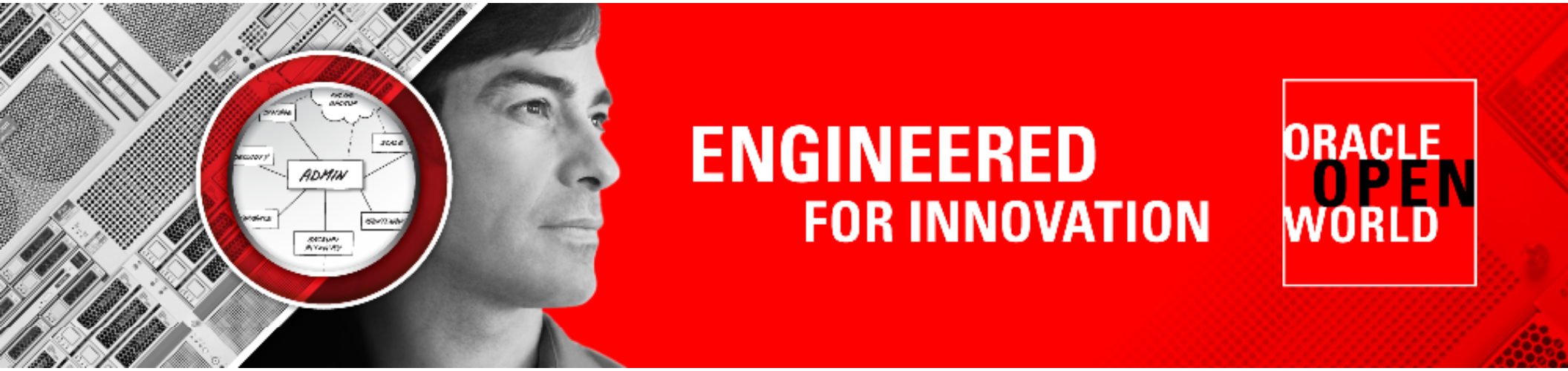

ORACLE®



ORACLE

Oracle Database Firewall: First Line of Defense

Greg Allen
Principal Security Sales Consultant

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

- SQL Injection & Threats to Databases
- Oracle Database Firewall
  - Security Models
  - Policy Enforcement
  - Database Monitoring and Reporting
  - Architecture and Deployment Modes
- Demonstration
- Conclusion
- Q&A

Over 900M Breached Records Resulted from Compromised Database Servers

Type	Category	% Breaches	% Records
Database Server	Servers & Applications	25%	92%
Desktop Computer	End-User Devices	21%	1%

Payment card	Offline Data	18%	<1%
POS server (store controller)	Servers & Applications	11%	<1%
Laptop computer	End-User Devices	7%	<1%
Documents	Offline Data	7%	<1%
POS terminal	End-User Devices	6%	<1%
File server	Servers & Applications	4%	81%
Automated Teller Machine (ATM)	End-User Devices	4%	<1%
FTP server	Servers & Applications	2%	3%
Mail server	Servers & Applications	2%	4%

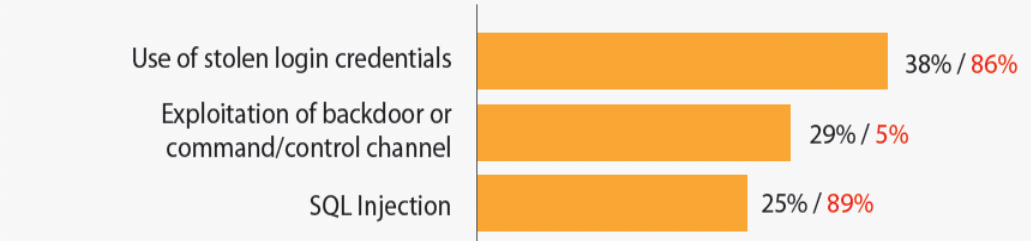
Verizon 2010 Data Breach Investigations Report

"The versatility and effectiveness of SQL injection make it a multi-tool of choice among cyber criminals."

Verizon 2010 Data Breach Investigations Report

- SQL injection attacks against databases are responsible for 89% of breached data
- SQL injection vulnerabilities are endemic
- Require code changes to be made to an application.

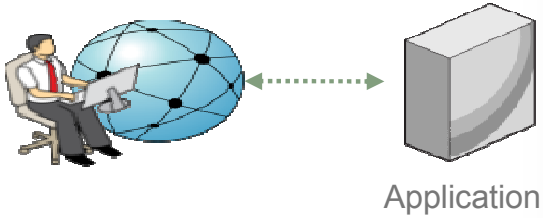
Figure 21. Types of hacking by percent of breaches within Hacking and percent of records



 |

What is a SQL Injection?

