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:
Motivation

Account number: 0001

Password: 123456 | Get My Balance

http://mybank.com
Web Application

\[
sql = \text{SELECT balance from accts}
\text{WHERE num=0001 AND password=' + input() + '}
\]

\[
result = \text{executeQuery(sql)}
\]

\[
\text{showResult(result)}
\]
Motivation

Web Application

\[
sql = \text{SELECT balance from accts}
\]
\[
\text{WHERE num=0001 AND password='} + \text{input()} + '\n\]
\[
result = \text{executeQuery(sql)}
\]
\[
\text{showResult(result)}
\]

Output Program

\[
\text{SELECT balance from accts WHERE num=0001 AND password='123456'}
\]
Motivation

Web Application

```
sql = SELECT balance from acct WHERE num=0001 AND password='input()'
result = executeQuery(sql)
showResult(result)
```

Output Program

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

Database
Motivation

Web Application

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

result = executeQuery(sql)

showResult(result)
```

Output Program

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

Database

Account Number Balance
0001 10$

Motivation

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

\[
sql = \text{SELECT balance from accts WHERE num=0001 AND password=' + input() + '}
\]

\[
result = \text{executeQuery(sql)}
\]

\[
\text{showResult(result)}
\]

Output Program

\[
\text{SELECT balance from accts WHERE num=0001 AND password=' OR 1=1 --'}
\]
Motivation

Web Application

\[
sql = \text{SELECT balance from accts WHERE num=0001 AND password=' + input() + '}
\]

\[
result = \text{executeQuery}(sql)
\]

\[
\text{showResult}(result)
\]

Output Program

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

Web Application

\[
sql = \text{SELECT balance from accts}
\text{WHERE num=0001 AND password=' + input() + '}
\]

result = executeQuery(sql)

showResult(result)

Output Program

\[
\text{SELECT balance from accts WHERE num=0001 AND password=' OR 1=1 --'}
\]
Motivation

http://sendmessage.com

Message: Hello

Send
Motivation

Web Application

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

result = sendMessage(code)
showResult(result)
```
Motivation

Web Application

\[
\text{code} = \text{\$data} = ' + \text{input()} + ';
\text{securityCheck(); \$data .} = '\&f=exit#';
\text{f()}
\]

\[
\text{result} = \text{sendMessage(\text{code})}
\]

\[
\text{showResult(\text{result})}
\]

Output Program

\[
\text{\$data} = 'Hello';
\text{securityCheck(); \$data .} = '\&f=exit#';
\text{f()}
\]

HTTP://sendmessage.com
Motivation

Web Application

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

result = sendMessage(code)
showResult(result)
```

Output Program

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

Message Sent
Motivation

Web Application

\[
\text{code} = \$\text{data} = ' + \text{input()} + ' \\
\text{securityCheck}(); \$\text{data} .= ' &f=exit#' ; \\
f() \\
\]

\[
\text{result} = \text{sendMessage}('code') \\
\text{showResult}(\text{result}) \\
\]

Output Program

\[
\$\text{data} = 'Hello'; \\
\text{securityCheck}(); \$\text{data} .= ' &f=exit#'; \\
f() \\
\]

Message sent successfully!
Motivation
Motivation

Web Application

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

   result = sendMessage(code)

   showResult(result)
)
Web Application

\[
\text{code} = \text{data} = ' + \text{input()} + ';
\text{securityCheck(); } \text{data} .= '\&f=exit#';
\text{f();}
\]

\[
\text{result} = \text{sendMessage(code)}
\]

\[
\text{showResult(result)}
\]
Message: 

Web Application

\[ code = \$data = ' + \text{input()} + '; \]
\[ \text{securityCheck(); } \$data .= '&f=exit#'; \]
\[ f() \]

\[ result = \text{sendMessage}(\text{code}) \]
\[ \text{showResult(result)} \]

Output Program

\[ \$data = '\'; \]
\[ \text{securityCheck(); } \$data .= '&f=exit#'; \]
\[ f() \]

Server is not responding
**Motivation**

**Web Application**

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

result = sendMessage(code)
showResult(result)
```

**Output Program**

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

Message Server

Server is not responding
Motivation

http://sendmessage.com

Message: Hello

Send
Motivation

Web Application

Message: Hello

Error 521
Web server is down

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.
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

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


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.

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

Output Program

```sql
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** (Noncode Insertion or Expansion) 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=' '
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**
Examples

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

**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:**
\[
\text{
$data='\; $data .='&f=exit#'; \n f();
\]

**Template program:**
\[
\text{
$data=''; $data .='&f=exit#'; \n f();
\]
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();
```

**Template program:**
```
$data='\'; securityCheck(); $data .='&f=exit#'; \n f();
```
Examples

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 (0, -10)
INSERT INTO trans VALUES (0, 0+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)`

**Template program:**
- `INSERT INTO trans VALUES ( , -10)`
- `INSERT INTO trans VALUES ( , +5)`

(float, 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:

```java
String sql = "SELECT balance from accts WHERE password=" + input();
```

Parameterized query:

```java
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

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**
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

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"

Input → Application → output program

forced to follow same control flow path

"intended" output program

VR → Application

Parse TreeComparer

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()
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

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 whitespace 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**

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

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

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

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

Legend:
- **Injected symbols**
- **Code tokens**
- **Noncode tokens**
Definitions

Definition 1 ([4]). For all alphabets $\Sigma$, the tainted-symbol alphabet $\overline{\Sigma}$ is

$$\{\sigma \mid \sigma \in \Sigma \lor (\exists \sigma' \in \Sigma : \sigma = \overline{\sigma'})\}.$$ 

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

Definition 2 ([4]). For all languages $L$ with alphabet $\Sigma$, the tainted output language $\overline{L}$ with alphabet $\overline{\Sigma}$ is

$$\{\sigma_1..\sigma_n \mid \exists \sigma'_1..\sigma'_n \in L: \forall i \in \{1..n\}: (\sigma_i = \overline{\sigma'_i} \lor \sigma_i = \overline{\sigma'_i})\}.$$ 

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

Definition 3 ([4]). For all alphabets $\Sigma$ and symbols $\sigma \in \Sigma$, 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 $L$-program $p = \sigma_1..\sigma_n$ such that $\exists i \in \{1..n\} : (injected(\sigma_i) \land Code(p, i))$.

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

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

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

such that:

- \( t_1 \leq t'_1, \ldots, t_n \leq t'_n \), and
- \( \text{tokenize}(p) = ([t'_1/t_1]..[t'_n/t_n]\text{tokenize}([\varepsilon/\sigma]p)) \cup I \).
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 $orall i_1,\ldots,i_n: A(i_1,\ldots,i_n) = A'(i_1,\ldots,i_{m-1},i_{m+1},\ldots,i_n)\overline{i_m}A''(i_1,\ldots,i_{m-1},i_{m+1},\ldots,i_n)$, where $1 \leq m \leq n$, and $\exists v_1,\ldots,v_n: (v_m \in \Sigma^{+}_{SQL} \land A(v_1,\ldots,v_n) \in SQL)$, then $\exists a_1,\ldots,a_n: A(a_1,\ldots,a_n) \in SQL$ and $A(a_1,\ldots,a_n)$ 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

Untrusted Inputs → Trusted Taint-Tracker

Application (that might be vulnerable)

Output program + Taint information

Not an attack. Execute output program. → Trusted Injected Attack Detector → Is an attack. Raise exception.