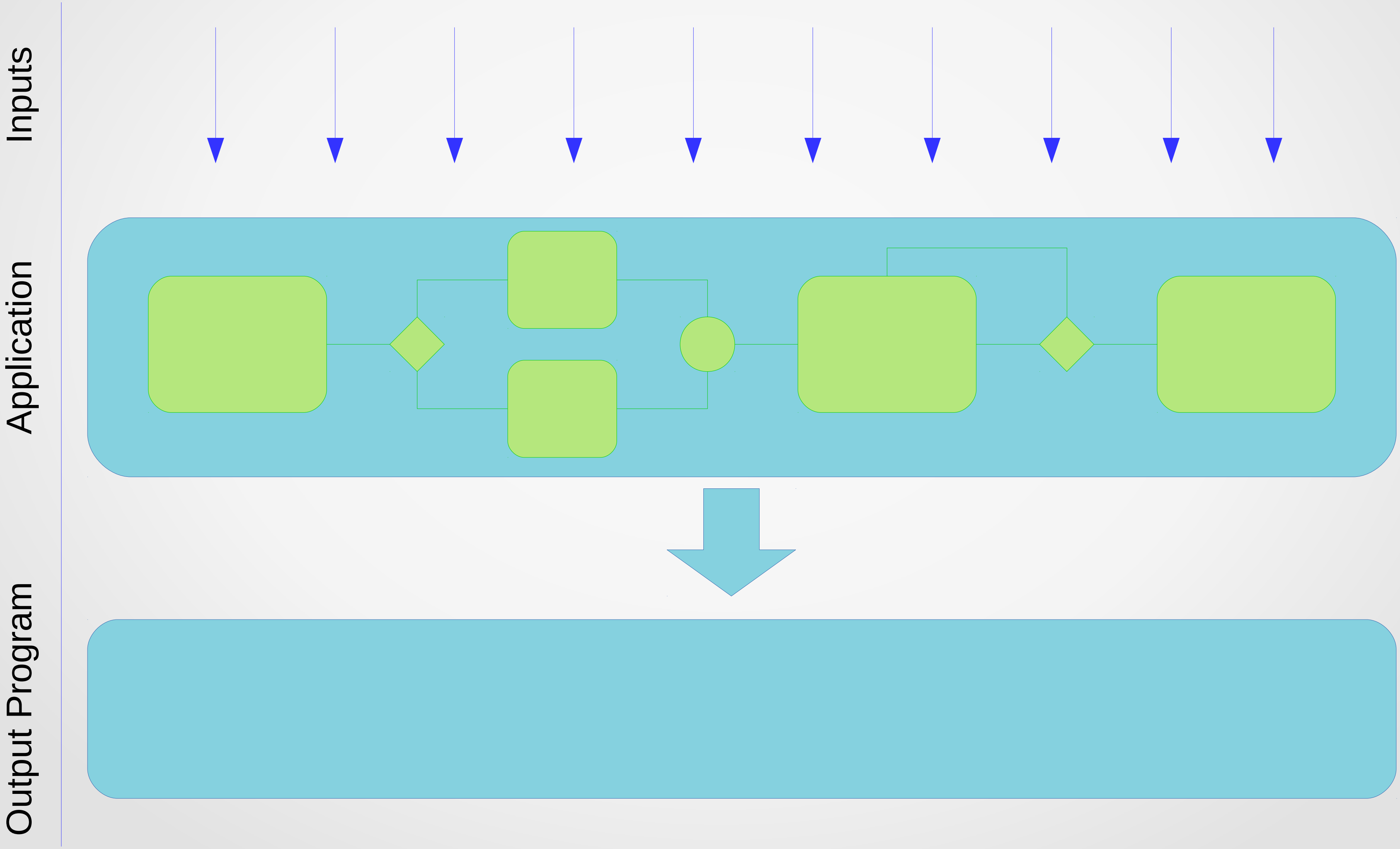
Defining Injection Attacks

RA:
Donald Ray
dray3@cse.usf.edu

PI:
Jay Ligatti
ligatti@cse.usf.edu

Motivation

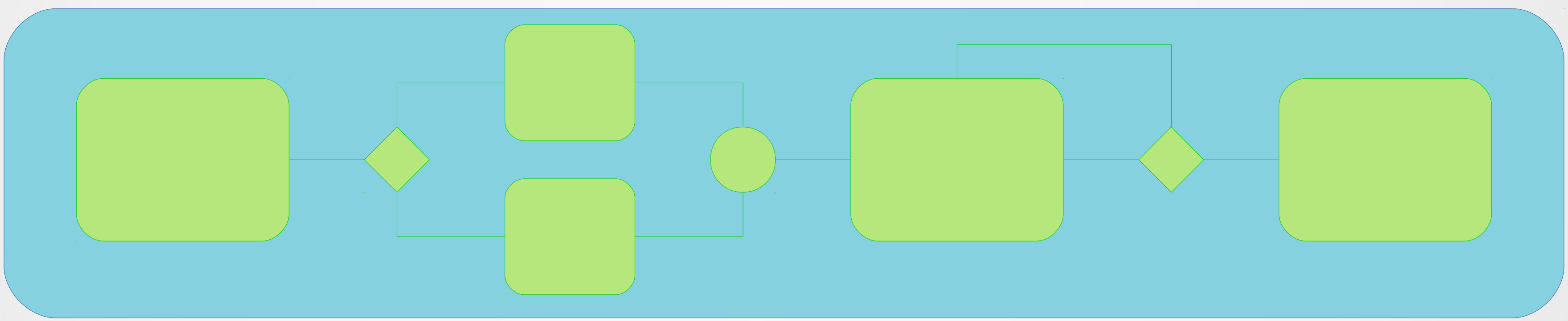