Too much trust in users

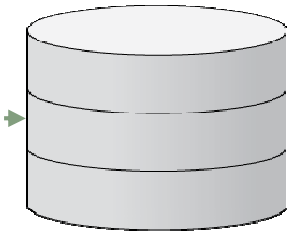



```
SELECT * from stock where
catalog-no =
'PHE8131'
 and location = 1
```

Good ☺

```
SELECT * from stock
where
 catalog-no =
'' union select cardNo, customerId, 0 from Orders --
 and location = 1
```

Bad ☹



- (Mis)users subvert the application to access to the database
- Applications Trust users
- Applications not designed defensively
- Applications are given high levels of privilege
- Each application is unique

How Do Applications Talk to Their Database?

Assembling a *Normal* SQL statement

Parameters for SQL come from user input (e.g. a web browser).
The application layer accepts the values for `catalog-no` and `location` (‘`PHE8131`’, ‘`1`’) and pastes them into the pre-canned query template.

```
SELECT * from stock where catalog-no = '_______' and location =
```

Output:

Description	*Price*	*# in Stock*
Star Trek - The Next Generation Season 2	39.35	15
Star Trek - The Next Generation Season 3	39.35	12
Star Trek - The Next Generation Season 4	39.35	13
Star Trek - The Next Generation Season 5	39.35	17

How Do Applications Talk to Their Database?

Assembling an *abnormal* SQL statement – SQL Injection

Instead of inputting a normal value for `catalog-no`, the user enters

```
' union select cardNo, customerId, 0 from Orders --
```

And the database receives the following query

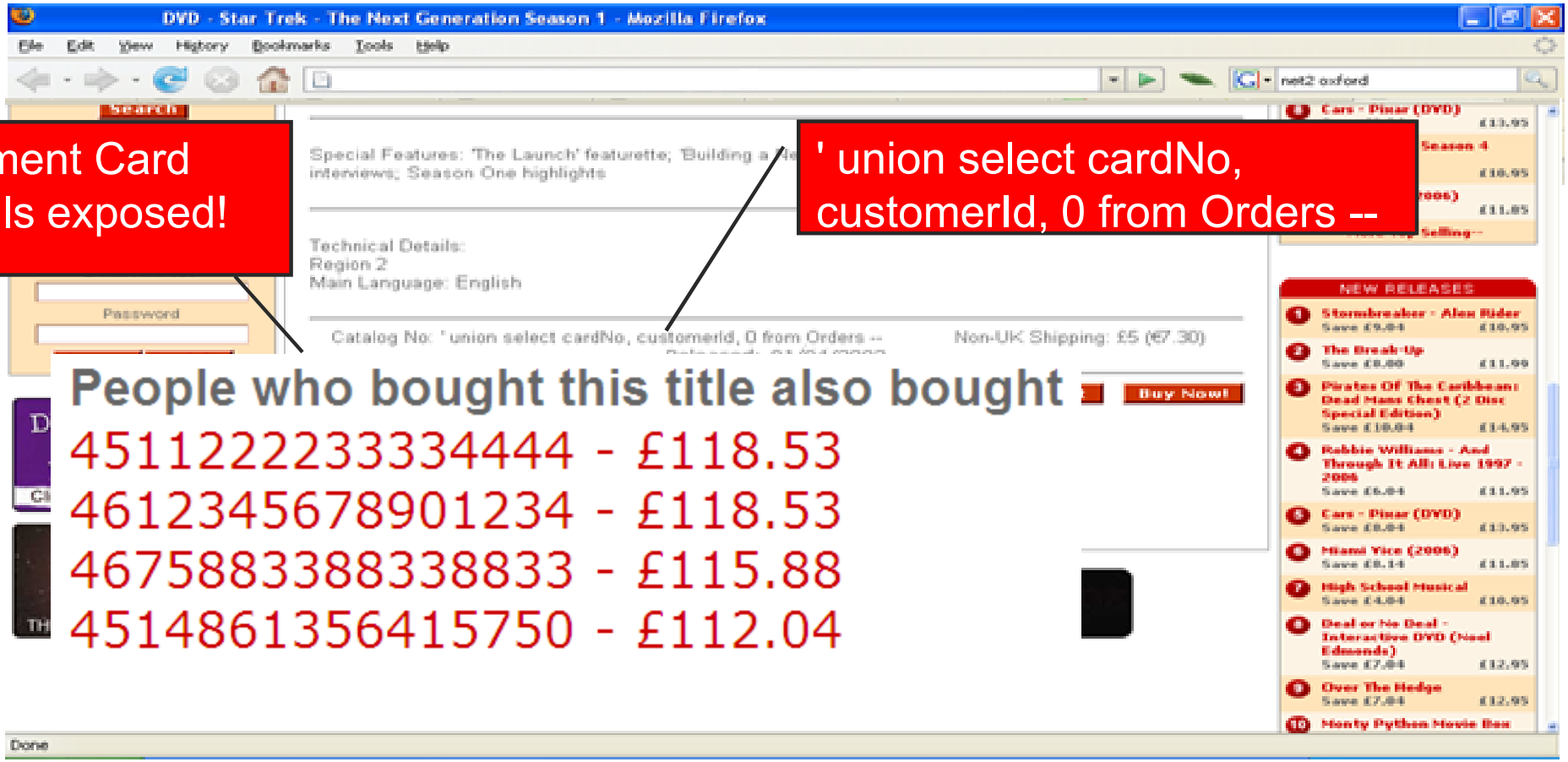
```
SELECT * from stock where catalog-no = '_______' and location =
_____________________________' and location =
```

