A Principled Approach to Injection-Attack Detection

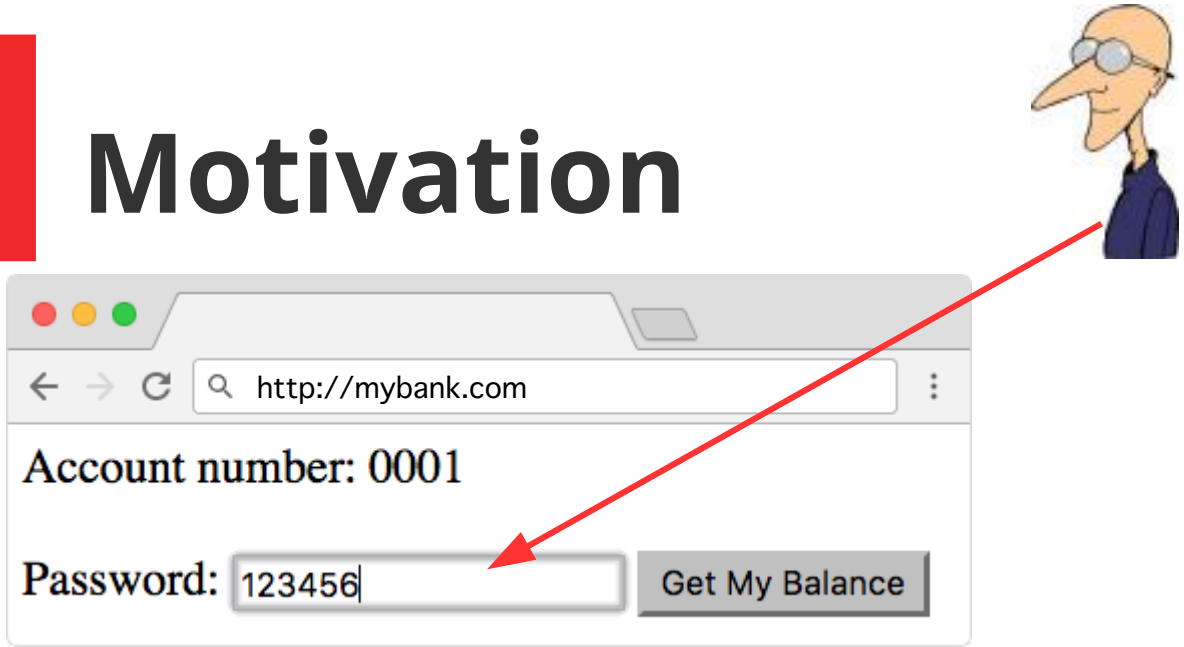
Donald Ray Jay Ligatti
Cagri Cetin

University of South Florida
Dept of Computer Science and Engineering

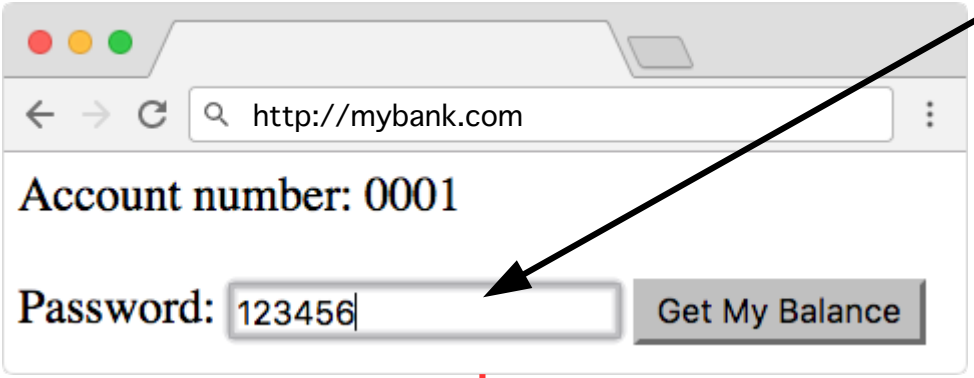
Ideas appeared in:

- Donald Ray and Jay Ligatti. Defining injection attacks. *International Conference on Information Security (**ISC**)*, 2014.
- Donald Ray and Jay Ligatti. Defining code-injection attacks. *Symposium on Principles of Programming Languages (**POPL**)*, 2012.

Motivation


http://mybank.com
Account number: 0001
Password: 123456
Get My Balance

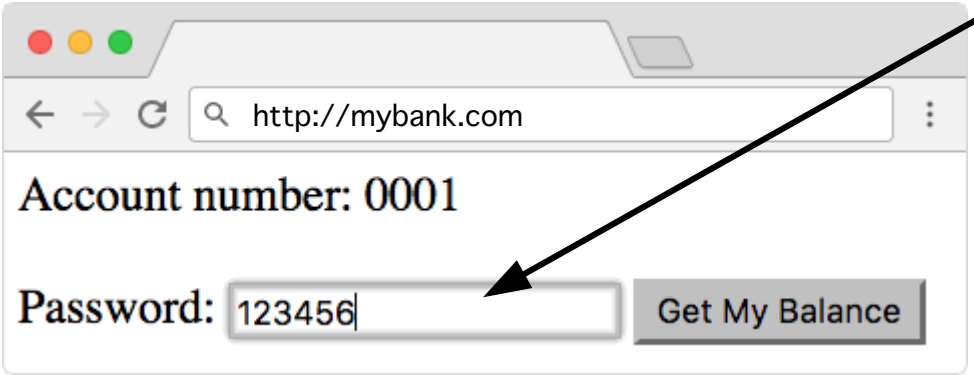
Motivation



Web Application

```
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
      password=' + input() + '

  result = executeQuery(sql)

      showResult(result)
```

Motivation

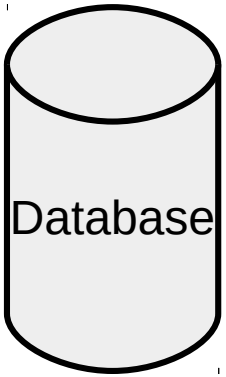


Web Application

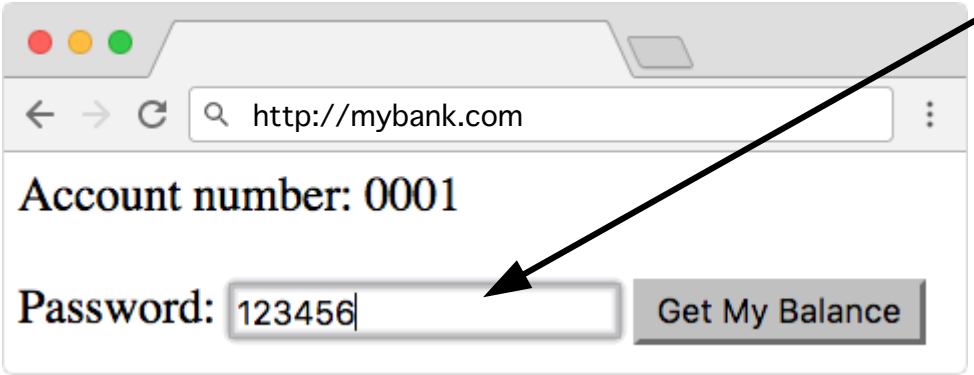
```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
    password=' + input() + '

result = executeQuery(sql)

showResult(result)
```

Output Program

```
SELECT balance
from accts WHERE
num=0001 AND
password='123456'
```



Motivation

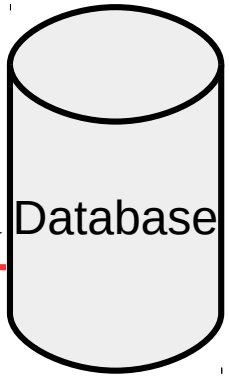


Web Application

```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
    password=' + input() + '

  result = executeQuery(sql)

      showResult(result)
```

Output Program

```
SELECT balance
from accts WHERE
num=0001 AND
password='123456'
```