Motivation

Inputs

123456

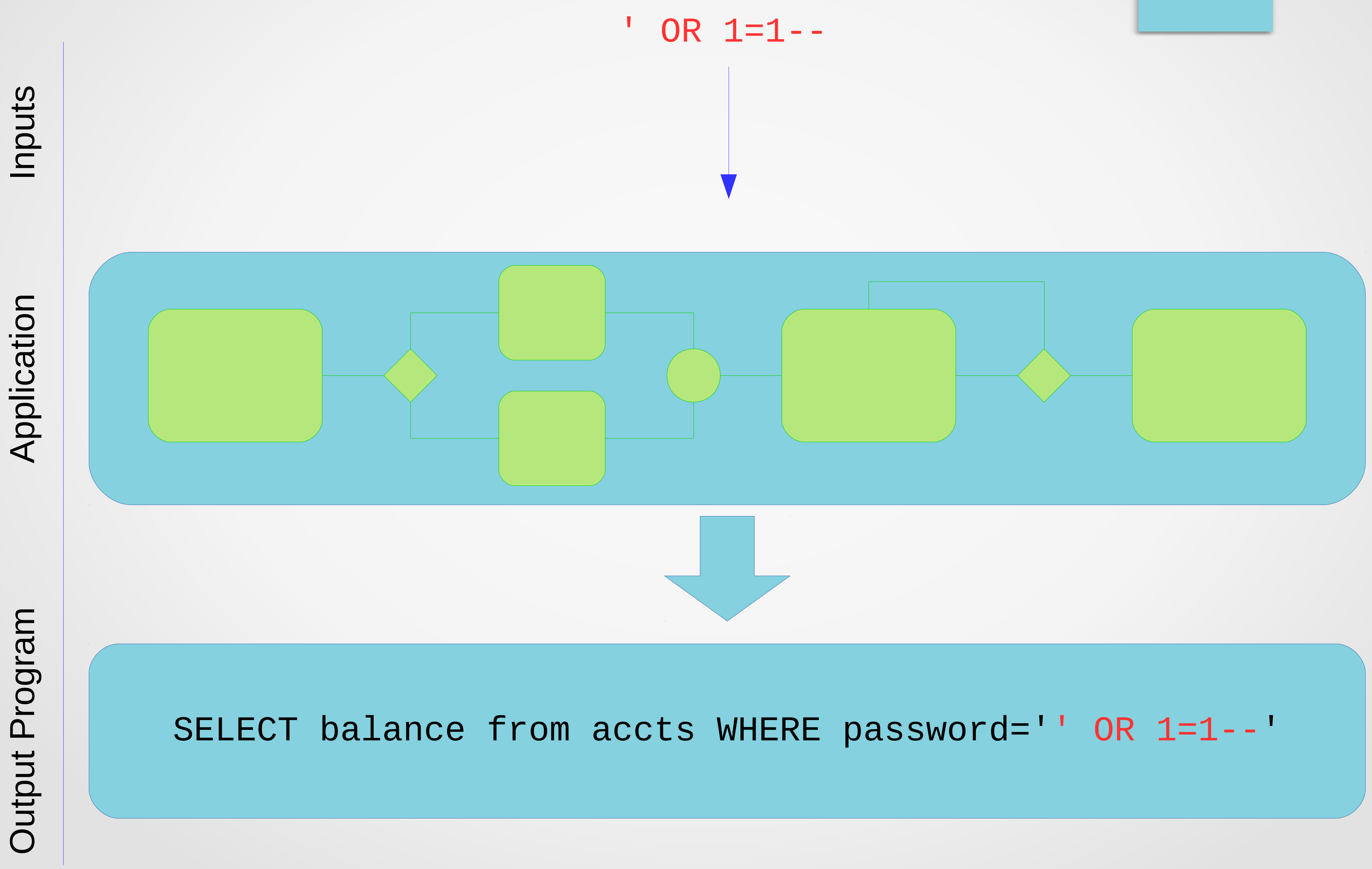
Application



Output Program

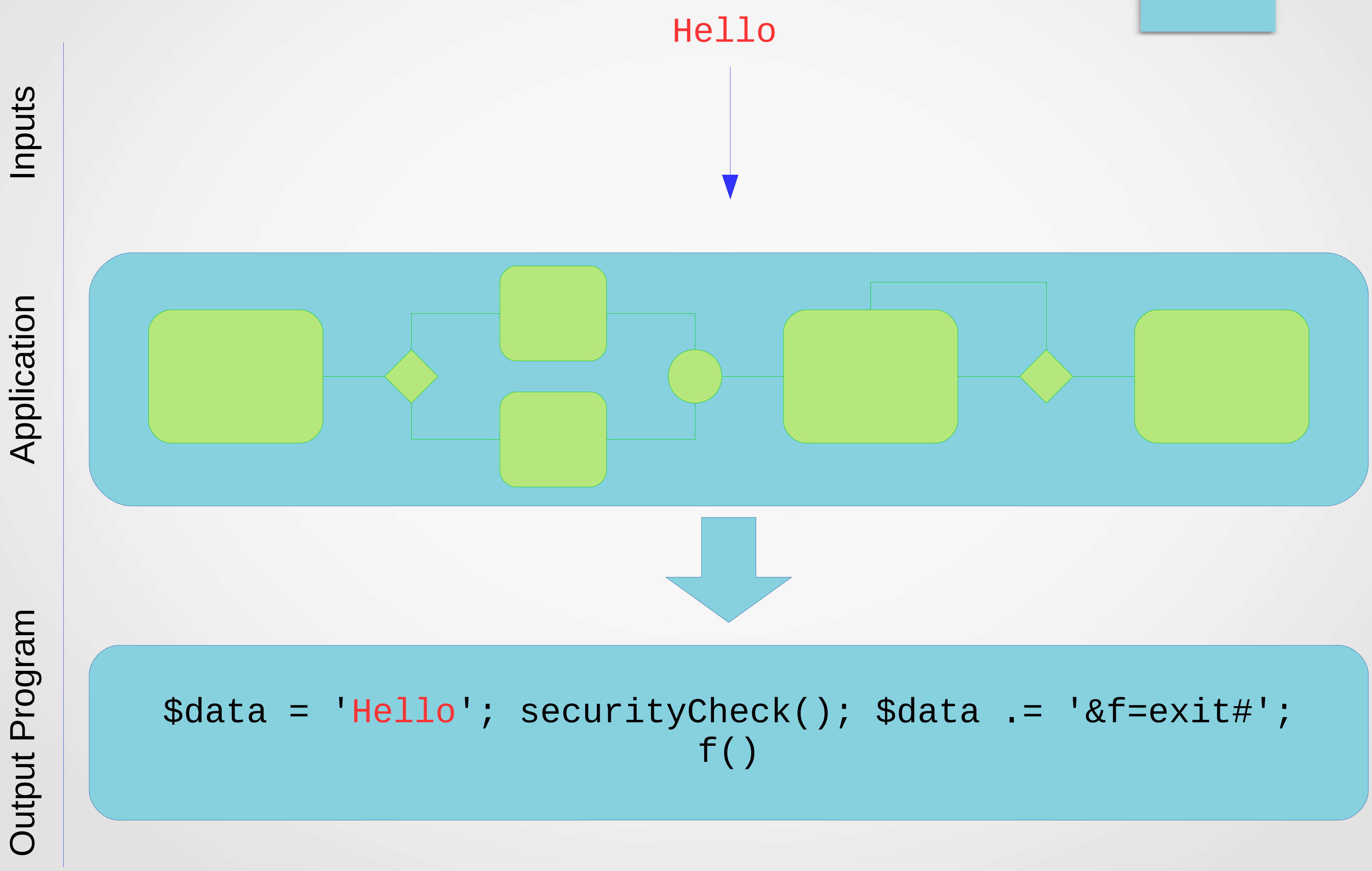
```
SELECT balance from accts WHERE password='123456'
```