Output:

Description			*Price*	*# in Stock*
4511222233334444	11853	0		
4612345678901234	11853	0		
4675883388338833	11588	0		
4514861356415750	11204	0		

Payment Card details

What does the attacker actually see?



 |

More SQL Injections

Endless alternatives

```
SELECT * from stock where catalog-no = 'PHE8131' and location = 1
```

```
SELECT * from stock where catalog-no = ''--' and location = 1
```

```
SELECT * from stock where catalog-no = '' having 1=1 -- ' and location = 1
```

```
SELECT * from stock where catalog-no = '' order by 4--' and location = 1
```

```
SELECT * from stock where catalog-no = '' union select cardNo,customerId,0
from Orders where name = 'John Smith'--' and location = 1
```

```
SELECT * from stock where catalog-no = '' union select min(cardNo),1,0 from
Orders where cardNo > '0'--' and location = 1
```

Challenges of SQL Injections

- How Do You Catch Them?
  - Must detect anomalous SQL
  - Need to ensure detection is accurate (i.e. do not have false positive nor false negative errors)
  - Must be performant (i.e. can be done in real-time)
- What Kind of Detection Engine Do I Need?
  - Why is Accuracy Important in a Detection Engine?
  - SQL Grammar versus Regular Expressions
  - White Lists versus Black Lists

Detection Engines

Regular Expressions Based Approach

- Regex
  - A pattern describing a set of strings

- With regular expressions you can:
  - Find out whether a certain pattern occurs in the text
  - Locate strings matching a pattern and remove them or replace them with something else
  - Extract the strings matching the pattern

[Source: Andrei Zmievski, Yahoo! Inc.]

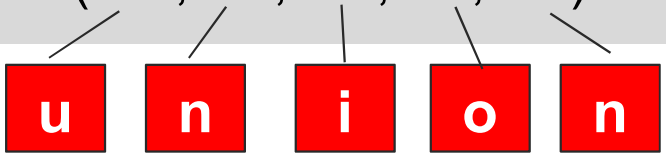
String Matching Keywords is Inadequate

- **union** is NOT universally bad when next to this **select**

```
SELECT lastname from boys
union
SELECT lastname from girls
```

- **union** without “saying it”

```
uni/* */on
u/* */nion
char(117,110,105,111,110)
```



Can you *'Tune'* Regular Expressions?

- **union** is bad when it appears ***near*** **select**

u(?:nion\b.{1,100}?\bselect

```
"(?:\b(?:(?:s(?:elect\b(?:.{1,100}?\b(?:(?:length|count|top)\b.{1,100}?\bfrom|from\b.{1,100}?\bwhere)
|.*?\b(?:d(?:ump\b.*\bfrom|ata_type)|(?:to_(?:numbe|cha)|inst)r))|p_(?:(?:addextendedpro|sqlexe)c|(?:
oacreat|prepar)e|execute(?:sql)?|makewebtask)|ql_(?:longvarchar|variant))|xp_(?:reg(?:re(?:movemultis
tring|ad)|delete(?:value|key)|enum(?:value|key)s|addmultistring|write)|e(?:xecresultset|numdsn)|(?:te
rminat|dirtre)e|availablemedia|loginconfig|cmdshell|filelist|makecab|ntsec)|u(?:nion\b.{1,100}?\bsele
ct|tl_(?:file|http))|group\b.*\bby\b.{1,100}?\bhaving|d(?:elete\b\W*?\bfrom|bms_java)|load\b\W*?\bdat
a\b.*\binfile|(?:n?varcha|tbcreato)r)\b|i(?:n(?:to\b\W*?\b(?:dump|out)file|sert\b\W*?\binto|ner\b\W*?
\bjoin)\b|(?:f(?:\b\W*?\(\W*?\bbenchmark|null\b)|snull\b)\W*?\()|a(?:nd\b
?(?:\d{1,10}|[\'\"][^=]{1,10}[\'\"]) ?[=<>]+|utonomous_transaction\b)|o(?:r\b
?(?:\d{1,10}|[\'\"][^=]{1,10}[\'\"]) ?[=<>]+|pen(?:rowset|query)\b)|having\b
?(?:\d{1,10}|[\'\"][^=]{1,10}[\'\"])
?[=<>]+|print\b\W*?\@\@|cast\b\W*?\()|(?:;\W*?\b(?:shutdown|drop)|\@\@version)\b|'(?:s(?:qloledb|a)|m
sdasql|dbo)')"
```

