CORE LABS RESEARCH & DEVELOPMENT

HACK.LU 08 22-24 OCTOBER

gFuzz: An instrumented Web application fuzzing environment

Ezequiel D. Gutesman
Corelabs
Core Security Technologies

Objectives

- Present a working tool (prototype) to test the security of a given web application. The tool tests for (SQL) injection attacks:
  - From the attacker's perspective
  - Intended to be included in QA process & security audits. Bringing precise information about potential security flaws. Not limited to security experts
  - Has high(er) accuracy than plain fuzzing and automated static analysis by themselves
  - Technique:
    - Fuzzing
    - Instrumentation

Agenda

- (quick!) Web application security overview
- SQL-injection attacks inside-out
- Fuzzing and gFuzz
- Detecting *A*n**O** ɯ AliEs with gFuzz
- Reporting
- Demo
- Future work

Agenda

- (**quick!**) **Web application security overview**
- SQL-injection attacks inside-out
- Fuzzing and gFuzz
- Detecting *A*n**O** ɯ AliEs with gFuzz
- Reporting
- Demo
- Future work

Why Web Applications?

- Common entry point for back-end system and database access
- Widely used
- Easy to develop
  - Scripting languages
  - Inexperienced programmers are not security-aware
- Difficult to (fuzz + validate) errors with low false positive rate

Top Vulnerabilities (From OWASP Top 10 - 2007)

- XSS
- **Injection Vulns (particularly SQL)**
- Malicious File execution
- Insecure Direct Object Reference
- CSRF
- Information Leakage and Error handling
- Broken auth. , session management

Consequences (SQL-injection)

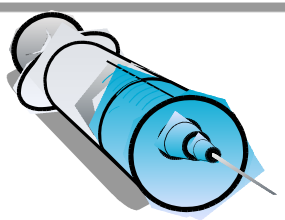
- Data theft
- Data unavailability
- Data alteration
- Money losses
- And much more

Agenda

- (quick!) Web application security overview
- **SQL-injection attacks inside-out**
- Fuzzing and gFuzz
- Detecting AnOωAliEs with gFuzz
- Reporting
- Demo
- Future work

SQL injection facts

- It <u>is</u> an injection attack



  - It happens when (malicious) input sent by an attacker reaches the back-end DBMS engine
  - The attacker can execute queries which were “supposedly” not allowed.

- Are widely known inside the security community
  - Yet, developers still fail in avoiding them

The SQL injection problem: Basic idea

User-supplied data

```
<?php

$client_id = $_POST["id"];

$client = mysql_query("SELECT * FROM clients WHERE id = " . $client_id);
```

Direct usage to query database

Supplying data

SERVER

3 | Query client

CLIENT

$_POST["id"] ← 3

SELECT * FROM clients WHERE id = 3

client.id	client.name
3	John Doe

```
<?php

$client_id = $_POST["id"];
$client = mysql_query("SELECT * FROM clients WHERE id = " . $client_id);
```

Supplying [offensive] data

SERVER

0 or 1=1 | Query client

CLIENT

```
$_POST["id"] ← 0 or 1=1
```

```
SELECT * FROM clients WHERE id = 0 or 1=1
```

client.id	client.name
1	George W
3	John Doe
4	Martin Green
5	Joshua B
76	Ellen Grant
8	Mark Twain

```php
<?php

$client_id = $_POST["id"];
$client = mysql_query("SELECT * FROM clients WHERE id = " . $client_id);
```