Motivation

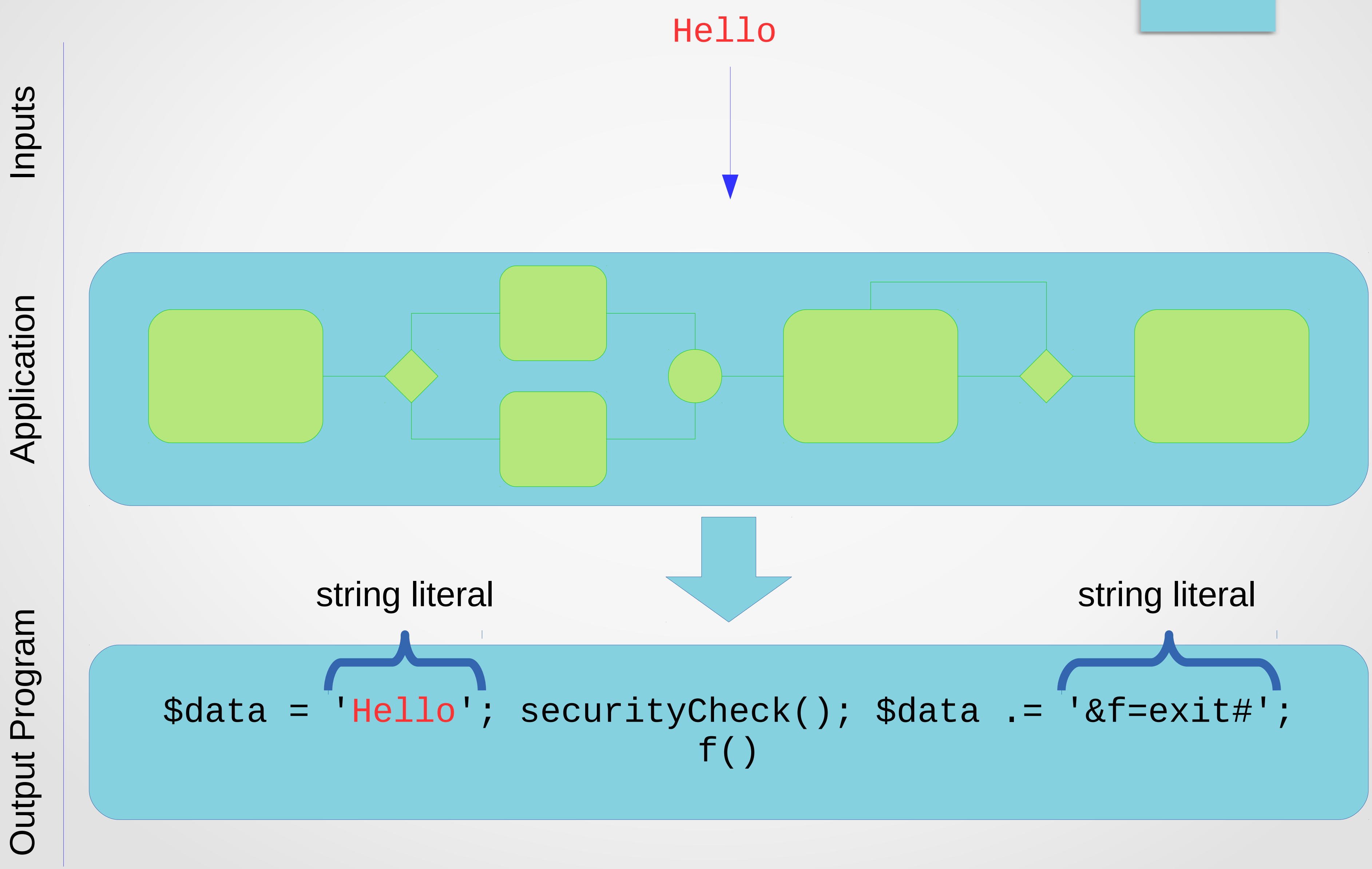


Inputs
Application
Output Program
' OR 1=1--
SELECT balance from accts WHERE password='' OR 1=1--'

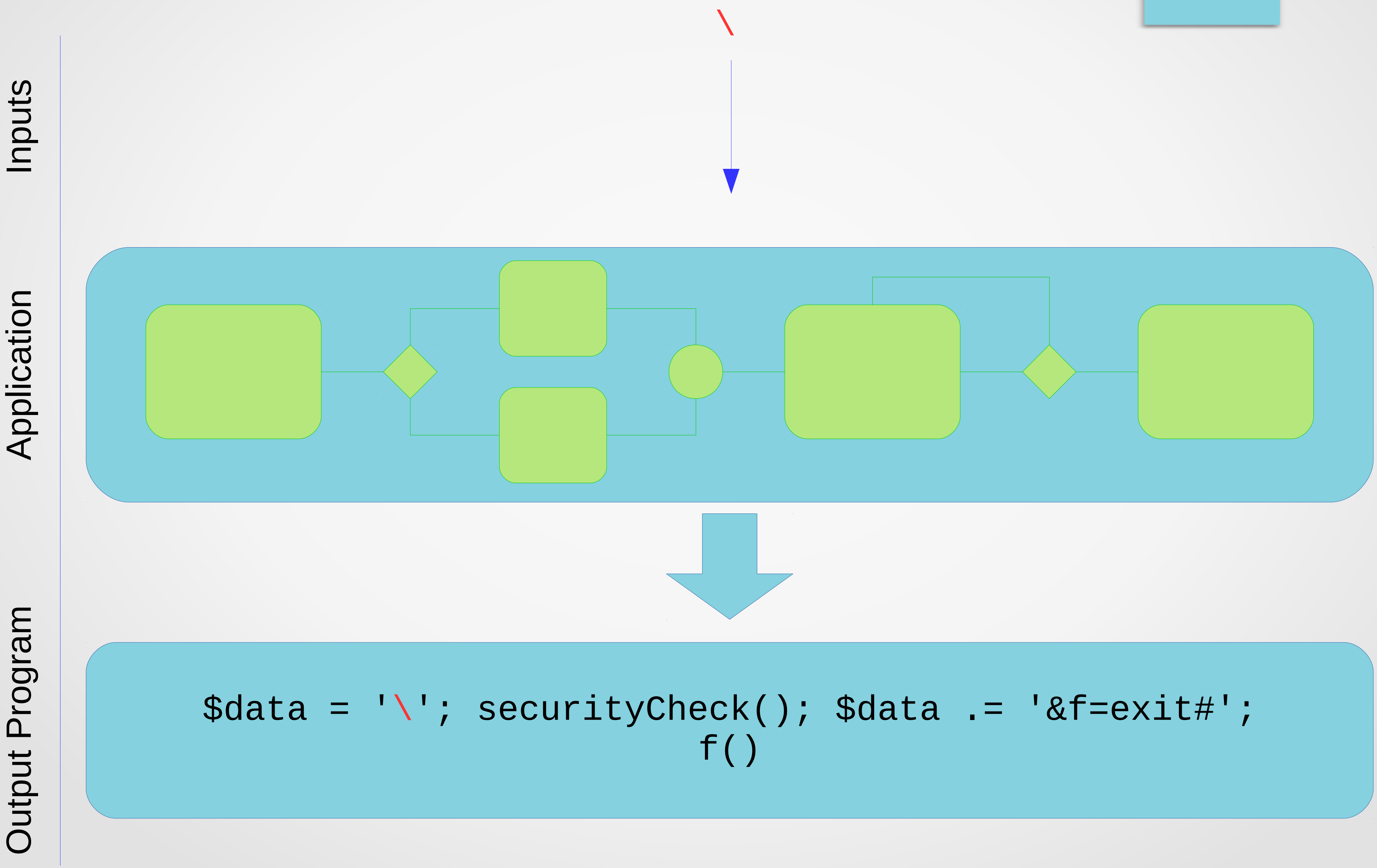
Motivation


Inputs
Application
Output Program
Hello
$data = 'Hello'; securityCheck(); $data .= '&f=exit#';
f()