[Source: ModSecurity, Web Application Firewall]

- Is this comprehensible or manageable?

 |

Detection Engines

Issues with Regular Expressions

- Fails to understand meaning, motives and intentions of SQL when you just use strings and text
- Good Statement

```
SELECT * from stock where catalog-no = 'PHE8131' and location = 1
```

- Bad Statement – SQL injection

```
SELECT * from stock where catalog-no = '' union select cardNo,
customerId, 0 from Orders --' and location = 1
```

Understanding SQL

- SQL is a language with about 400 key words and a strict grammar structure

```
UPDATE tbl_users SET comments = 'The user has asked for another
account_no, and wishes to be billed for services between 1/27/2009
and 2/2/2009, and wants to know where the invoice should be sent
to. She will select the new service level agreement to run from
3/7/2009 next month' WHERE id = 'A15431029';
```

KEY WORDS	SCHEMA	DATA	OPERATORS

- Grammatical context necessary to prevent false categorization

Anomaly Detection

Why is Accuracy Important?

- 1,000 transactions per second = 86 Million per day
- 0.001% false positive rate = 27,000 audit errors per month
- High performance run-time matching ensure only appropriate SQL interactions are sent to a database
  - False positives detects when it should not
  - False negatives avoid detection

0.0001% False Negative Rate Result In
86 Potential Attacks Per Day!

Oracle Database Firewall

Meets The Challenges of SQL Accuracy

- Regular expressions
  - Pattern matching does not understand SQL intention
  - Can generate false positives and non-detection
  - High maintenance
- Oracle Database Firewall Grammar-based detection engine
  - Clusters are deterministic and provide accurate policy application
  - Speed of lookup is constant regardless of the number of clusters in the policy
  - By understanding the SQL grammar, SQL injection and other out-of-policy SQL are detected as anomalies and blocked

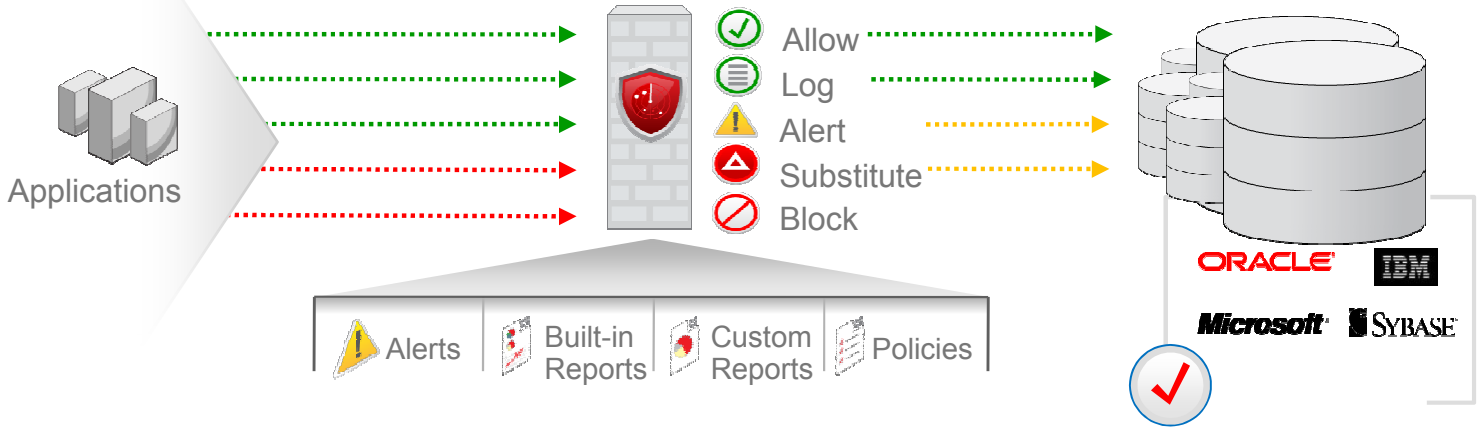
Oracle Database Firewall

Next Generation Detection Framework

- Use the same level of language power to **detect** as to **transmit** SQL
  - This provides **accuracy**
- White-listing approach only allows those SQL to be sent to the database
  - Learn from the SQL you wish to control
  - Go beyond syntax and get close to the meaning or intent or motives of the SQL
- SQL is analyzed and grouped into clusters
  - Only need to record the SQL one time and read many times
  - Eliminate false negatives