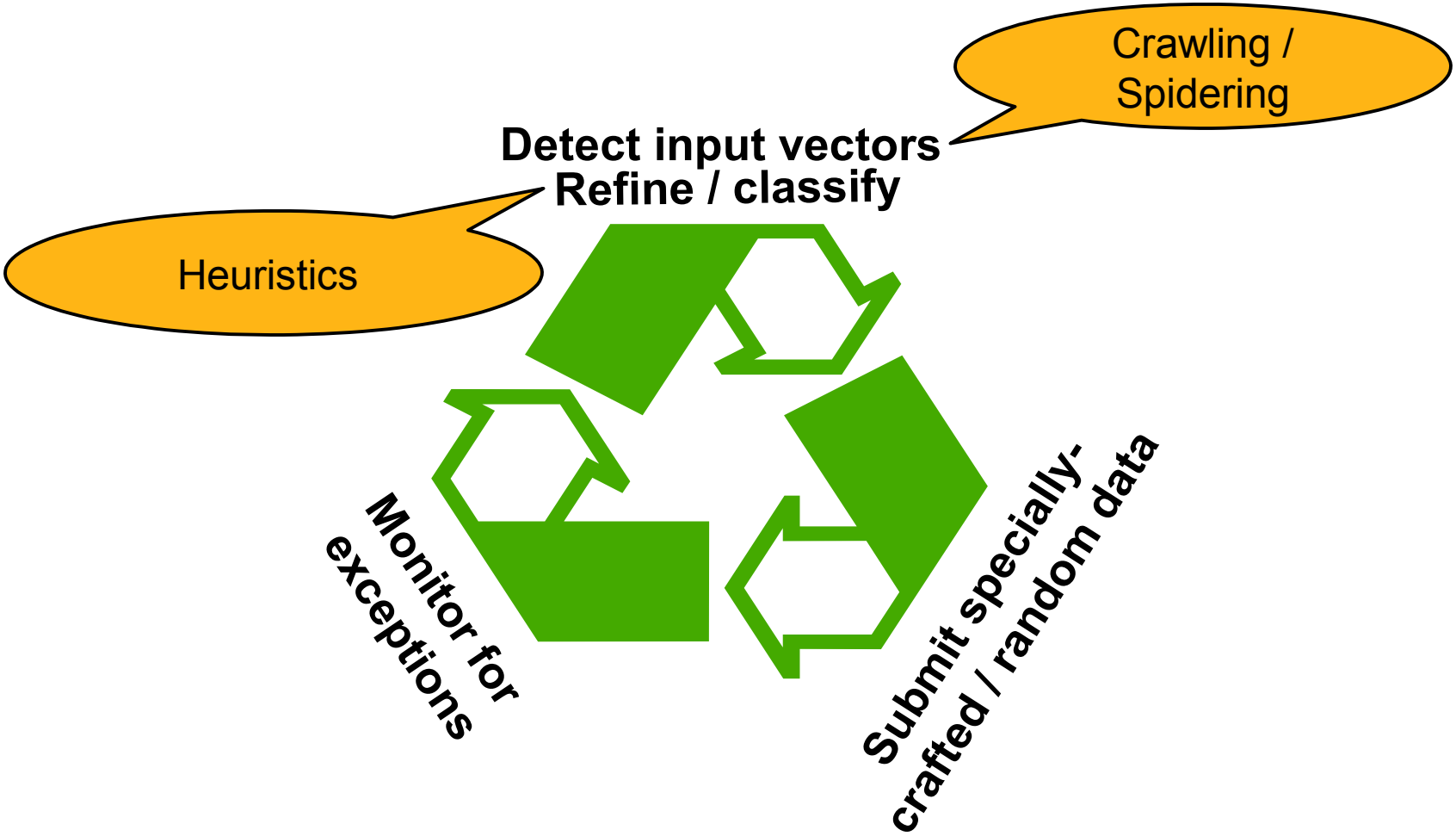
Countermeasure technologies

- Web Application firewalls (& IDS - IPS)
- Static code analysis tools
- Dynamic code analysis tools
- Scanners
- Code audits

Agenda

- (quick!) Web application security overview
- SQL-injection attacks inside-out
- **Fuzzing and gFuzz**
- Detecting AnOmAliEs with gFuzz
- Reporting
- Demo
- Future work

Fuzzing (general)



Web Application fuzzing

- Exception monitoring is not trivial
  - Which are “REAL exceptions”?
- Classification is not trivial
  - Difficult to distinguish between real vulns and false positives (or negatives)

- Validation and discovery heuristics are commonly used
  - Error message detection
  - Sent text reflected
  - Timing, and other
- Relating Fuzz vectors with exceptions and vulns is difficult

gFuzz's approach

Fuzzing

+

Character-grained taint analysis (aka. Core GRASP)

+

Grammar-based analysis

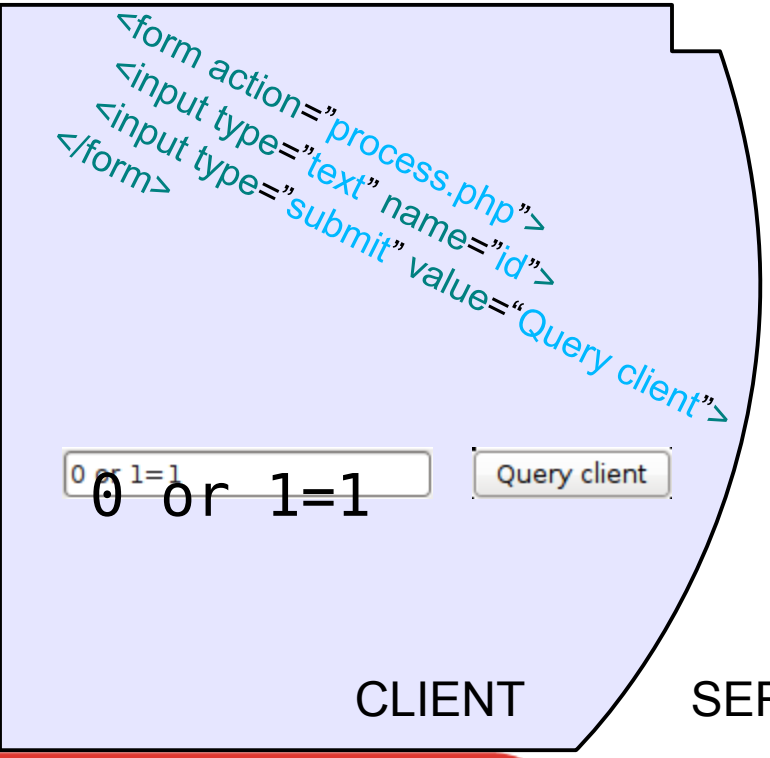
A LOT of information!

Agenda

- (quick!) Web application security overview
- SQL-injection attacks inside-out
- Fuzzing and gFuzz
- **Detecting *A*nO ɯ AliEs with gFuzz**
- Reporting
- Demo
- Future work

Character-grained taint analysis

- Run-time instrumentation
- It “paints” attacker-controlled characters as tainted and propagates taint information during execution.



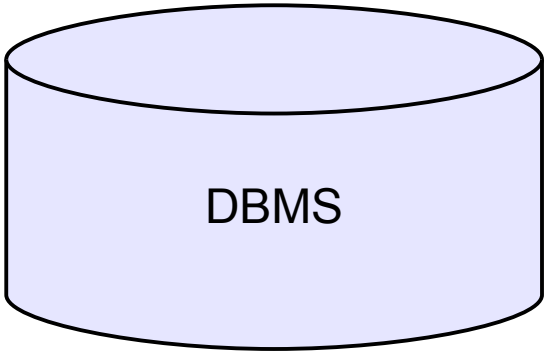
0 or 1=1

Character-grained taint analysis (cont)


```
SELECT * FROM clients
WHERE user like 'john'
AND password = password( 'aaa' )
OR 1 = 1; --)
```

SELECT * FROM clients WHERE id = 0 or 1=1

Scripting language Interpreter (PHP)



Character-grained taint analysis & gFuzz

- Data is marked from untrusted sources (e.g., GET, POST)

- Taint marks are propagated between string operations during execution

- GRASP sends information about executed queries to gFuzz (from inside the interpreter!)