Motivation

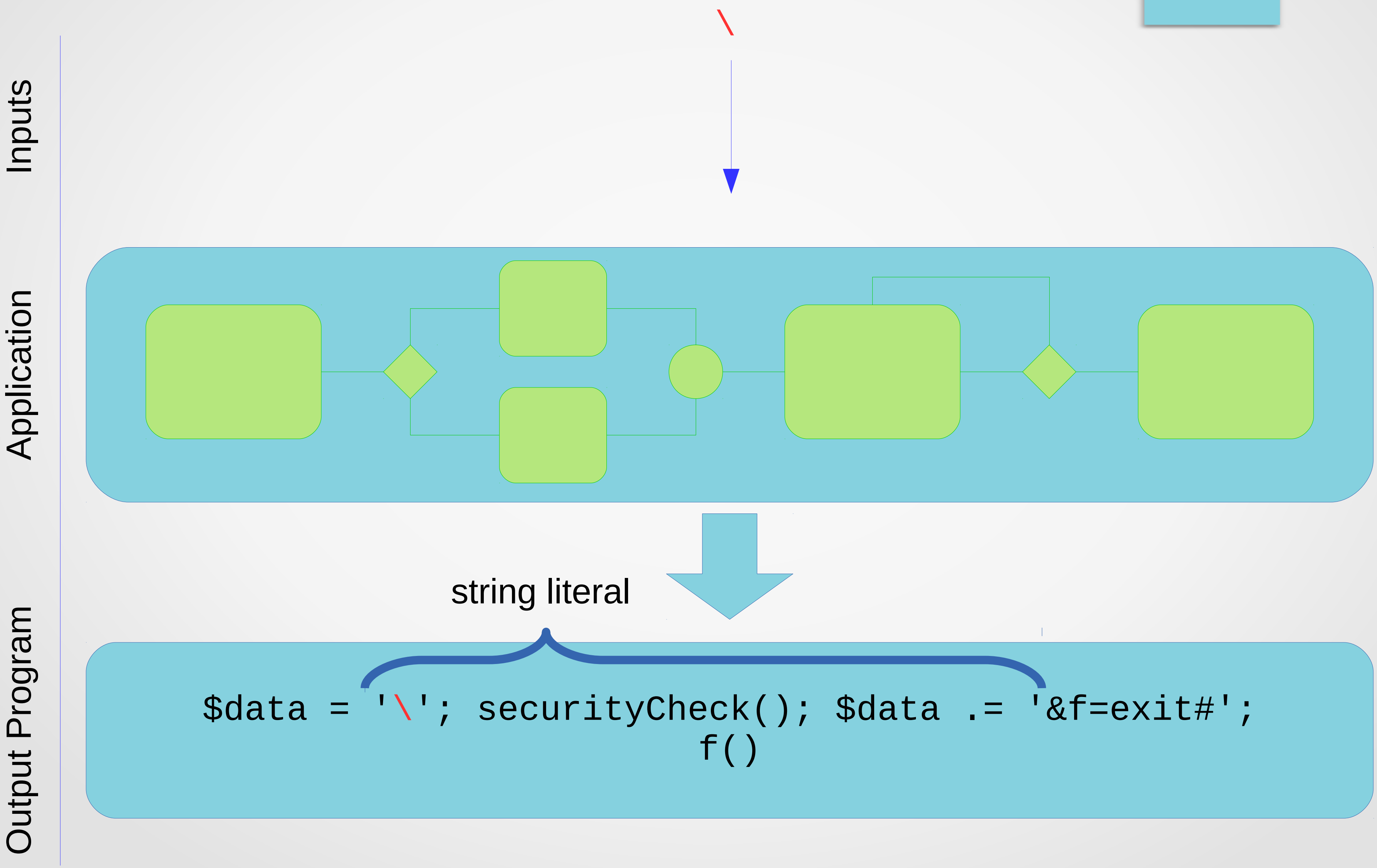

Hello
Inputs
Application
Output Program
string literal
string literal
$data = 'Hello'; securityCheck(); $data .= '&f=exit#';
f()