Account Number	Balance
0001	10$

Motivation

http://mybank.com

Account number: 0001

Password: 123456 Get My Balance

http://mybank.com

Account Number	Balance
0001	10$

Web Application

```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
    password=' + input() + '

  result = executeQuery(sql)

      showResult(result)
```

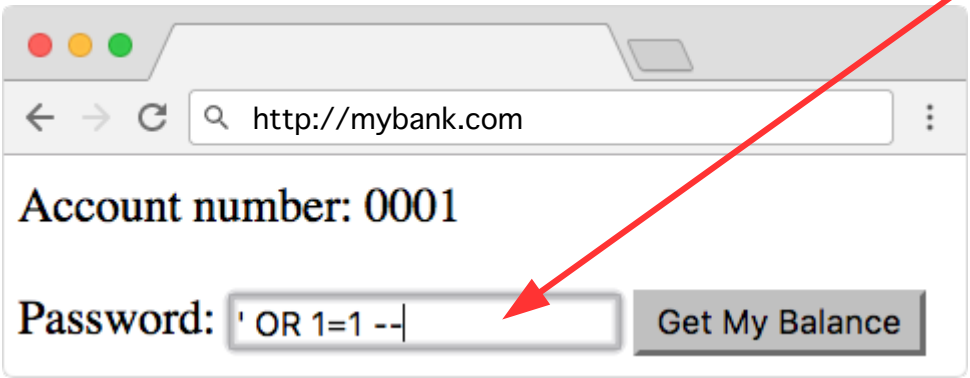
Output Program

```
SELECT balance
from accts WHERE
num=0001 AND
password='123456'
```

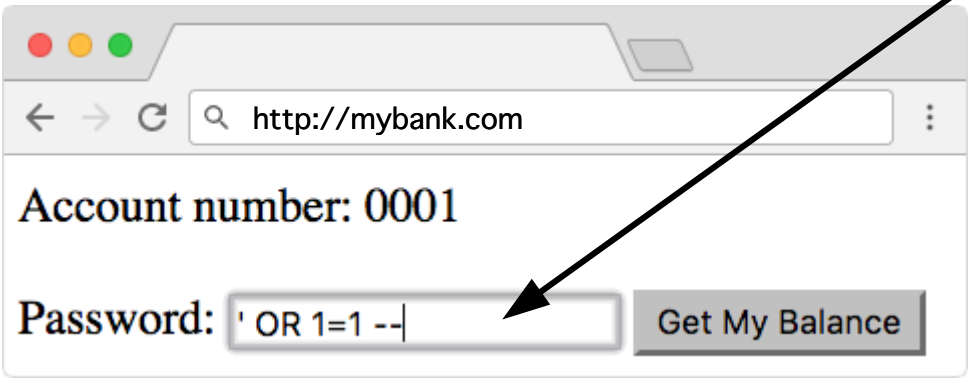
Database

Account Number	Balance
0001	10$

Motivation


http://mybank.com
Account number: 0001
Password:
' OR 1=1 --
Get My Balance

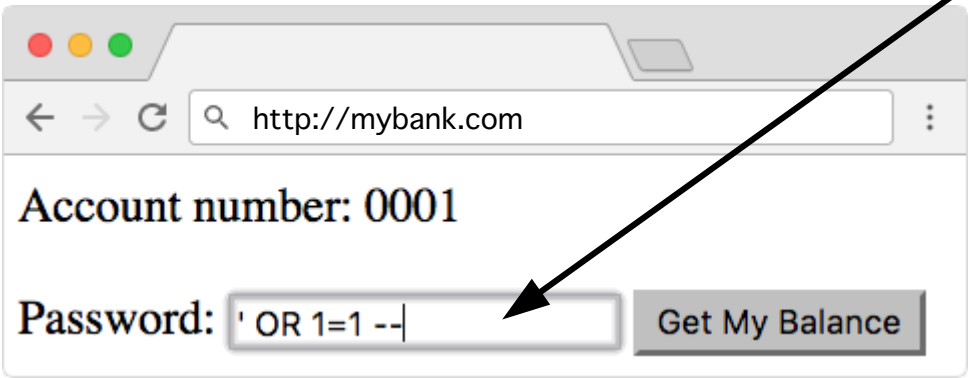
Motivation



Web Application

```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
    password=' + input() + '

   result = executeQuery(sql)

       showResult(result)
```

Motivation

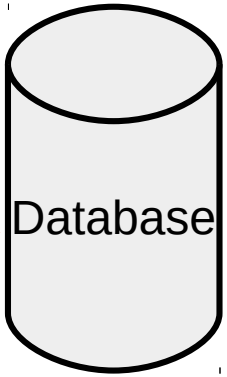


Web Application

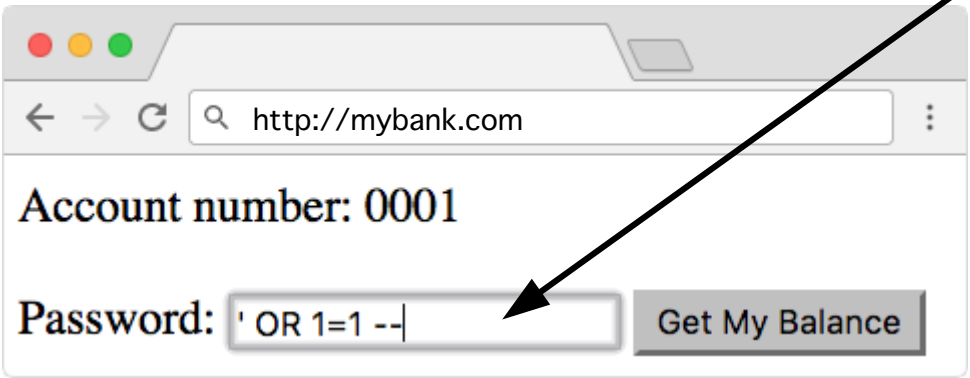
```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
   password=' + input() + '

result = executeQuery(sql)

showResult(result)
```

Output Program

```
SELECT balance from
accts WHERE
num=0001 AND
password='' OR 1=1 --'
```



Motivation

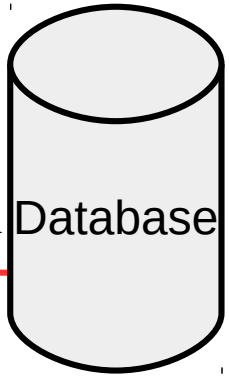


Web Application

```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
   password=' + input() + '

   result = executeQuery(sql)

      showResult(result)
```

Output Program

```
SELECT balance from
   accts WHERE
   num=0001 AND
password='' OR 1=1 --'
```



Account Number	Balance
0001	10$
0002	15$
0003	5$
0004	1000$
0005	100$

Motivation

http://mybank.com

Account number: 0001

Password: ' OR 1=1 --

Get My Balance

http://mybank.com

Account Number	Balance
0001	10$
0002	15$
0003	5$
0004	1000$
0005	100$

Web Application

```
.
.
.
sql = SELECT balance from accts
      WHERE num=0001 AND
   password=' + input() + '

  result = executeQuery(sql)

showResult(result)
```

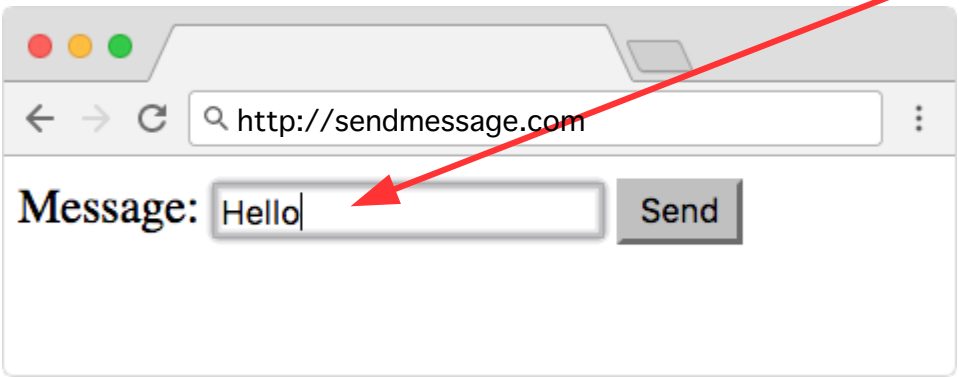
Output Program

```
SELECT balance from
    accts WHERE
    num=0001 AND
password='' OR 1=1 --'
```