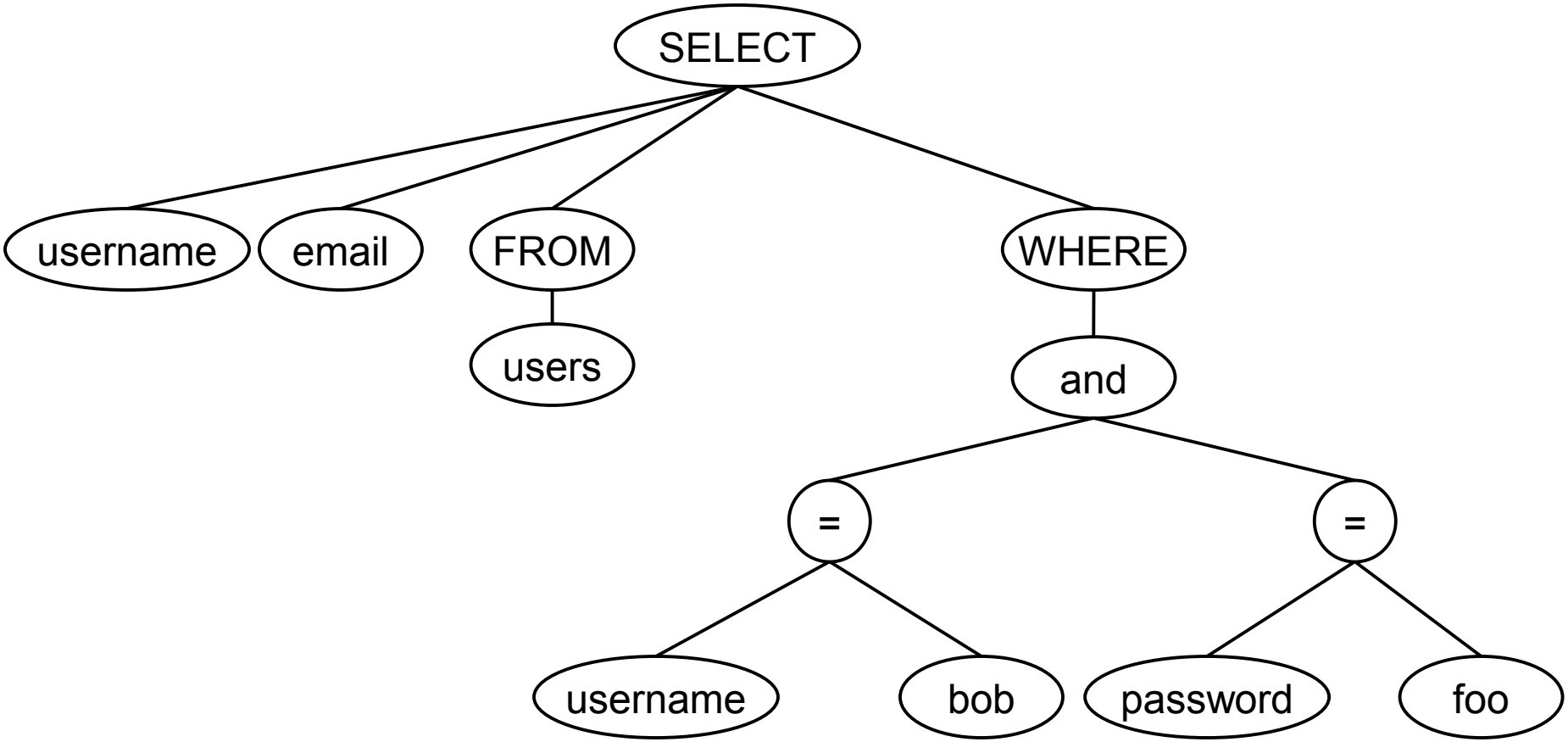
gFuzz entry sent by GRASP

```
<GRASP_FUZZ_ENTRY>
  <GRASP_QUERY_ID>
    /location/of/the/executed/file/userlogin.php:40
  </GRASP_QUERY_ID>
  <GRASP_FUZZ_IS_ATTACK>0</GRASP_FUZZ_IS_ATTACK>
  <GRASP_FUZZ_QUERY>
   SELECT name,email FROM users WHERE username='bob' and password='foo'
  </GRASP_FUZZ_QUERY>
  <GRASP_FUZZ_QUERY_MARK>
   ..............................................XXX................XXX.
  </GRASP_FUZZ_QUERY_MARK>
</GRASP_FUZZ_ENTRY>
```