Oracle Database Firewall

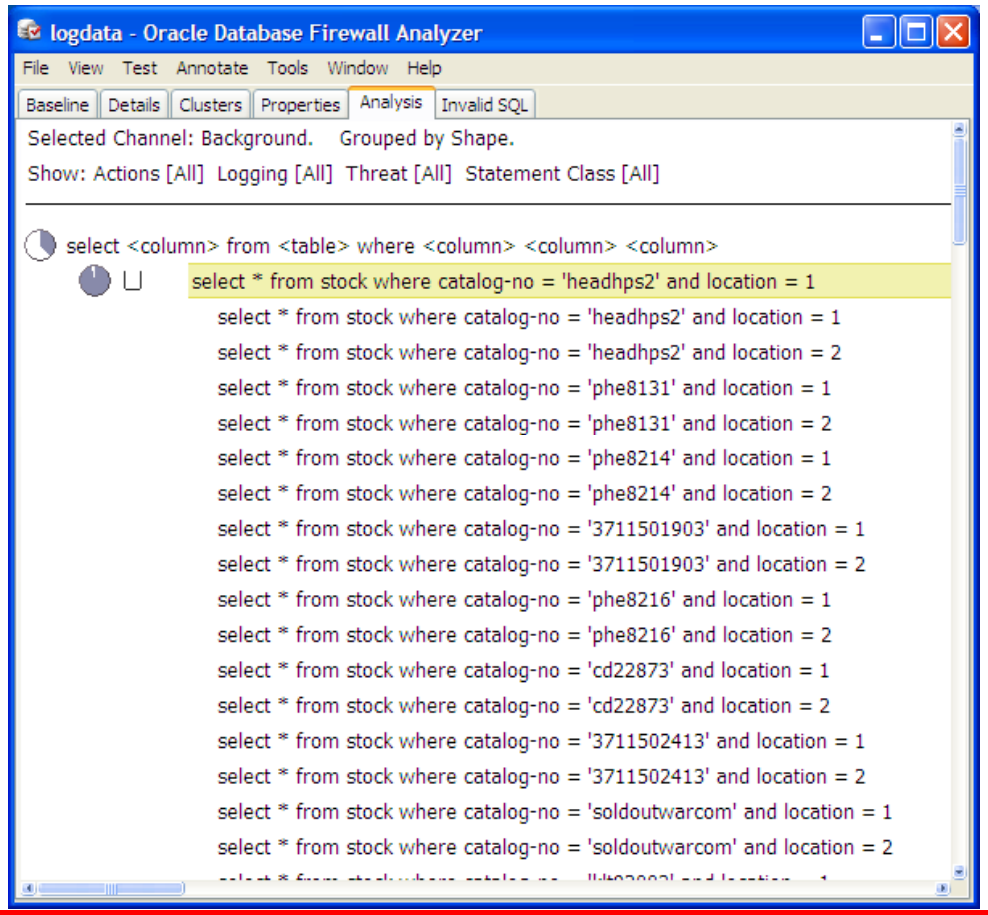
First Line of Defense



- Monitor and control database activity on the network
- Prevent
  - SQL injections
  - Unauthorized database access
  - Misuse of database privilege
- Capture and log database interactions for forensic analysis and compliance reporting

Grammar-based Clustering

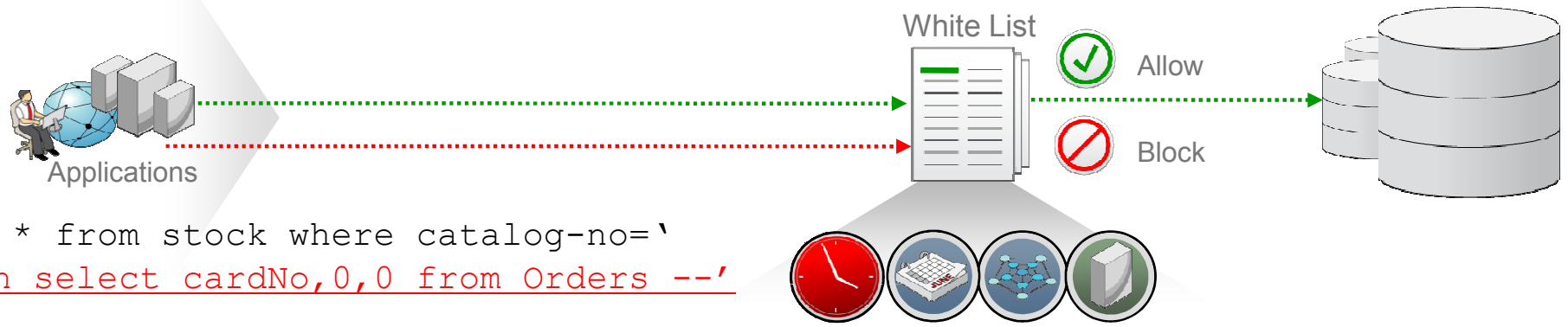
- Thousands of individual statements transformed into manageable clusters
- Anomalous statements produce a new cluster and are instantly detected (e.g. blocked)