Database

Account Number	Balance
0001	10$
0002	15$
0003	5$
0004	1000$
0005	100$

Motivation



Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

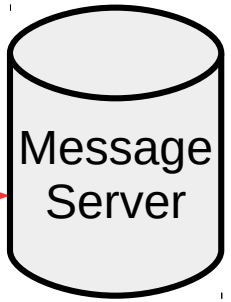
Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

Output Program

```
$data = 'Hello';
securityCheck();
$data .= '&f=exit#';
f()
```



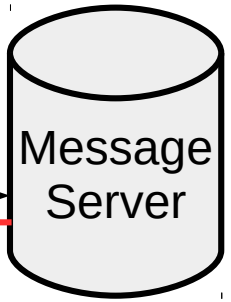
Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

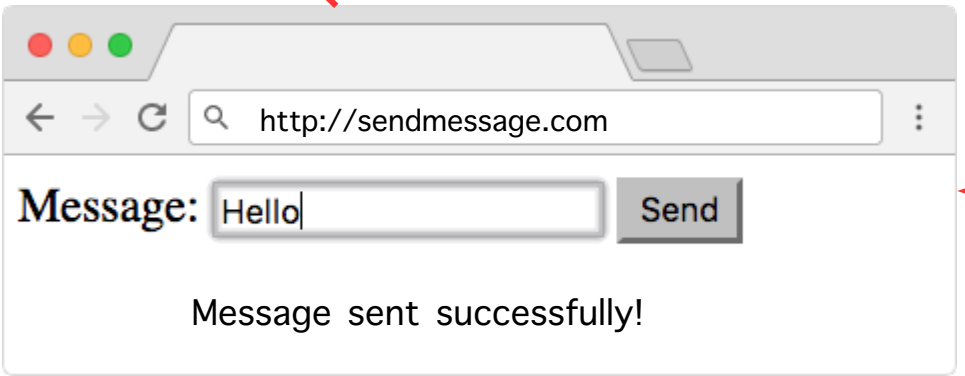
Output Program

```
$data = 'Hello';
securityCheck();
$data .= '&f=exit#';
f()
```



Message Sent

Motivation



Web Application

```
.
.
.
code = $data =' + input() + '
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

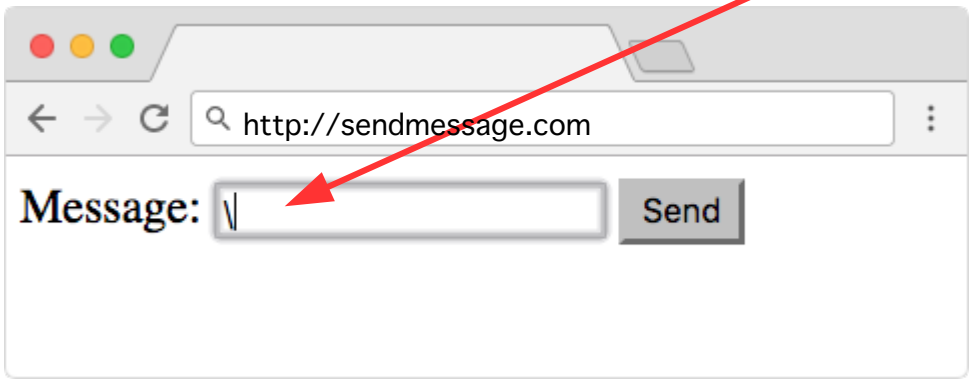
Output Program

```
$data = 'Hello';
securityCheck();
$data .= '&f=exit#';
f()
```

Message Server

Message Sent

Motivation



Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

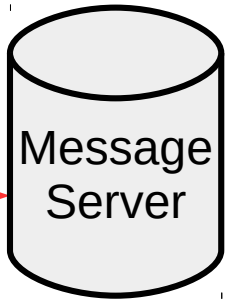
Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

Output Program

```
$data = '\';
securityCheck();
$data .= '&f=exit#';
f()
```



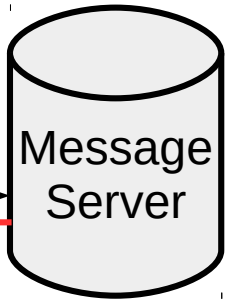
Motivation

Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

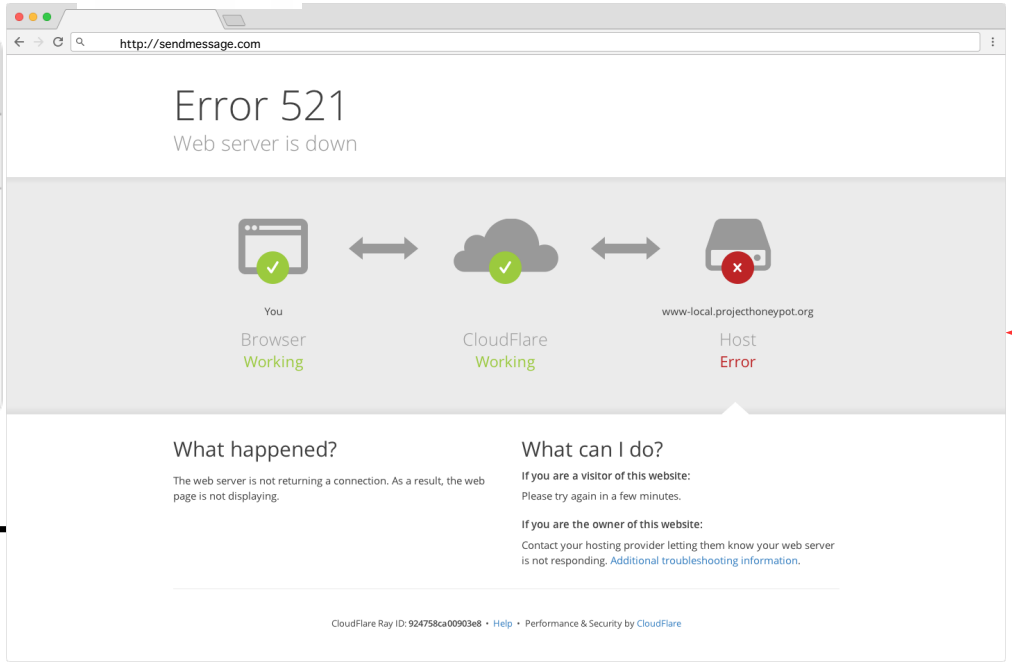
Output Program

```
$data = '\';
securityCheck();
$data .= '&f=exit#';
f()
```



Server is not responding

Motivation



Web Application