gFuzz entry sent by GRASP

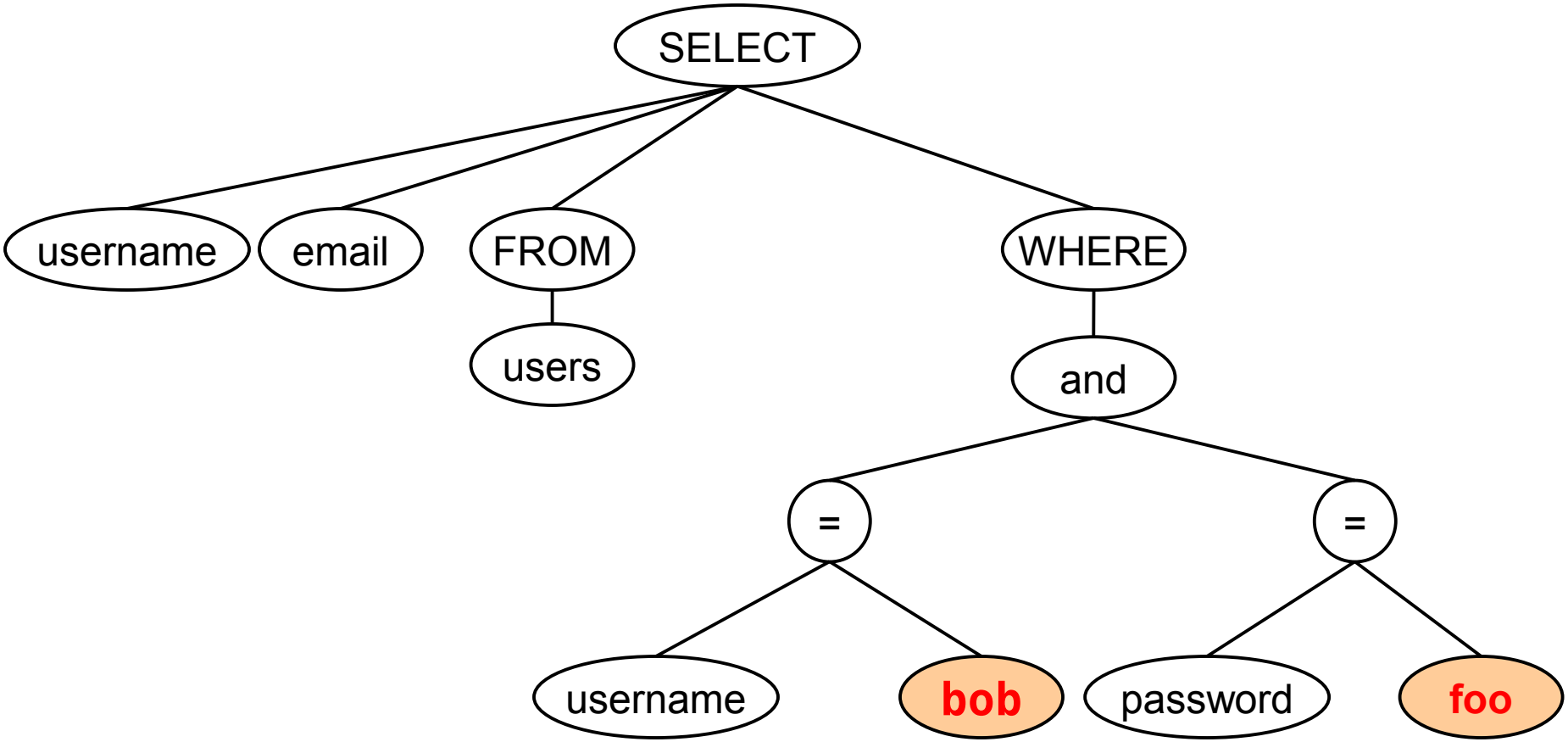
```
<GRASP_FUZZ_ENTRY>
  <GRASP_QUERY_ID>
<GRASP_FUZZ_QUERY>
 SELECT name,email FROM users WHERE username='bob' and password='foo'
</GRASP_FUZZ_QUERY>
<GRASP_FUZZ_QUERY_MARK>
 ..........................................XXX................XXX.
  <GRASP_FUZZ_QUERY_MARK>
   ..........................................XXX...............XXX.
  </GRASP_FUZZ_QUERY_MARK>
</GRASP_FUZZ_ENTRY>
```

Grammatical analysis of SQL queries



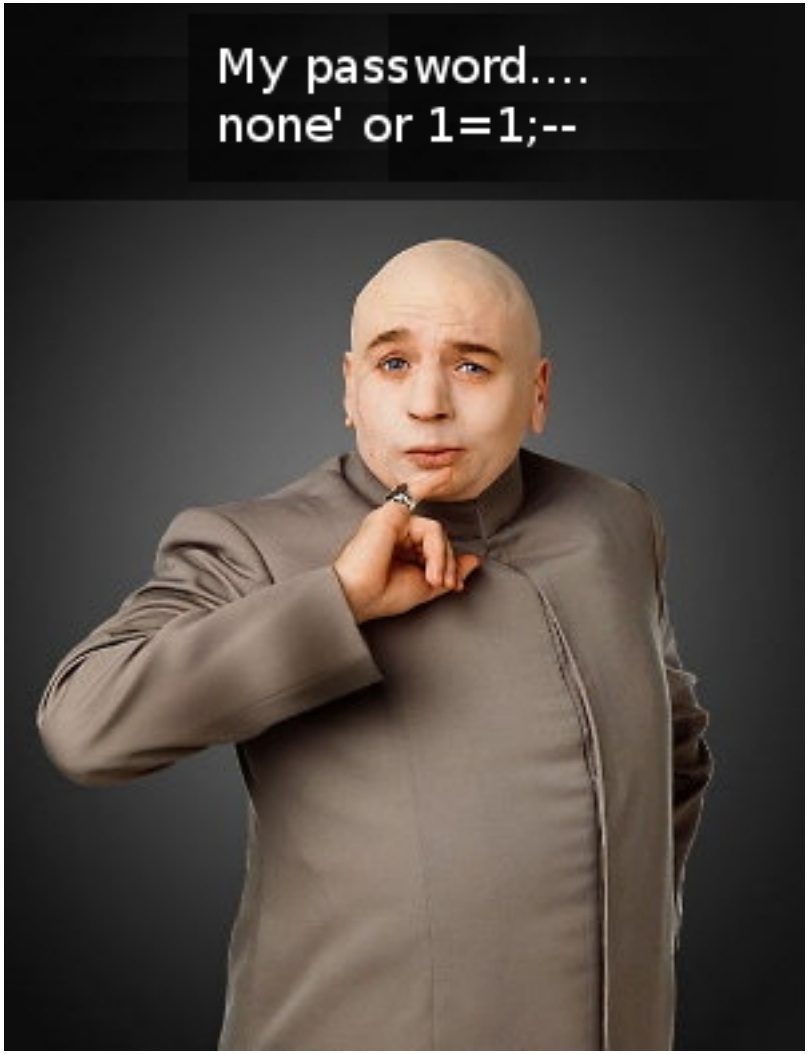

```
SELECT name,email FROM users
WHERE username='bob' and password='foo'
```