 |

Positive Security Model

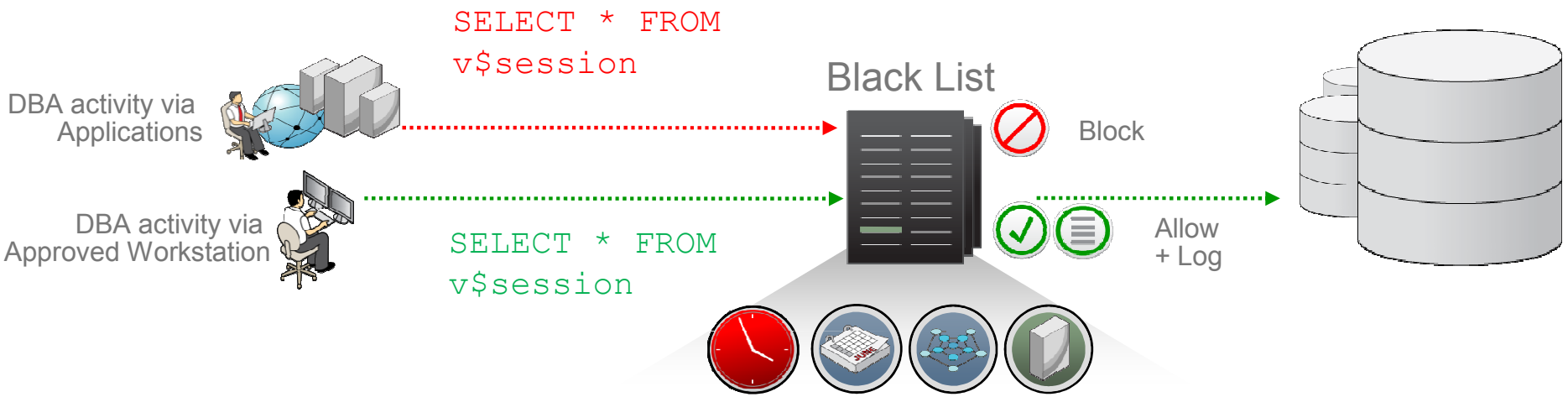
```
SELECT * from stock where catalog-no='PHE8131'
```




```
SELECT * from stock where catalog-no='
' union select cardNo,0,0 from Orders --'
```

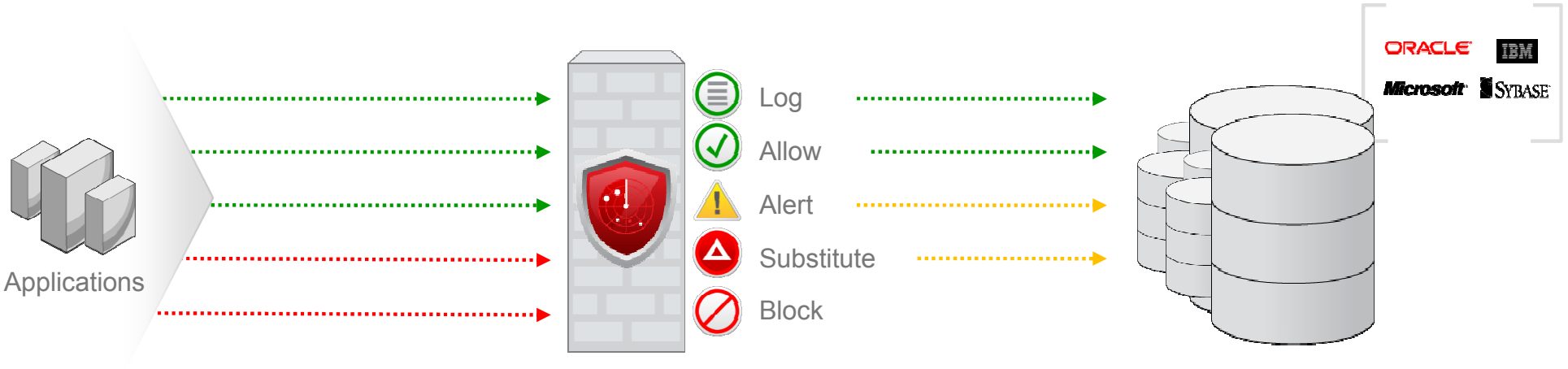
- "Allowed" behavior can be defined for any user or application
- Automated whitelist generation for any application
- Many factors to define policy (e.g. network, application, etc)
- Out-of-policy Database network interactions instantly blocked