```
.
.
.
code = $data =' + input() + ';
securityCheck(); $data .= '&f=exit#';
f()

result = sendMessage(code)

showResult(result)
```

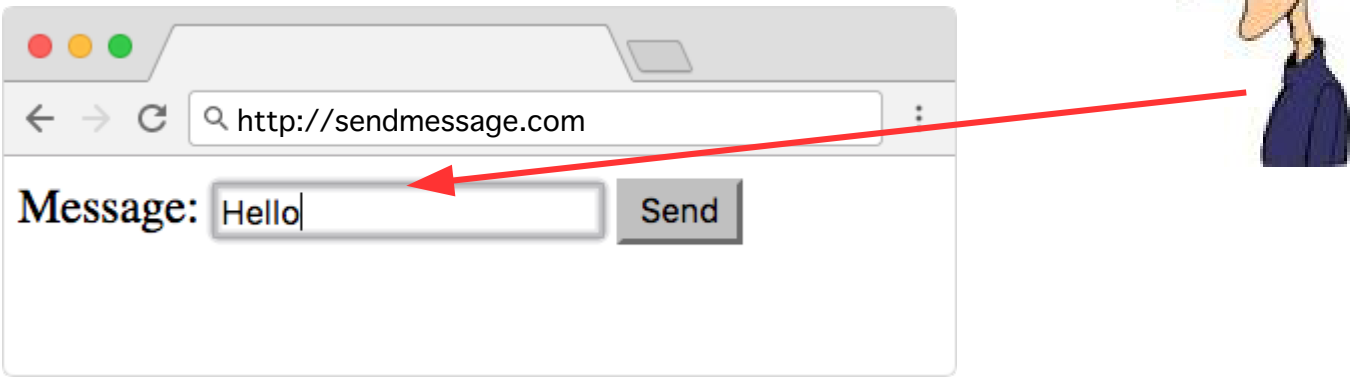
Output Program

```
$data = '\';
securityCheck();
$data .= '&f=exit#';
f()
```

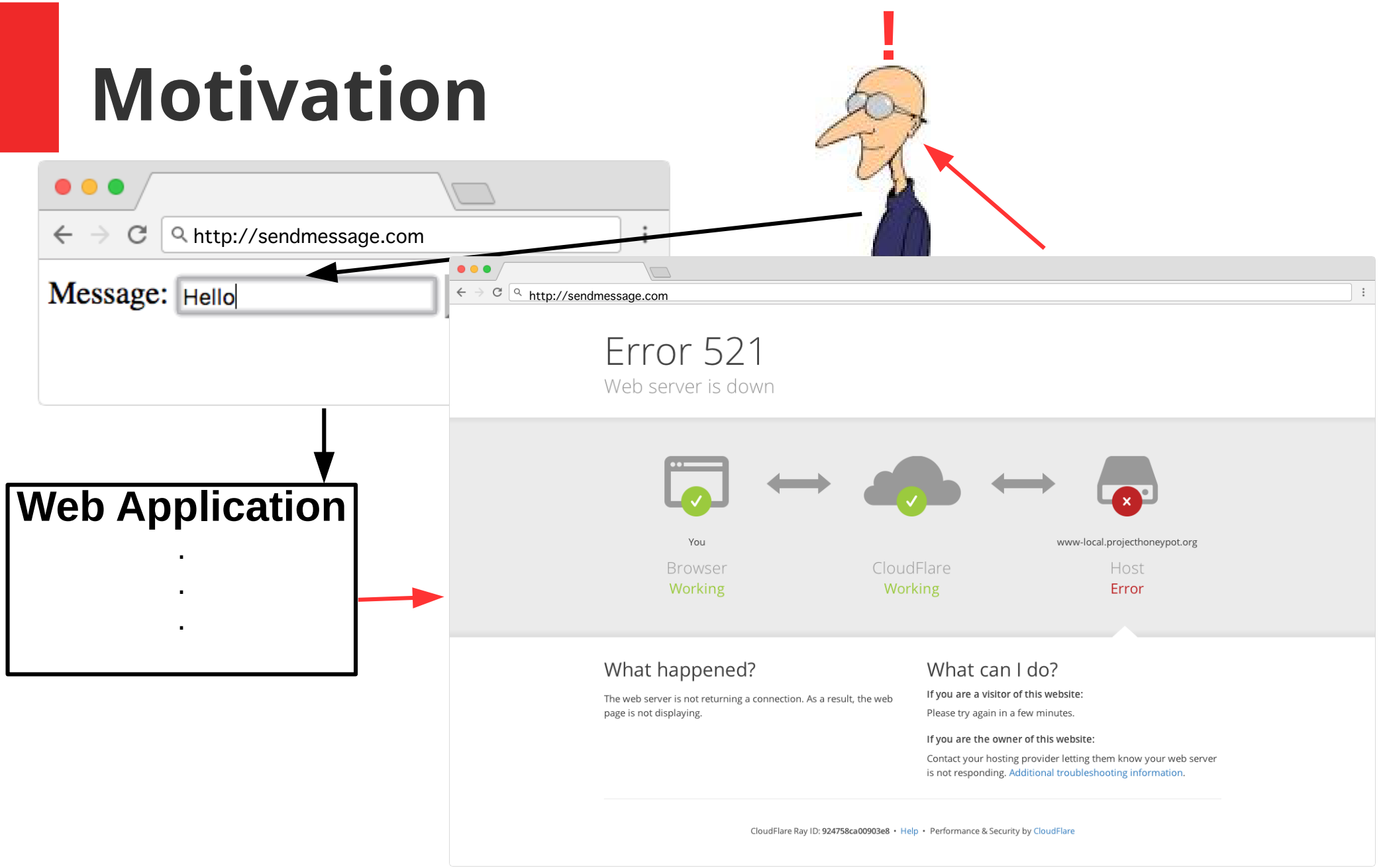
Message Server

Server is not responding

Motivation



Motivation


http://sendmessage.com
Message: Hello
Web Application
.
.
.
http://sendmessage.com
Error 521
Web server is down
You
Browser
Working
CloudFlare
Working
www-local.projecthoneypot.org
Host
Error
What happened?
The web server is not returning a connection. As a result, the web page is not displaying.
What can I do?
If you are a visitor of this website:
Please try again in a few minutes.
If you are the owner of this website:
Contact your hosting provider letting them know your web server is not responding. Additional troubleshooting information.
CloudFlare Ray ID: 924758ca00903e8 • Help • Performance & Security by CloudFlare

Motivation

2013 OWASP Top 10 Most Critical Web Application Security Risks

1. **Injection**
2. Broken Authentication and Session
3. **Cross-site Scripting**
4. Insecure Direct Object References
5. Security Misconfiguration

https://www.owasp.org/index.php/Top_10_2013-Top_10

2011 MITRE CWE/SANS Top 8 Most Dangerous Software Errors

1. **SQL Injection**
2. **OS Cmd Injection**
3. Buffer Overflow
4. **Cross-site Scripting**
5. No Authentication

https://cwe.mitre.org/top25/

Outline

- Motivation
- ➢ **Related Work**
- Defining Injection Attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- An Algorithm for Detecting and Preventing BroNIEs
- Conclusion

Related Work: Academic

- Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: Proceedings of the Symposium on Principles of Programming Languages (POPL). (2006) 372–382
- Bisht, P., Madhusudan, P., Venkatakrishnan, V.N.: CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. Transactions on Information and System Security (TISSEC) 13(2) (2010) 1–39
- Halfond, W., Orso, A., Manolios, P.: Wasp: Protecting web applications using positive tainting and syntax-aware evaluation. Transactions on Software Engineering (TSE) 34(1) (2008) 65–81
- Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically hardening web applications using precise tainting. In: Proceedings of the International Information Security Conference (SEC). (2005) 372–382
- Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In: Proceedings of the USENIX Security Symposium. (2006) 121–136

All suffer from **false positives** and **false negatives** [POPL'12]

Related Work: Parameterized Queries

- Applications output templates that have **placeholders** for where the untrusted inputs should be used.

```
SELECT balance from accts WHERE num=0001 AND
        password=[          ].string
```

Related Work: Parameterized Queries

- Applications output templates that have **placeholders** for where the untrusted inputs should be used.

```
SELECT balance from accts WHERE num=0001 AND
          password=[          ].string
```

```
SELECT balance from accts WHERE num=0001 AND
          password=[ '\'OR 1=1--' ].string
```

Related Work: Parameterized Queries

Problems:

- Requires significant, manual rewrites in the application
- Not mandatory to use
- Not implemented in many output languages
- Programmers are not using enough

Outline

- Motivation
- Related Work
- **Defining Injection Attacks**
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- An Algorithm for Detecting and Preventing BroNIEs
- Conclusion

Defining Injection

A symbol has been **injected** iff it propagates unmodified from an **untrusted input** into the **output program**.

Defining Injection

A symbol has been **injected** iff it propagates unmodified from an **untrusted input** into the **output program**.

123456

Application

.

.

output(SELECT balance from accts WHERE num=0001 AND password=' + **input()** + ')

.

Output Program

SELECT balance from accts WHERE num=0001 AND password='**123456**'

Defining (Non)Code

Literals are **noncode** symbols in output programs*

- ‘Hello world’
- true
- false
- 10
- 3.14
- Jan 1 2005 1:29PM
- 1998/11/23
- 6.4E10

* The formal definition is more complex. For details see ISC and POPL papers.

Defining NIEs

NIE (**N**oncode **I**nsertion or **E**xpansion) Property:

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Defining NIEs

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
SELECT balance from accts WHERE num=001
        password='123456'
```

Template program:

```
SELECT balance from accts WHERE num=0001 AND
        password=''
```

Defining Injection Attacks

A BroNIE (Broken NIE) occurs exactly when the output program **does not satisfy** the NIE property.