Grammatical analysis + taint marks

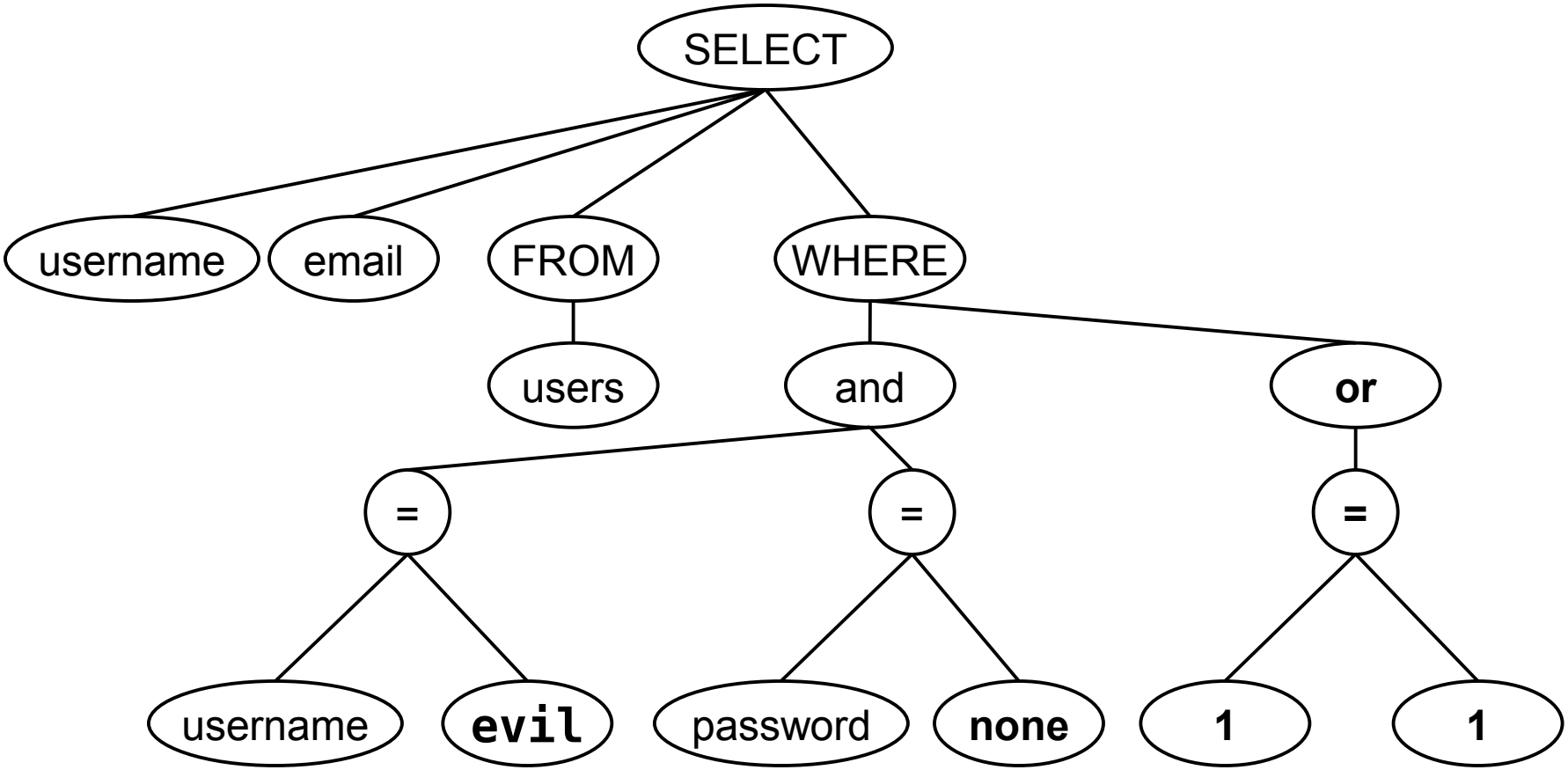



```
SELECT name,email FROM users
WHERE username='bob' and password='foo'
```

Evil inputs...