Negative Security Model



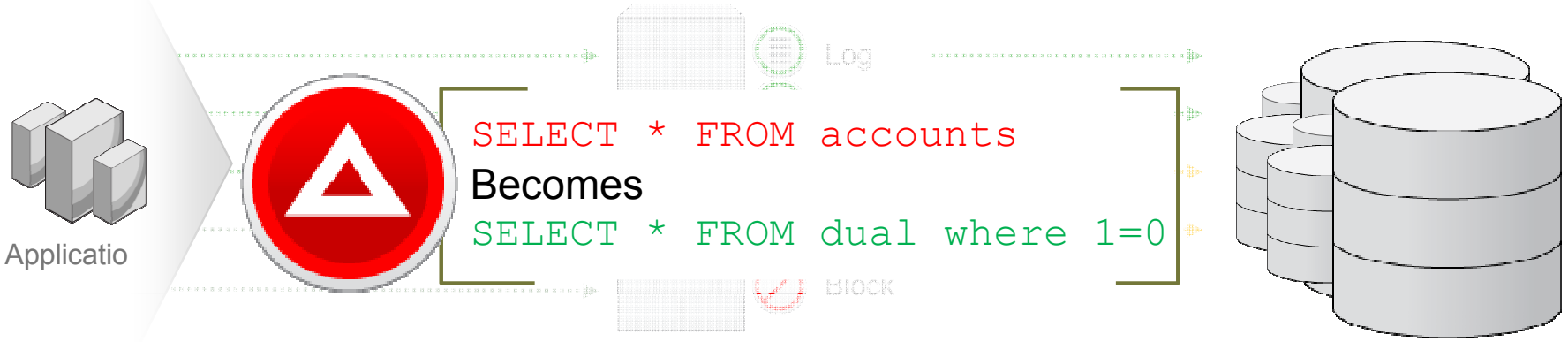
- Stop specific unwanted SQL interactions, user or schema access
- Ensures database interactions originate from appropriate sources
- Blacklist can take into account built-in factors such as time of day, day of week, network, application, etc
- Provide flexibility to authorized DBAs to do their job whilst monitoring usage

Policy Enforcement



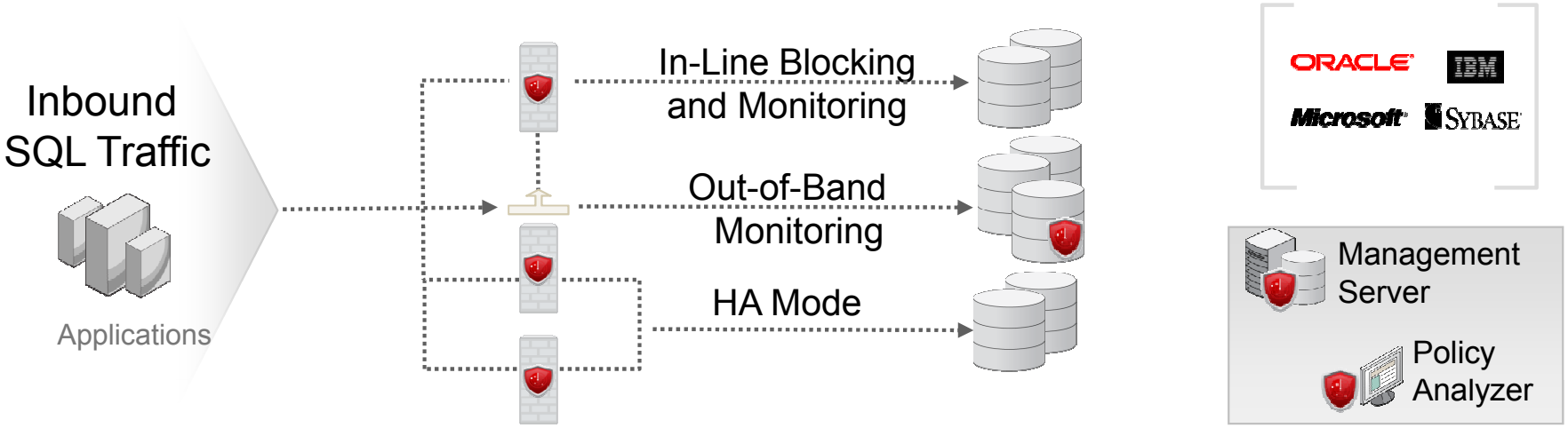
- Innovative SQL grammar technology reduces millions of SQL statements into a small number of SQL characteristics or “clusters”
- Superior performance and policy scalability
- Highly accurate without costly and time consuming false positives errors
- Flexible enforcement at SQL level: block, substitute, alert and pass, log only