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
SELECT balance from accts WHERE num=001 AND
password='' OR 1=1 --'
```

Template program:

```
SELECT balance from accts WHERE num=001 AND
password=''
```

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
SELECT balance from accts WHERE num=001 AND
        password='' OR 1=1 --'
```

code code

Template program:

```
SELECT balance from accts WHERE num=001 AND
        password=''
```

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

comment

Output program:

```
INSERT INTO users VALUES ('evilUser', TRUE)--', FALSE)
```

Template program:

```
INSERT INTO users VALUES ('                    ', FALSE)
```

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
$data='\'; securityCheck(); $data .='&f=exit#'; \n f();
```

Template program:

```
$data=' '; securityCheck(); $data .='&f=exit#'; \n f();
```

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
$data='\'; securityCheck(); $data .='&f=exit#'; \n f();
```

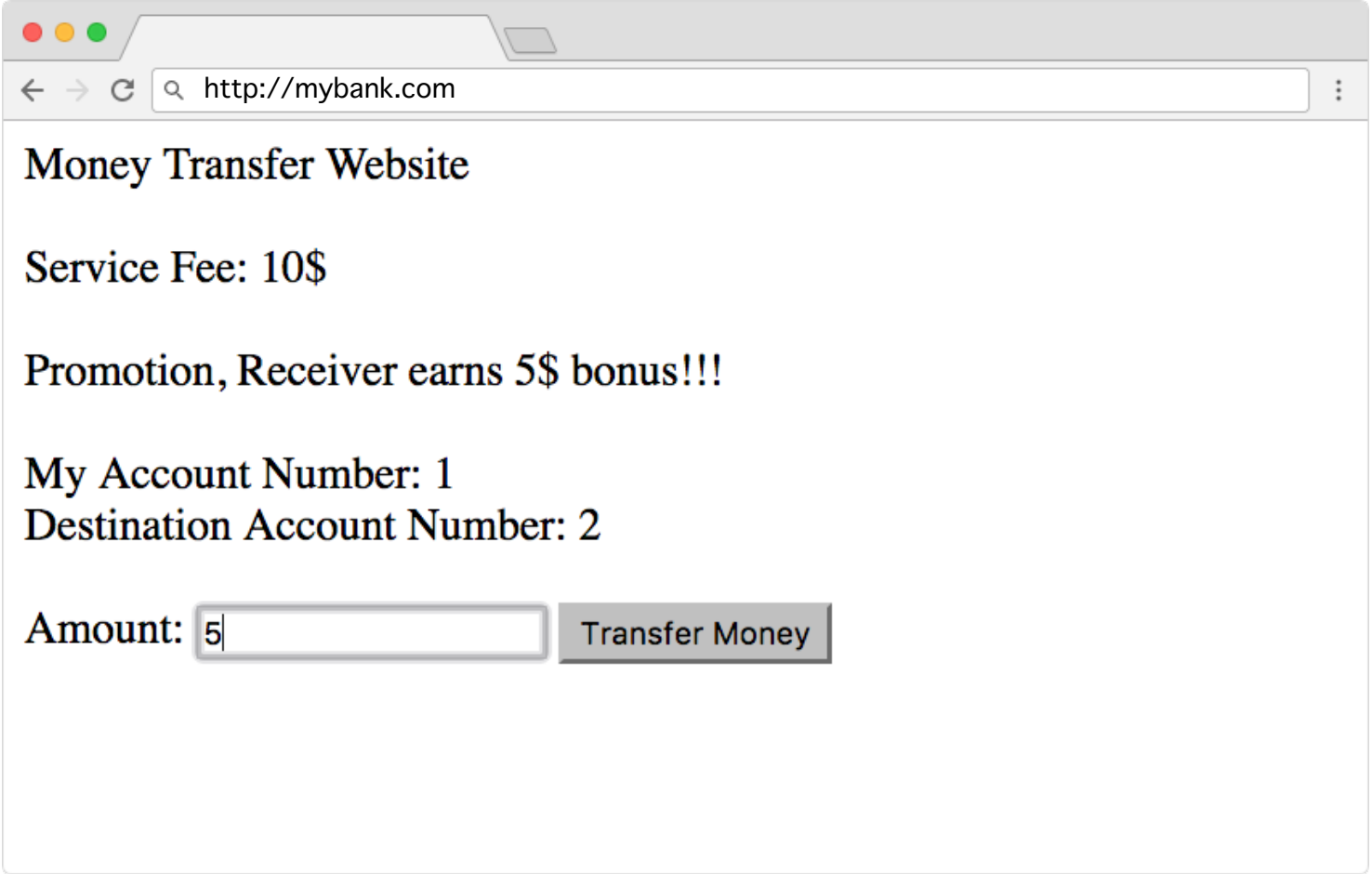
string literal

Template program:

```
$data=' '; securityCheck(); $data .='&f=exit#'; \n f();
```

code code

Examples


http://mybank.com
Money Transfer Website
Service Fee: 10$
Promotion, Receiver earns 5$ bonus!!!
My Account Number: 1
Destination Account Number: 2
Amount: 5
Transfer Money

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
INSERT INTO trans VALUES (1,-5E-10)
INSERT INTO trans VALUES (2, 5E+5)
```

Template program:

```
INSERT INTO trans VALUES (□,-□-10)

INSERT INTO trans VALUES (□,□+5)
```

Examples

An output program satisfies the **NIE property** iff the **injected symbols** only **insert or expand** noncode (i.e., literals).

Output program:

```
INSERT INTO trans VALUES (1,-5E-10)
INSERT INTO trans VALUES (2, 5E+5)
```

float

float

Template program:

```
INSERT INTO trans VALUES (  ,-  -10)

INSERT INTO trans VALUES (  ,  +5)
```

code

code

Outline

- Motivation
- Related Work
- Defining Injection Attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- ➢ **An Algorithm for Detecting and Preventing BroNIEs**
- Conclusion

An Algorithm for BroNIE Detection and Prevention

BroNIEs can be **precisely** and **automatically** prevented by:

An Algorithm for BroNIE Detection and Prevention

BroNIEs can be **precisely** and **automatically** prevented by:

1. Finding **injected symbols** in an **output program** (e.g., with a taint-tracking mechanism),

An Algorithm for BroNIE Detection and Prevention

BroNIEs can be **precisely** and **automatically** prevented by:

1. Finding **injected symbols** in an **output program** (e.g., with a taint-tracking mechanism),

2. Detecting whether the **output program** satisfies the **NIE property** (by comparing it with its template), and

An Algorithm for BroNIE Detection and Prevention

BroNIEs can be **precisely** and **automatically** prevented by:

1. Finding **injected symbols** in an **output program** (e.g., with a taint-tracking mechanism),

2. Detecting whether the **output program** satisfies the **NIE property** (by comparing it with its template), and

3. Executing the output program iff it satisfies the NIE property**.**

Outline

- Motivation
- Related Work
- Defining Injection Attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- An Algorithm for Detecting and Preventing BroNIEs
- **Conclusion**

Conclusion

- A new kind of attack—***noncode*** injection attack—has been demonstrated and defined precisely

- A new technique for detecting and preventing ***BroNIEs***—including code and noncode injection attacks—has been provided

Thanks

Questions?

cagricetin@mail.usf.edu

Papers are available at
http://www.cse.usf.edu/~ligatti/projects/ciao/

Related Work: Parameterized Queries

Regular query:

```
String sql = “SELECT balance from accts WHERE password=” + input();
```

Parameterized query:

```
String sql = “SELECT balance from accts WHERE password= ?”;
PreparedStatement prepStmt = conn.prepareStatement(sql);
prepStmt.setString(1, input());
ResultSet rs = prepStmt.executeQuery();
```

Related Work: SqlCheck

Program **does not** exhibit an attack

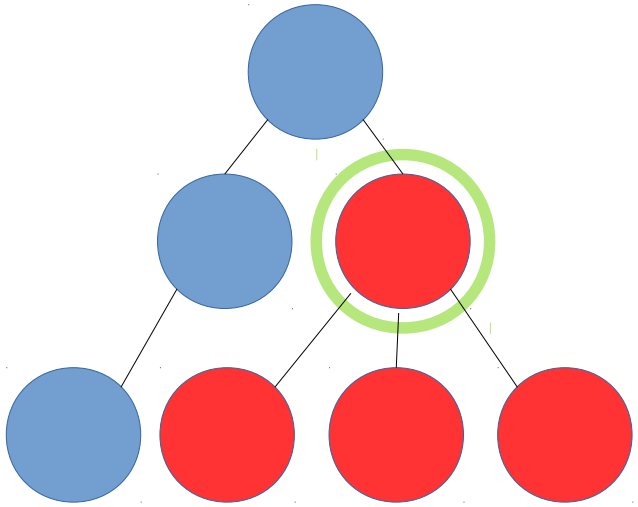
There is a node in the program's parse tree that is **entirely injected** and that contains **all injected symbols**

Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: Proceedings of the Symposium on Principles of Programming Languages (POPL). (2006) 372–382

Related Work: SqlCheck

Program **does not** exhibit an attack

There is a node in the program's parse tree that is **entirely injected** and that contains **all injected symbols**

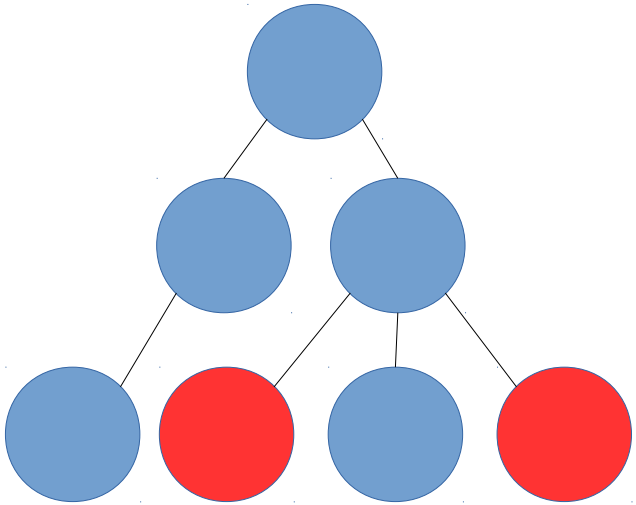


Not an attack

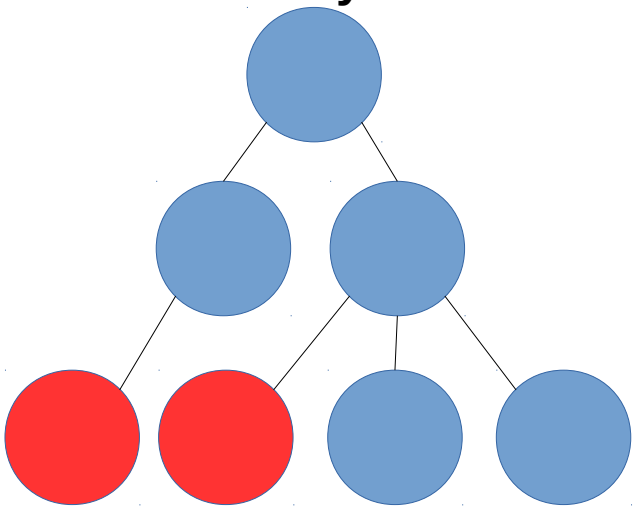
Related Work: SqlCheck

Program **does not** exhibit an attack

There is a node in the program's parse tree that is **entirely injected** and that contains **all injected symbols**



Attack



Attack

Related Work: SqlCheck

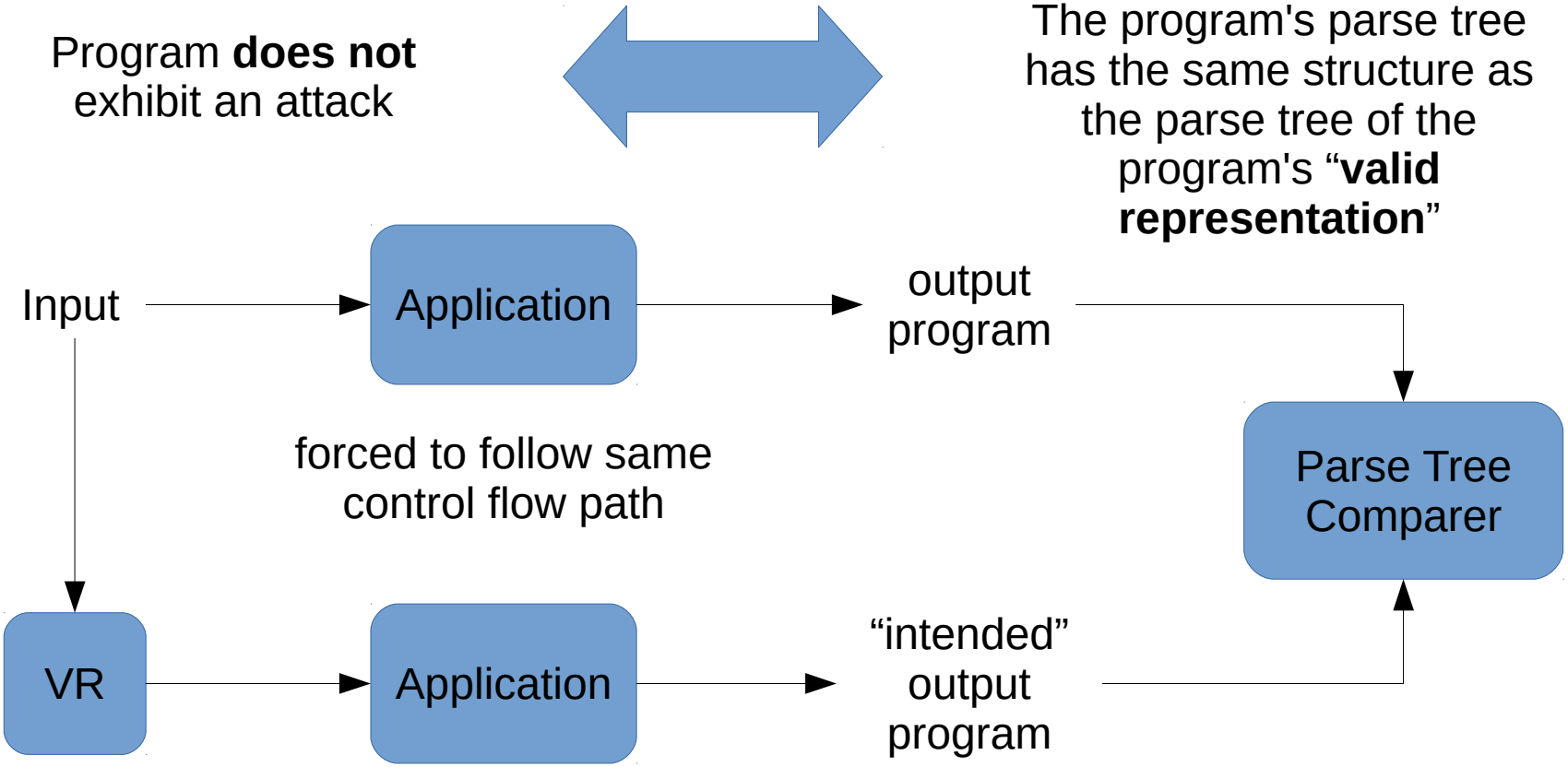
Program **does not** exhibit an attack

There is a node in the program's parse tree that is **entirely injected** and that contains **all injected symbols**

False Positive: `SELECT * FROM table WHERE 'filename.extension'`

False Negative: `SELECT * FROM table WHERE pin=exit()`

Related Work: CANDID



Bisht, P., Madhusudan, P., Venkatakrishnan, V.N.: CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. Transactions on Information and System Security (TISSEC) 13(2) (2010) 1–39

Related Work: CANDID

Program **does not** exhibit an attack

The program's parse tree has the same structure as the parse tree of the program's "**valid representation**"

False Positive: `SELECT * FROM table WHERE false`

Valid Representation: `SELECT * FROM table WHERE aaaaa`

False Negative: `SELECT * FROM table WHERE pin=exit()`

Valid Representation: `SELECT * FROM table WHERE pin=aaaa()`

Bisht, P., Madhusudan, P., Venkatakrishnan, V.N.: CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. Transactions on Information and System Security (TISSEC) 13(2) (2010) 1–39

Background: Program Tokenizing