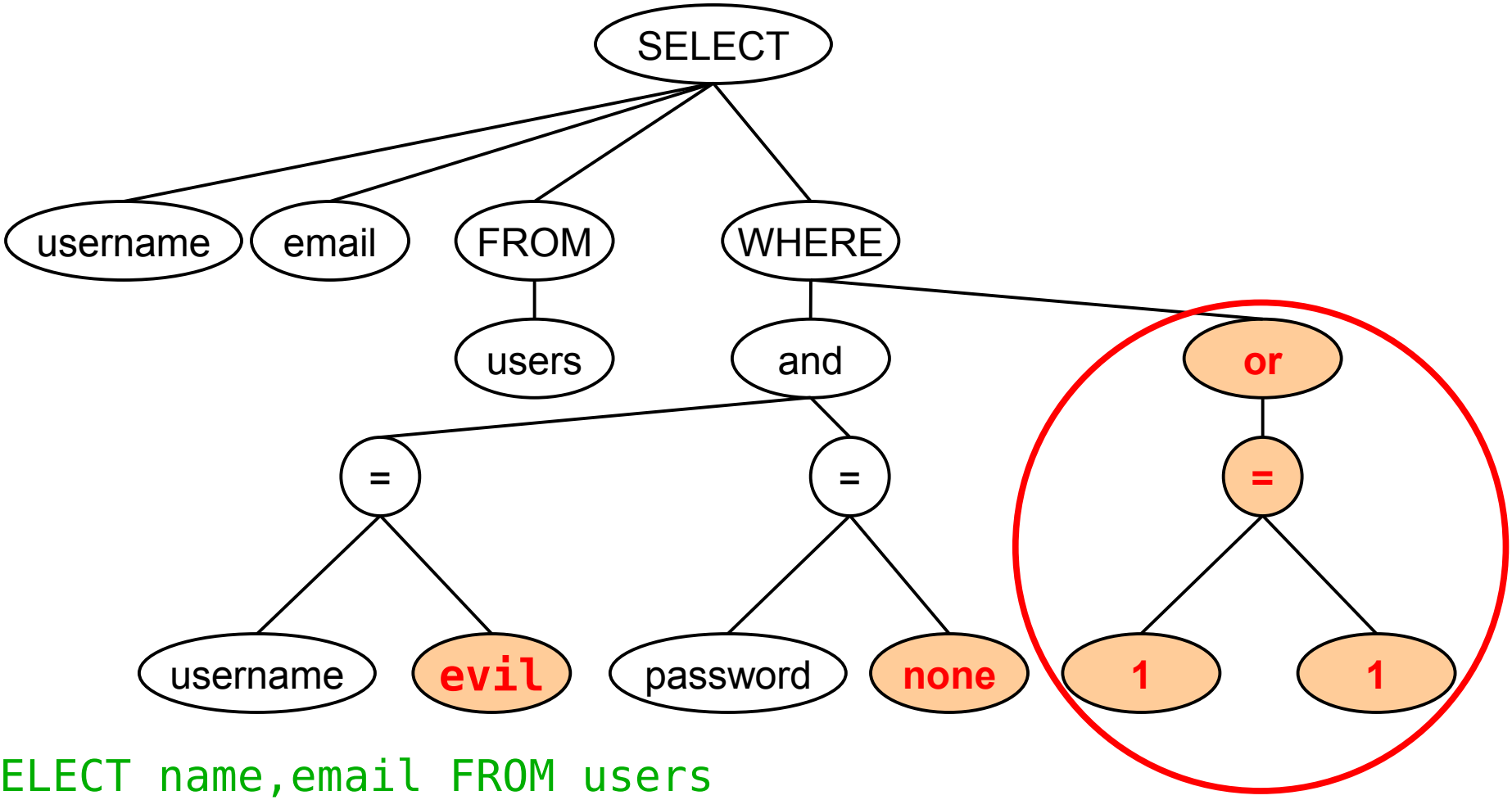


Grammatical analysis



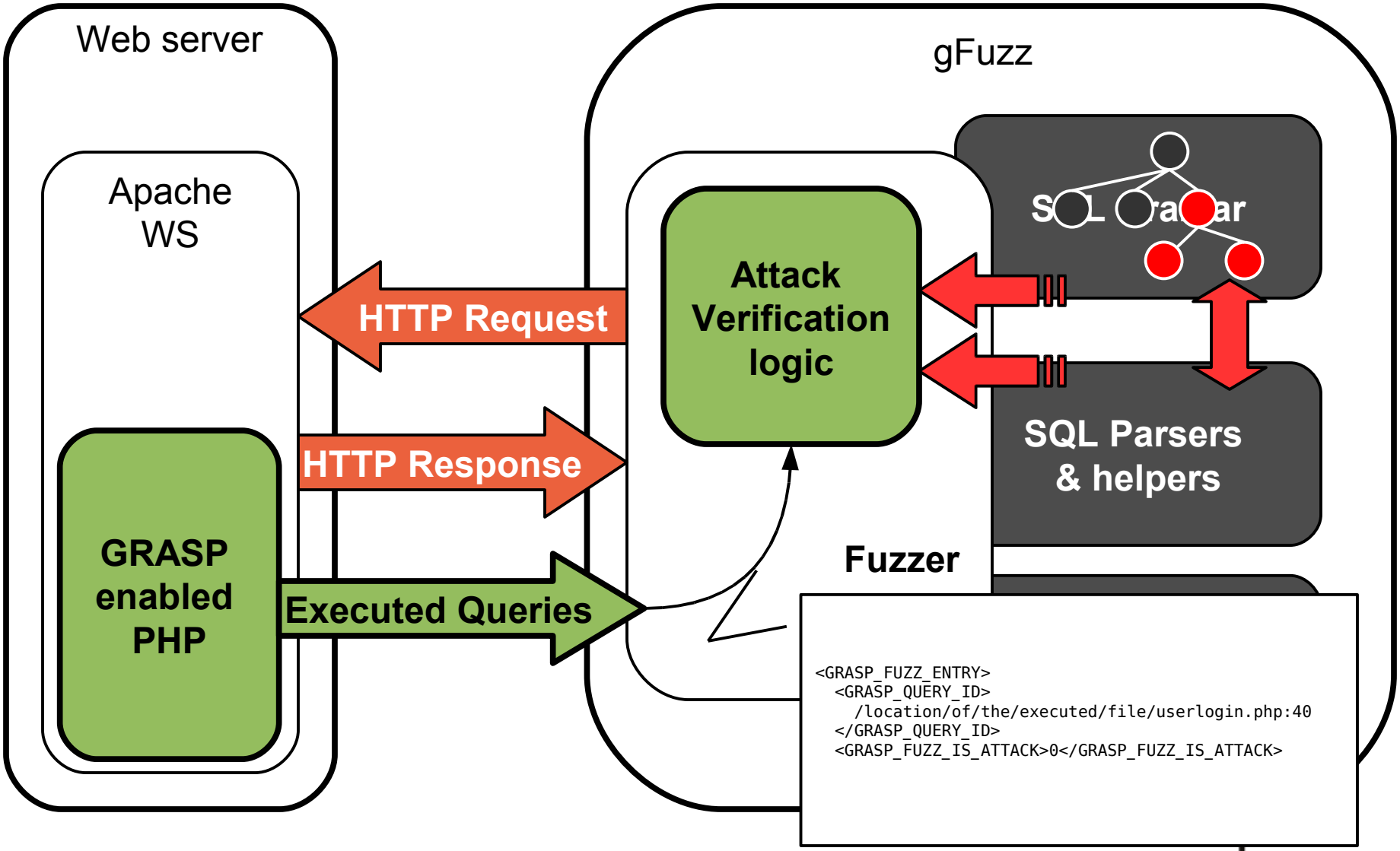

```
SELECT name,email FROM users
WHERE username='evil' and password='none' or 1=1;--'
```