Statement Substitution



- Application – Database connection pools cannot be disconnected
- Unique graceful blocking achieved by substituting out-of-policy statement with predefined benign statement
- Database guaranteed protected from inappropriate statement whilst the network session remains open

Deployment Architecture

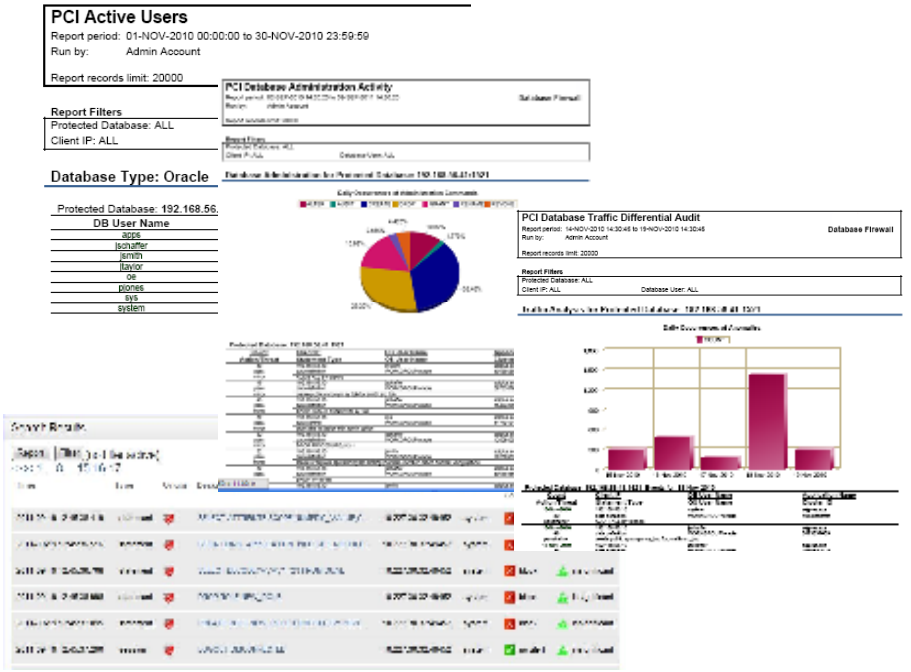


- Application and database vendor agnostic
- Deployment modes:
  - Inline, Out-of-Band, Optional Host Based Agents
  - Policy enforcement separated from policy management and reporting
  - High Availability
- Software solution based on hardened OE Linux and Intel for flexibility and scalability

Database Monitoring and Compliance Reporting

- Full Activity Report
- Database Administration
- Active Users
- Differential Audit
- Data Modification Detail

…etc

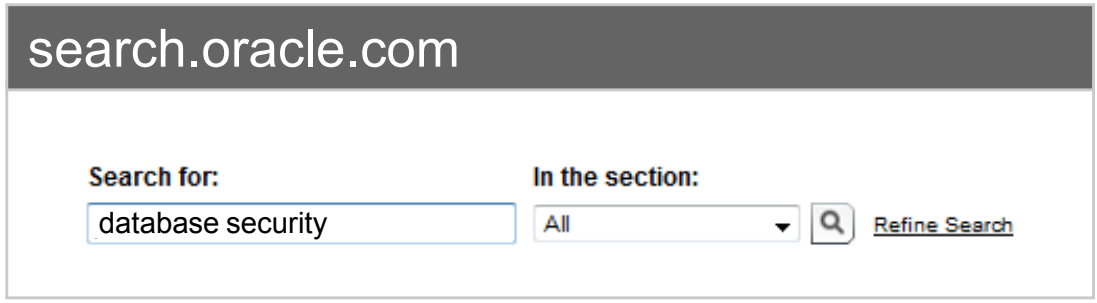


 |

Oracle Database Firewall -- First Line of Defense

DEMONSTRATION

For More Information



or

oracle.com/database/security
oracle.com/goto/database/firewall

Q&A

Hardware and Software

ORACLE

Engineered to Work Together