```
SELECT * FROM orders WHERE username='cagri'
```

Background: Program Tokenizing

```
SELECT * FROM orders WHERE username='cagri'
```

SELECT	*	FROM	ORDERS	WHERE	username	=	‘cagri’

Taint-Tracking Mechanisms

- Halfond, W., Orso, A., Manolios, P.: Wasp: Protecting web applications using positive tainting and syntax-aware evaluation. Transactions on Software Engineering (TSE) 34(1) (2008) 65–81
- Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically hardening web applications using precise tainting. In: Proceedings of the International Information Security Conference (SEC). (2005) 372–382
- Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In: Proceedings of the USENIX Security Symposium. (2006) 121–136
- Pietraszek, T., Berghe, C.V.: Defending against injection attacks through context- sensitive string evaluation. In: Proceedings of Recent Advances in Intrusion Detection (RAID). (2005) 124–145
- Son, S., McKinley, K.S., Shmatikov, V.: Diglossia: detecting code injection attacks with precision and efficiency. In: Proceedings of the Conference on Computer and Communications Security (CCS). (2013) 1181–1192
- Dalton, M., Kannan, H., Kozyrakis, C.: Raksha: A flexible information flow architecture for software security. In: Proceedings of the International Symposium on Computer Architecture (ISCA). (2007) 482–493
- Clause, J., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA). (2007) 196–206

Defining (Non)Code

- **Free variables** specify dynamic substitution operations, so values must be closed (i.e., contain no free variables) to be considered dynamically passive

- In languages were whitespace is **significant** (e.g., Python), indenting whitepsace cannot be considered lexically removed and is thus dynamically active.

Defining (Non)Code

Noncode symbols in output programs are those that are **dynamically passive**:

- Values

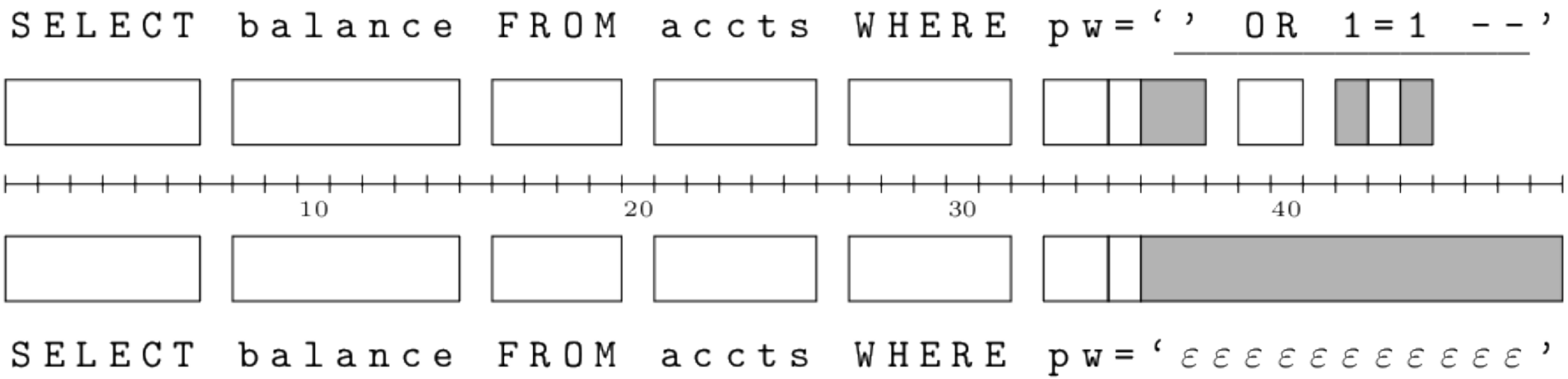
```
12000          [1,1,2,3,5,8,13]            'Hello world!'

(“John Doe”,false,25)              6.02E24           true

false       3.14       [(“orange”,0.25),(“apple”,0.20)]
```

- Lexically-removed symbols