Grammatical analysis + taint marks




```
SELECT name,email FROM users
WHERE username='evil' and password='none' or 1=1;--'
```

Altogether


Web server
Apache WS
GRASP enabled PHP
HTTP Request
HTTP Response
Executed Queries
gFuzz
Attack Verification logic
SQL Parsers & helpers
Fuzzer
<GRASP_FUZZ_ENTRY>
<GRASP_QUERY_ID>
/location/of/the/executed/file/userlogin.php:40
</GRASP_QUERY_ID>
<GRASP_FUZZ_IS_ATTACK>0</GRASP_FUZZ_IS_ATTACK>

Attack verification - witnesses

- The fuzzer sends “witness” requests
  - Not always possible
  - How to choose witness strings (heuristic):

```
SELECT *
FROM users
WHERE
  username = '12345'
AND
  password = '12345'
```

```
SELECT *
FROM users
WHERE
  username = 12345
AND
  password = 12345
```

```
SELECT *
FROM users
WHERE
  username = 'someString'
AND
  password = 'someString'
```

```
SELECT *
FROM users
WHERE
  username = someString
AND
  password = someString
```

Attack verification - witnesses

- The fuzzer sends “witness” requests
  - Web application logic is set appart:

```
<?php

    if ( isset($_POST[“concerned”]) &&
         isset($_POST[“indifferent”]) && isset($_POST[“dontknow”]) )
    {
        echo “you cannot be concerned, indifferent and
              don't know about it at the same time!”;
    }

?>
```

This is related to fuzz logic. But must be taken into account for witnesses

Attack verification - witnesses

Conclusion:

It is not always possible to submit a **witness** query.

