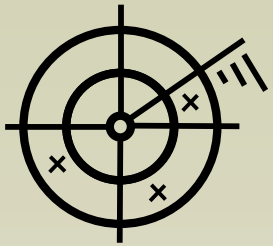
Attacks

- Resetting users' passwords
  - Post data: **`name='; UPDATE nuke_users SET user_password='<new_md5_pass>' WHERE username='<user>';  --`**
  - Result: **`SELECT active, view FROM nuke_modules WHERE title='Statistics'; UPDATE nuke_users SET user_password='<new_md5_pass>' WHERE username='<user>'; --'`**
- Enumerating all users
  - Post data 1: **`name=Your_Account`**
  - Post data 2: **`op=userinfo`**
  - Post data 3: **`username=' OR username LIKE 'A%'; --`**
  - Result: **`SELECT uname FROM nuke_session WHERE uname='' OR username LIKE 'A%'; -- '`**



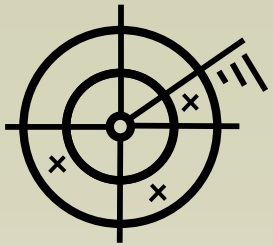
Attacks

- Parallel password guessing
  - Post data 1: `name=Your_Account`
  - Post data 2: `username=' OR user_password = '<md5_pass>';`
  - Post data 3: `user_password=<password>`
  - Result: `SELECT user_password, user id, .... FROM nuke_users WHERE username='' OR user_password = '<md5 password>' ;'`
- Cross-site scripting
  - Referer HTTP header field set to `"onclick="alert(document.domain);"`
  - Result: `INSERT INTO nuke_referer VALUES (NULL, '" onclick="alert(document.domain);"')`
- Notes:
  - Magic quotes were disabled
  - Used bleeding-edge version of MySQL supporting multiple queries separated by semicolon



Results

- All attacks were detected with no false positives
- Running the false positive test (15704 attack-free queries) caused 58 false positives (0.37%)
  - Problem with changing month
- Adding new custom data types ("month" and "year") reduced false positive to just 2 (0.013%)
  - Queries that were not observed in training



Conclusions

- Web applications are vulnerable to attacks against back-end databases
- We developed an anomaly detection system that performs learning-based, multi-model characterization of SQL queries performed by web applications
- Evaluated our tool against a real-world application and real "novel" attacks
- Both detection rate and false positive rate are satisfactory
- Future work
  - More models
  - More testing
  - Integration with webAnomaly and sysAnomaly

Questions?