Motivation



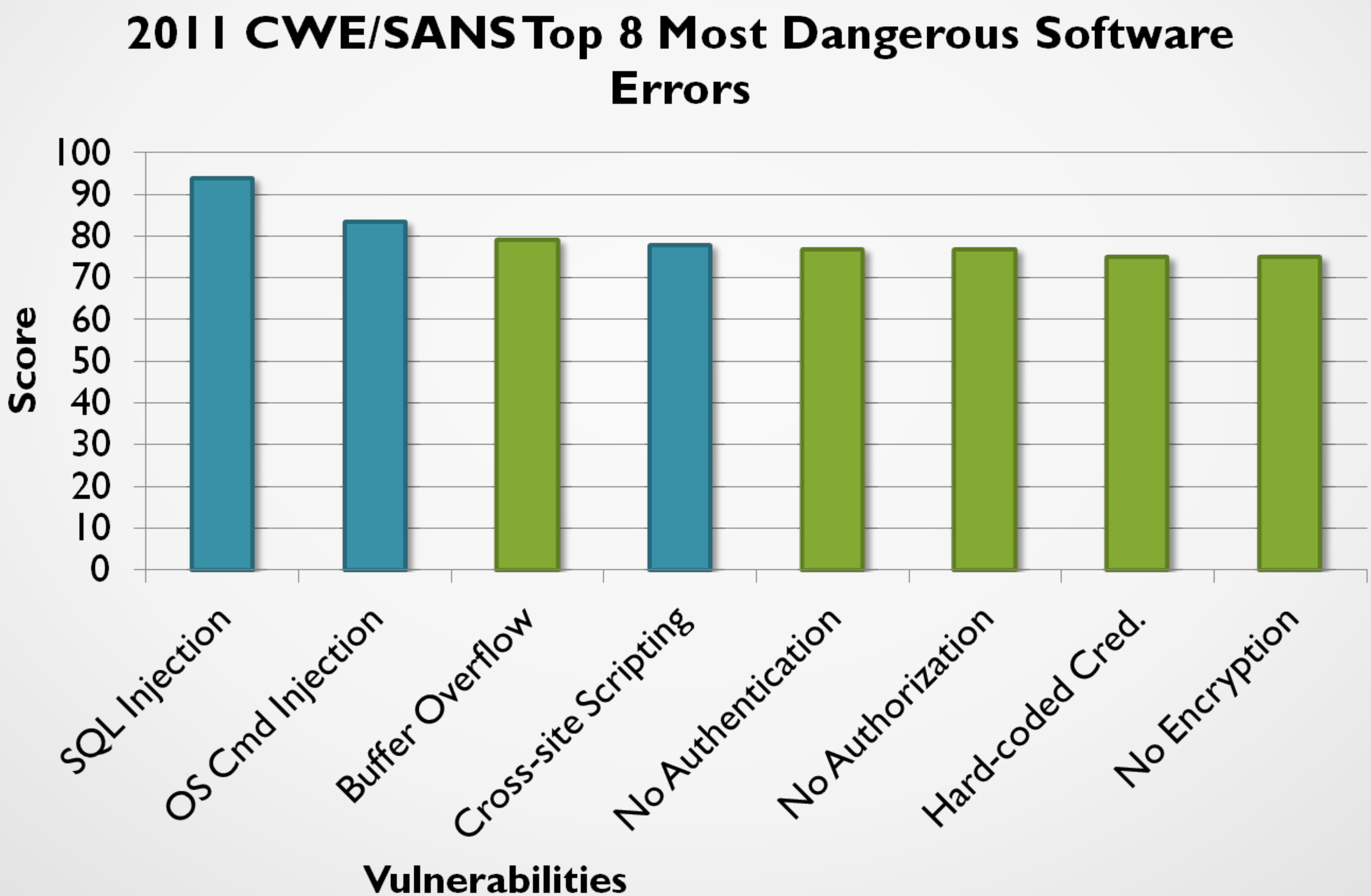
\
Inputs
Application
Output Program
$data = '\'; securityCheck(); $data .= '&f=exit#';
f()

Motivation



\
Inputs
Application
Output Program
string literal
$data = '\'; securityCheck(); $data .= '&f=exit#';
f()

Motivation



2011 CWE/SANS Top 8 Most Dangerous Software Errors
100
90
80
70
60
50
40
30
20
10
0
Score
SQL Injection
OS Cmd Injection
Buffer Overflow
Cross-site Scripting
No Authentication
No Authorization
Hard-coded Cred.
No Encryption
Vulnerabilities

Outline

- Motivation
- **Related work**
- Defining injection attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- Preventing injection attacks in practice.

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:

Ray, D., Ligatti, J.: Defining code-injection attacks. In: Proceedings of the Symposium on Principles of Programming Languages (POPL). (2012) 179–190

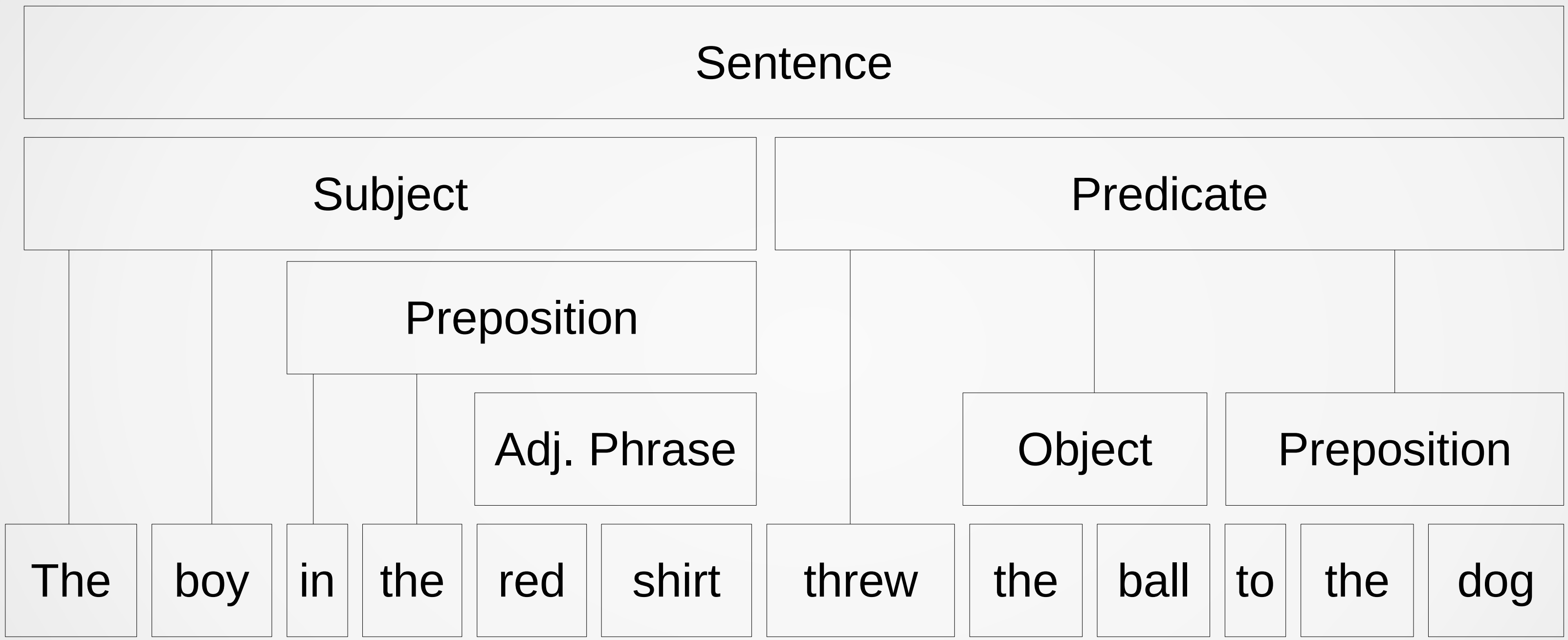
Program Parsing

The boy in the red shirt threw the ball to the dog.