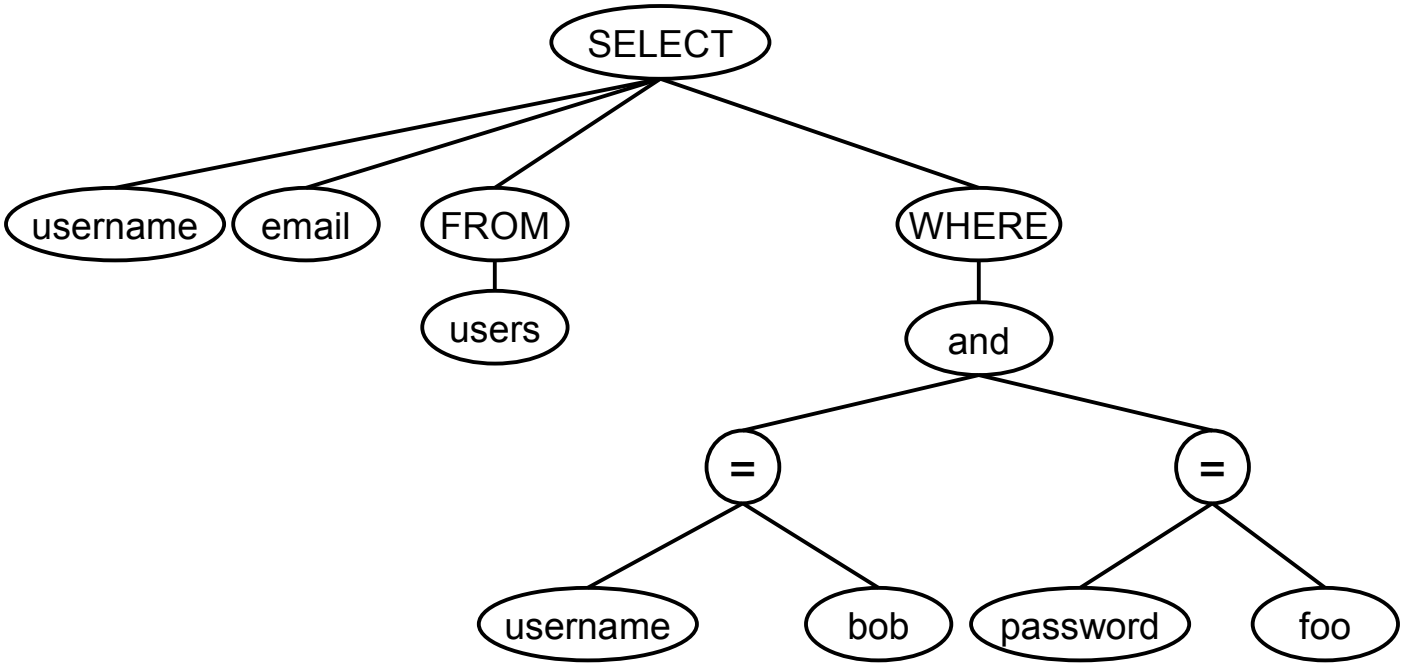
Classifying

For each query received

- If it had a witness, perform grammatical analysis to compare structural differences
- Otherwise, check if there's a terminal node with parent and brother fully controlled
- Report with instrumentation info

Query classification

- **Harmless**: Valid query and no terminal nodes are **fully** (brothers and parent) controlled by the attacker



Query classification

- **Warning**: The query is not grammar-compliant (and could not be analyzed):

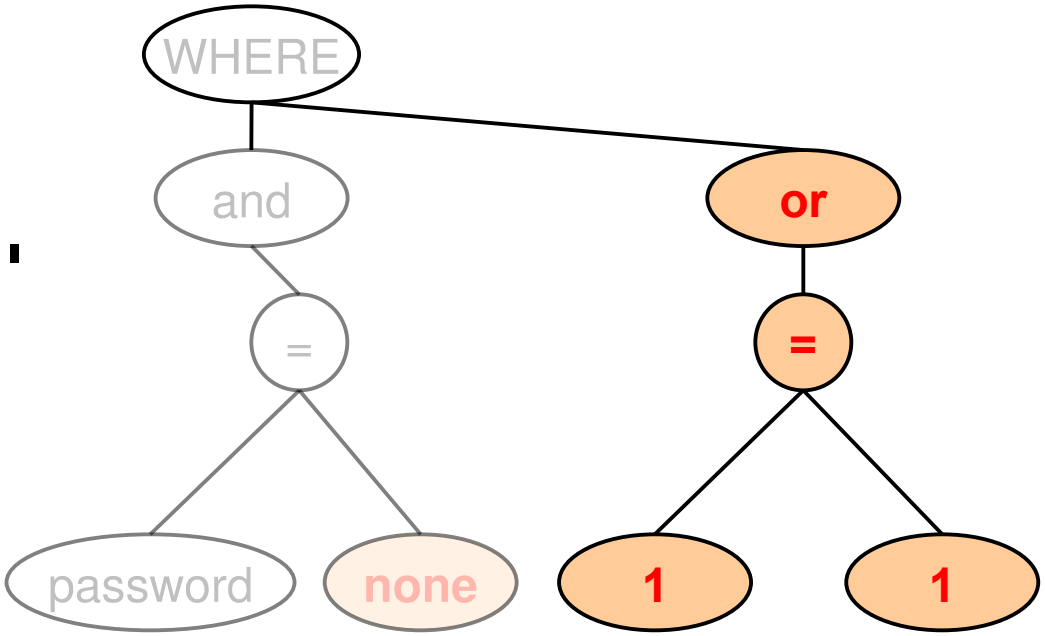
```
SELECT   name,email FROM users
WHERE    username='bob'
and      password=''
```

Could result in a successful attack or unexploitable error (this case IS exploitable)

Query classification

- **Successful Attack**: the attacker can control a terminal node, its brothers and its parent:

```
SELECT name,email
FROM users
WHERE username='bob'
and
  password='none'
or 1=1; --'
```



Agenda

- (quick!) Web application security overview
- SQL-injection attacks inside-out
- Fuzzing and gFuzz
- Detecting AnOwAliEs with gFuzz
- **Reporting**
- **Demo**
- Future work

Reporting

Gfuzz analysis

ATTACK	POST	select title from books where id = 'test-- or id= ''test--'	'test--

Grasp analysis & fuzz method

Executed query (controlled chars in red) with analysis info (background)

Fuzz string

```
<input type="text" name="id" size="40" />
```

Input / URL parameter

```
/7.php
<form action="" method="post">
<input type="text" name="id" size="40" />
<input type="text" name="name" size="40" />
<input type="text" name="surname" size="40" />
<input type="text" name="age" size="40" />
<input type="text" name="sex" size="40" />
<input type="text" name="religion" size="40" />
<input type="submit" value="Submit Query" />
</form>
```

Target

Fuzz vector

Demo

About the prototype

- The fuzzing logic is very simple, can be significantly improved
- SQL grammar is standard ANSI SQL-92 and only for selects. Can be extended
  (e.g., INSERT, UPDATE, nested SELECTS, ...)
- In Python, BSD license
- Any volunteers wishing to help?

Future

- Improve SQL support / attack detection
- Improve fuzzing engine
  - Create an audit module for w3af framework! (http://w3af.sourceforge.net)
- Add XSS detection
  - Bounded to GRASP support for XSS! (Any volunteer to help?)
- Improve run time!

Thanks!

Useful data

Corelabs research site:

http://corelabs.coresecurity.com

CORE Grasp for PHP (original version):

http://grasp.coresecurity.com

contact:

egutesman@coresecurity.com

Acknowledgments

- Pictures from
  - http://www.sxc.hu
  - http://www.openclipart.org
  - http://www.flickr.com (cc)

- People who helped
  - **Sebastián Cufre**
  - Ariel Waissbein
  - Pedro Varangot
  - Fernando Russ
  - Aureliano Calvo