```
/* typically, whitespace and comment symbols  */
```

Examples

An output program satisfies the **NIE property** if and only if the **injected symbols** only **insert or expand** noncode.



Legend:

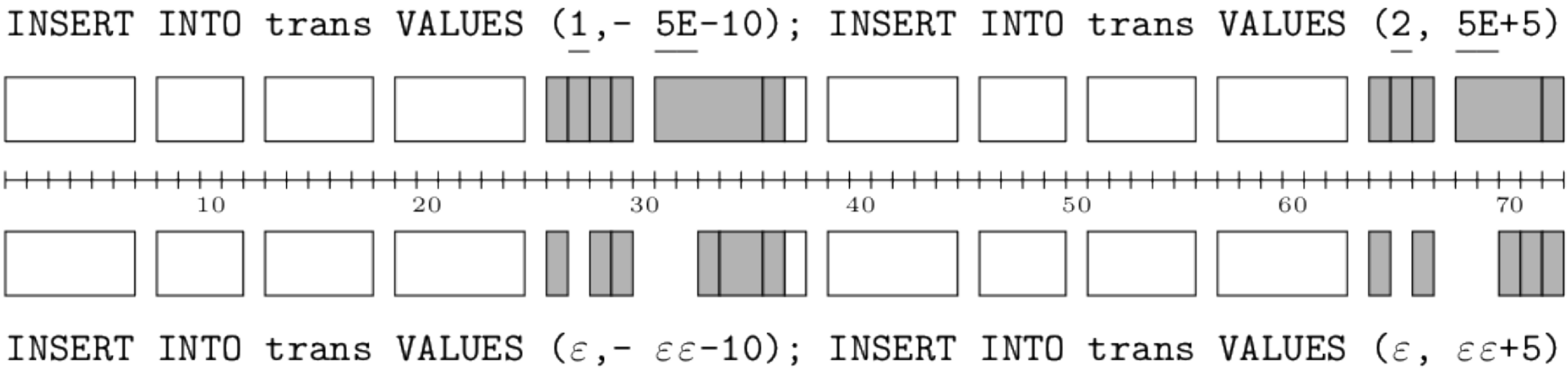
Injected symbols

Code tokens

Noncode tokens

Examples

Removing all **injected symbols** from an output program should **only delete** or **contract noncode tokens**.



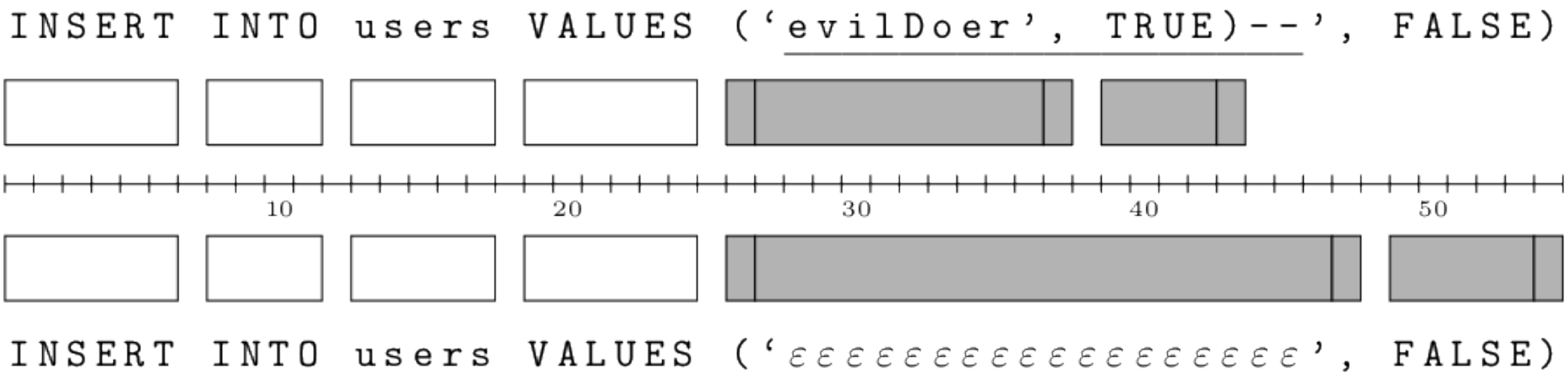
Legend:
Injected symbols

Code tokens

Noncode tokens

Examples

An output program satisfies the **NIE property** if and only if the **injected symbols** only **insert or expand** noncode.



Legend:

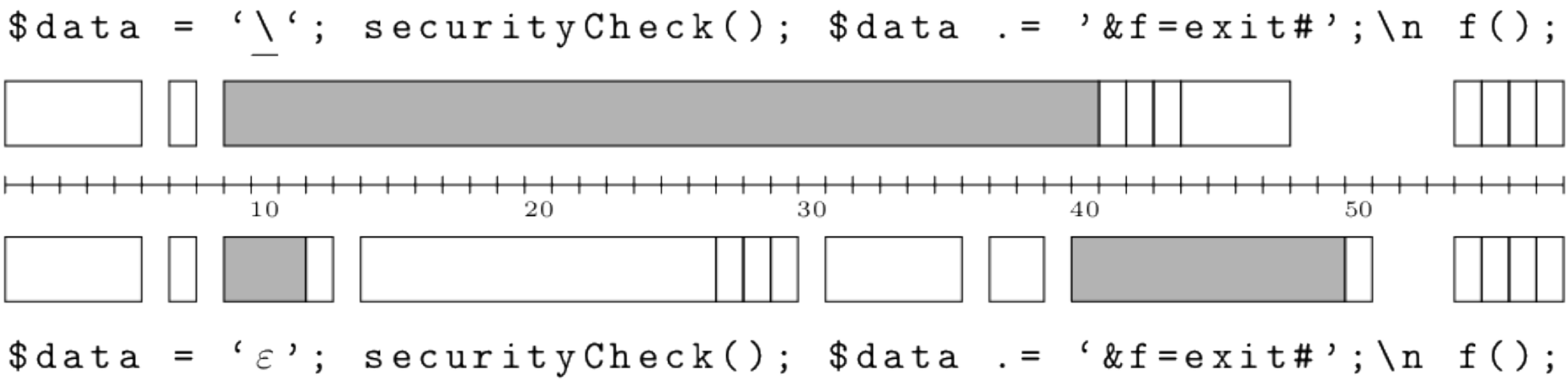
Injected symbols

Code tokens

Noncode tokens

Examples

An output program satisfies the **NIE property** if and only if the **injected symbols** only **insert or expand** noncode.



Legend:
Injected symbols

Code tokens

Noncode tokens

Definitions

Definition 1 ([4]). *For all alphabets* Σ, *the* tainted-symbol alphabet $\underline{\Sigma}$ *is* $\{\sigma \mid \sigma \in \Sigma \vee (\exists \sigma' \in \Sigma : \sigma = \underline{\sigma'})\}$.

Next, language L is augmented to allow programs to contain tainted symbols.

Definition 2 ([4]). *For all languages* L *with alphabet* Σ, *the* tainted output language $\underline{L}$ *with alphabet* $\underline{\Sigma}$ *is* $\{\sigma_1..\sigma_n \mid \exists \sigma'_1..\sigma'_n \in L\colon \forall i \in \{1..n\}\colon (\sigma_i = \sigma'_i \vee \sigma_i = \underline{\sigma'_i})\}$.

Finally, an output-program symbol is injected if and only if it is tainted.

Definition 3 ([4]). *For all alphabets* Σ *and symbols* $\sigma \in \underline{\Sigma}$, *the predicate* $injected(\sigma)$ *is true iff* $\sigma \notin \Sigma$.

Definitions

Definition 4. *For all L-programs $p = \sigma_1..\sigma_n$ and position numbers $i \in \{1..|p|\}$, predicate $Noncode(p, i)$ holds iff $TR_L(p, i)$ or there exist low and high symbol-position numbers $l \in \{1..i\}$, $h \in \{i..|p|\}$ such that $\sigma_l..\sigma_h$ is a closed value in p.*

Definition 5 ([4]). *A* CIAO *occurs exactly when a taint-tracking application outputs $\underline{L}$-program $p = \sigma_1..\sigma_n$ such that $\exists i \in \{1..n\} : (injected(\sigma_i) \ \wedge \ Code(p, i))$.*

Definition 6. *The* template *of a program p, denoted $[\varepsilon/\underline{\sigma}]p$, is obtained by replacing each injected symbol in p with an ε.*

Definitions

Definition 7. *A token* $t = \tau_i(v)_j$ *can be* expanded *into token* $t' = \tau'_{i'}(v')_{j'}$, *denoted* $t \preceq t'$, *iff:*

- $\tau = \tau'$
- $i' \leq i \leq j \leq j'$ *and*
- v *is a subsequence of* v'.

Definition 8. *An* L*-program* p satisfies the NIE property *iff there exist:*

- $I \subseteq noncodeToks(p)$ *(i.e., a set of* p*'s inserted noncode tokens),*
- $n \in \mathbb{N}$ *(i.e., a number of* p*'s expanded noncode tokens),*
- $\{t_1..t_n\} \subseteq tokenize([\varepsilon/\underline{\sigma}]p)$ *(i.e., a set of template tokens to be expanded), and*
- $\{t'_1..t'_n\} \subseteq noncodeToks(p)$ *(i.e., a set of* p*'s expanded noncode tokens)*

such that:

- $t_1 \preceq t'_1, \ldots, t_n \preceq t'_n$, *and*
- $tokenize(p) = ([t'_1/t_1]..[t'_n/t_n]tokenize([\varepsilon/\underline{\sigma}]p)) \cup I$.

Definitions

Definition 9. *A* BroNIE *(Broken NIE) occurs exactly when a taint-tracking application outputs a program that violates the NIE property.*

Theorem 1. *If a program exhibits a CIAO, then it exhibits a BroNIE.*

Theorem 2. *For all n-ary functions A and (n-1)-ary functions A′ and A″, if* $\forall i_1,..,i_n{:}\, A(i_1,..,i_n) = A'(i_1,..,i_{m-1},i_{m+1},..,i_n)\underline{i_m}A''(i_1,..,i_{m-1},i_{m+1},..,i_n)$, *where* $1 \leq m \leq n$, *and* $\exists v_1,..,v_n{:}\,(v_m \in \Sigma^+_{SQL} \wedge A(v_1,..,v_n) \in SQL)$, *then* $\exists a_1,..,a_n{:}\, A(a_1,..,a_n) \in SQL$ *and* $A(a_1,..,a_n)$ *exhibits a CIAO and a BroNIE.*

Theorem 3. *Algorithm 1 executes output-program p iff p does not exhibit a BroNIE.*

Theorem 4. *The BroNIE-detection part of Algorithm 1 (i.e., Lines 2–27) executes in* $O(n)$ *time, where n is the length of the output program.*

BroNIE in Practice