Program Parsing

The	boy	in	the	red	shirt	threw	the	ball	to	the	dog

Program Parsing



Related Work: SqlCheck

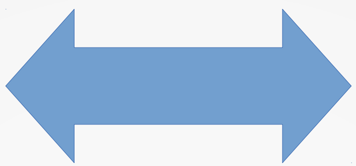
Program does not exhibit an attack

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

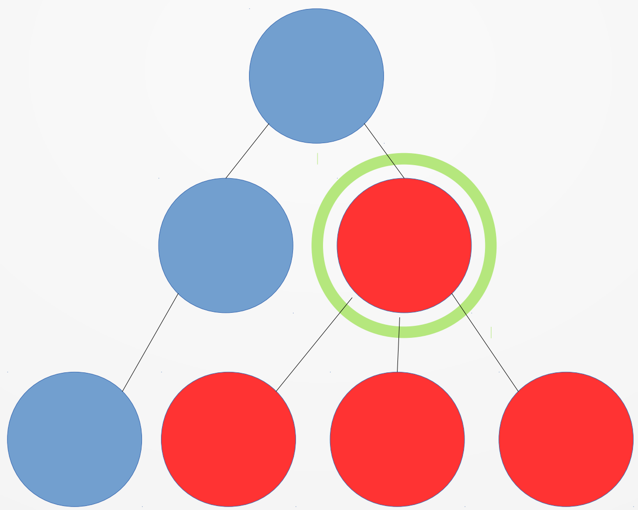
Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: Proceedings of the Symposium on Principles of Programming Languages (POPL). (2006) 372–382

Related Work: SqlCheck

Program does not exhibit an attack



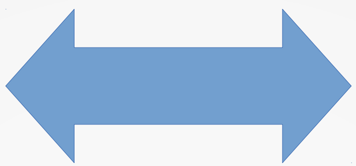
There is a node in the program's parse tree that is entirely injected and that contains all injected symbols



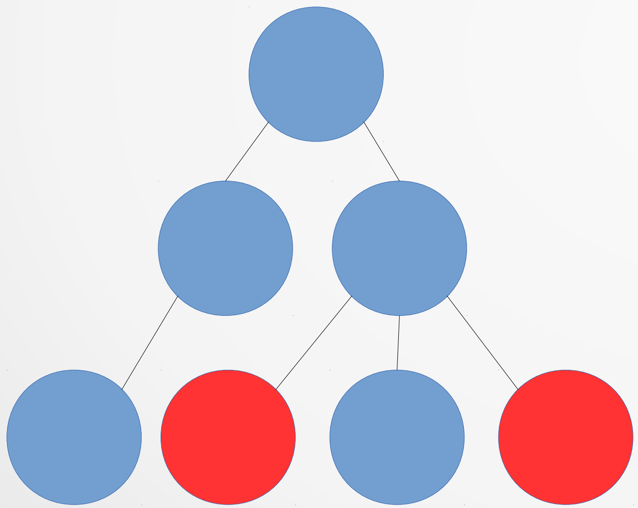
Not an attack

Related Work: SqlCheck

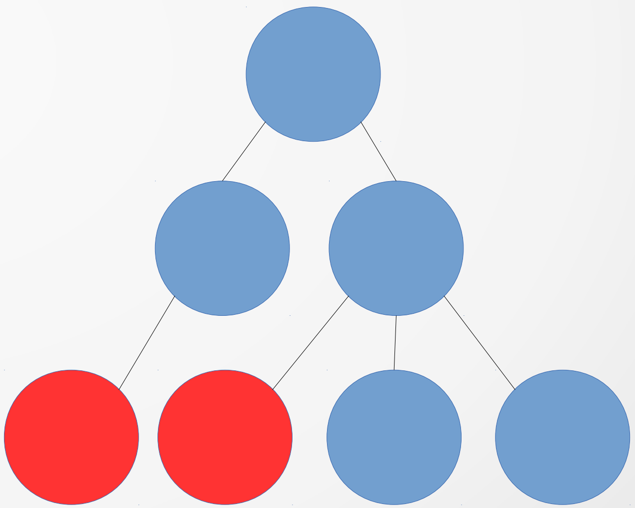
Program does not exhibit an attack



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()`

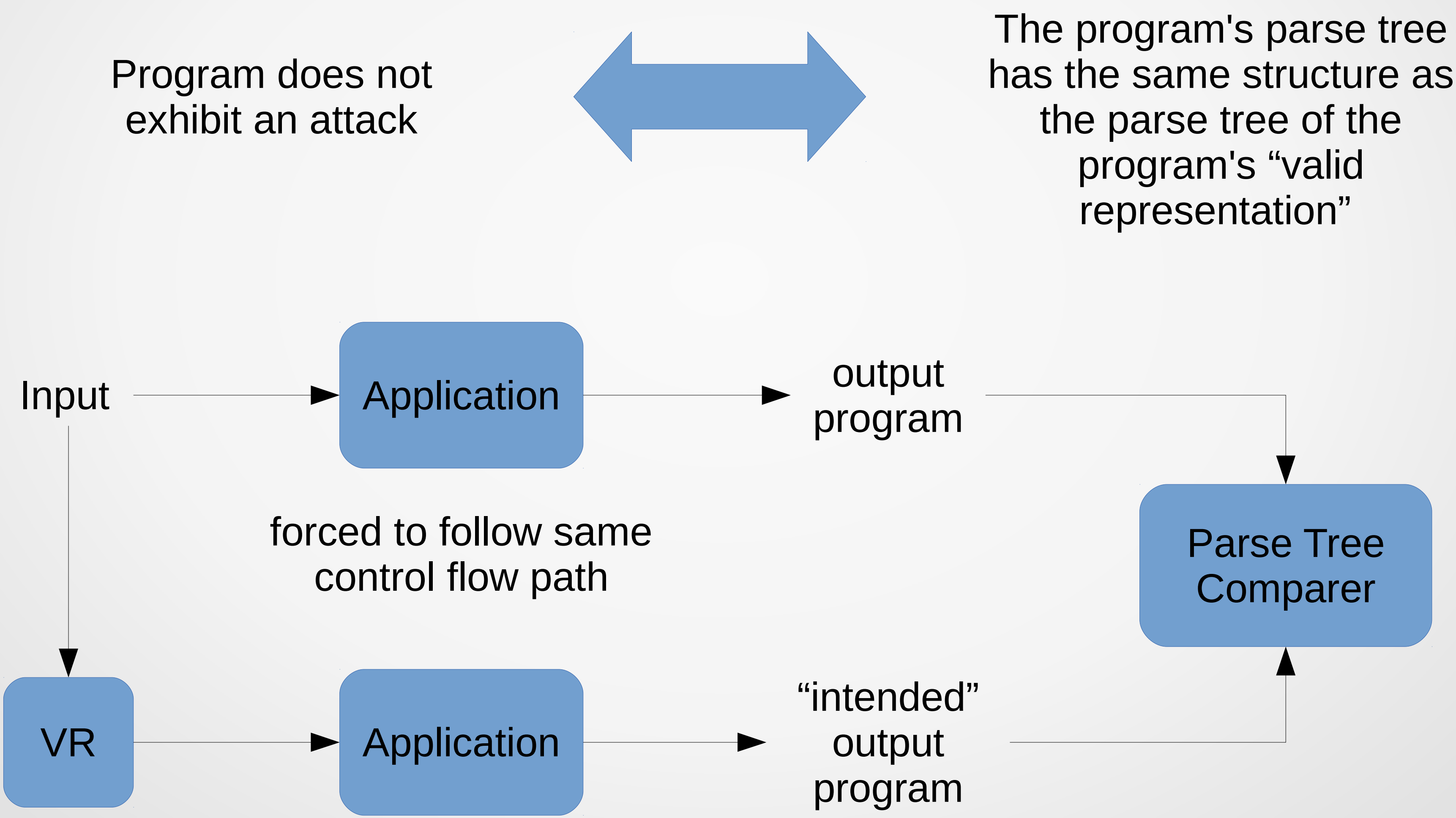
Related Work: CANDID

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”

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



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()`

Related Work: Parameterized Queries

High-level Idea

Don't let applications output programs with injected input in them.

Instead, have applications output untrusted input separately from a program template that has placeholders for where the untrusted inputs should be used.

Related Work: Parameterized Queries

Problems:

- Not implemented in many output languages.
- Not mandatory to use
- Requires significant, manual rewrites in the application

Outline

- Motivation
- Related work
- **Defining injection attacks**
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- Preventing injection attacks in practice.

Defining Injection Attacks

High-level Approach

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

Defining Injection Attacks

High-level Approach

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

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

Defining Injection Attacks

High-level Approach

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

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

Defining Injection Attacks

High-level Approach

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

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

Defining Injection Attacks

High-level Approach

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

Requires subdefinitions of “injected” and “noncode”

Outline

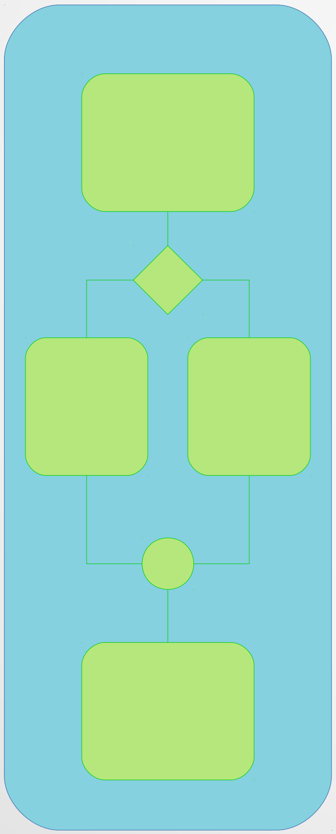
- Motivation
- Related work
- Defining injection attacks
  - **Defining injection**
  - Defining code
  - Defining NIEs
- Examples
- Preventing injection attacks in practice.

Defining Injection

Intuitively, a symbol has been *injected* if it propagates unmodified from an untrusted input into the output program

Defining Injection

Intuitively, a symbol has been *injected* if it propagates unmodified from an untrusted input into the output program



```
output('Math.' + input() + '(4.3)')

output('Math.' + 'floor' + '(4.3)')

output('Math.floor' + '(4.3)')

output('Math.floor(4.3)')
```

Taint-Tracking Mechanisms

- Taint all untrusted inputs to an application.
- Taints propagate *transparently* through copy and output operations during execution.
- A symbol has been *injected* if and only if it is tainted in the output of a taint-tracking application.

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

Outline

- Motivation
- Related work
- Defining injection attacks
  - Defining injection
  - **Defining code**
  - Defining NIEs
- Examples
- Preventing injection attacks in practice.

Defining (Non)Code

Intuitively, noncode symbols in output programs are those that are *dynamically passive:*

- Values (i.e., normal forms)

Defining (Non)Code

Intuitively, noncode symbols in output programs are those that are *dynamically passive:*

- Values (i.e., normal forms)

```
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)]
```

Defining (Non)Code

Intuitively, noncode symbols in output programs are those that are *dynamically passive:*

- Values (i.e., normal forms)

```
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

Defining (Non)Code

Intuitively, noncode symbols in output programs are those that are *dynamically passive:*

- Values (i.e., normal forms)

```
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  */
```

Defining (Non)Code

Some technicalities:

- 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.

Symbol Taxonomy

```
SELECT_*_FROM_t_WHERE_name_=_'admin'
```

```
SELECT_*_FROM_t_WHERE_perimeter_>_(5_*_3.14)
```

```
SELECT_*_FROM_t_WHERE_name_=_NULL--comment
```

```
INSERT_INTO_t_VALUES_(1,_true,_'Alice')
```

Legend:
Noncode symbols
Code symbols

Side note: Defining Code-injection Attacks

A CIAO (**C**ode-**I**njection **A**ttack on **O**utput program) occurs exactly when a taint-tracking application outputs a program that contains a symbol that is both injected and code.

Side note: Defining Code-injection Attacks

A CIAO (**C**ode-**I**njection **A**ttack on **O**utput program) occurs exactly when a taint-tracking application outputs a program that contains a symbol that is both injected and code.

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

SELECT_balance_from_accts_WHERE_password=''_OR_1=1--'
```

Side note: Defining Code-injection Attacks

A CIAO (**C**ode-**I**njection **A**ttack on **O**utput program) occurs exactly when a taint-tracking application outputs a program that contains a symbol that is both injected and code.

```
SELECT balance from accts WHERE password='' OR 1=1--'
SELECT_balance_from_accts_WHERE_password=''_OR_1=1--'
```

Outline

- Motivation
- Related work
- Defining injection attacks
  - Defining injection
  - Defining code
  - **Defining NIEs**
- Examples
- Preventing injection attacks in practice.

Defining NIEs

Reminder:

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

Defining NIEs

Reminder:

Injected symbols should not affect output programs beyond the insertion or expansion of noncode tokens.

In other words:

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

Defining NIEs

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

SELECT balance from accts WHERE password=''
```

Defining NIEs

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

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

Defining Injection Attacks

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

Outline

- Motivation
- Related work
- Defining injection attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- **Examples**
- Preventing injection attacks in practice.

Examples

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

```
SELECT balance from accts WHERE password=''
```

Examples

string literal

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

string literal

string literal

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

Examples

comment

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

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

Examples

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

INSERT INTO trans VALUES (,- -10);
INSERT INTO trans VALUES (, +5)
```

Outline

- Motivation
- Related work
- Defining injection attacks
  - Defining injection
  - Defining code
  - Defining NIEs
- Examples
- **Preventing injection attacks in practice**

CIAO-BroNIE Relationship

Theorem 1:

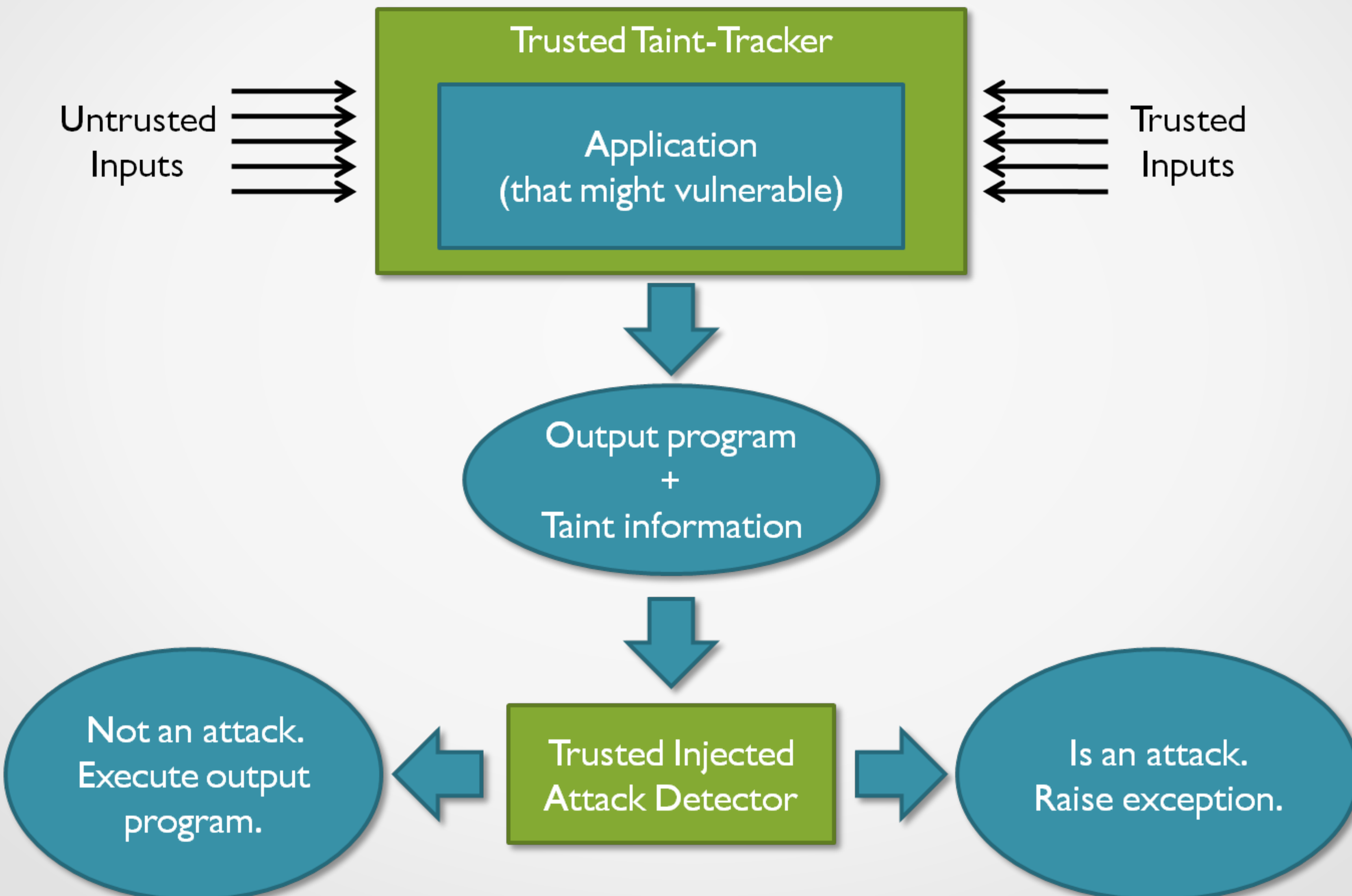
Any program that exhibits a CIAO also exhibits a BroNIE

Pervasiveness of CIAOs and BroNIEs

Theorem 2:

Any application that generates SQL programs by blindly copying some input into its output is vulnerable to both CIAOs and BroNIEs.

Preventing BroNIEs



An Algorithm for Preventing BroNIEs

```
p = A(T, Taint(U))
pgmTokens = tokenize(p)
temTokens = tokenize(p.removeInjected())
MarkNoncodeTokens(pgmTokens)
i = j = 1
while i <= pgmTokens.length and j <= temTokens.length:
    if pgmTokens[i] = temTokens[j]:
        increment i and j
    else if pgmTokens[i].isNoncode and temTokens[i].canExpandInto(pgmTokens[i]):
        increment i and j
    else if pgmTokens[i].isNoncode:
        increment i
    else:
        throw BronieException
while i <= pgmTokens.length and pgmTokens[i].isNoncode:
    increment i
if i > pgmTokens.length and j > temTokens.length:
    execute(P)
else:
    throw BronieException
```

An Algorithm for Preventing BroNIEs

Theorem 3:

The algorithm executes the output program iff it does not exhibit a BroNIE

Theorem 4:

The algorithm executes in time linear in the length of the output program

Taint-tracking Problems

- Could be hard to identify all untrusted inputs
- Could be hard to identify all ways to output a program
- Taints must be tracked with fine granularity $\Longrightarrow$high runtime overhead
  - Taint tracking could interfere with real-time applications

Questions?

Project website:

http://www.cse.usf.edu/~ligatti/projects/ciao/

Extra Slides

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* $\mathit{injected}(\sigma)$ *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.*

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

```
SELECT * FROM orders WHERE s='x' AND true
```

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

```
SELECT * FROM orders WHERE s='x' AND true
```

1 10 20 30 40

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

```
SELECT * FROM orders WHERE s='x' AND true
```

1 10 20 30 40

$\texttt{SELECT}_{1}(\texttt{SELECT})_{6}$ $\texttt{STAR}_{8}(\texttt{*})_{8}$

$\texttt{FROM}_{10}(\texttt{FROM})_{13}$ $\texttt{ID}_{15}(\texttt{orders})_{20}$ $\texttt{WHERE}_{22}(\texttt{WHERE})_{26}$

$\texttt{ID}_{28}(\texttt{s})_{28}$ $\texttt{EQUALS}_{29}(\texttt{=})_{29}$ $\texttt{STRING}_{30}(\texttt{'x'})_{32}$

$\texttt{AND}_{34}(\texttt{AND})_{36}$ $\texttt{TRUE}_{38}(\texttt{true})_{41}$

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

If none of the symbols in a token *t* are code, then *t* is a *noncode token*.

Anatomy of a Token

A token of kind τ composed of symbols $\sigma_i..\sigma_j$

is represented as $\tau_i(\sigma_i..\sigma_j)_j$

If none of the symbols in a token *t* are code, then *t* is a *noncode token*.

$\texttt{SELECT}_1\texttt{(SELECT)}_6$ $\texttt{STAR}_8\texttt{(*)}_8$

$\texttt{FROM}_{10}\texttt{(FROM)}_{13}$ $\texttt{ID}_{15}\texttt{(orders)}_{20}$ $\texttt{WHERE}_{22}\texttt{(WHERE)}_{26}$

$\texttt{ID}_{28}\texttt{(s)}_{28}$ $\texttt{EQUALS}_{29}\texttt{(=)}_{29}$ $\texttt{STRING}_{30}\texttt{('x')}_{32}$

$\texttt{AND}_{34}\texttt{(AND)}_{36}$ $\texttt{TRUE}_{38}\texttt{(true)}_{41}$

Removing Symbols

The *template* of a program p is obtained by replacing each injected symbol in p with an ε.

Removing Symbols

The *template* of a program p is obtained by replacing each injected symbol in p with an ε.

```
SELECT * FROM orders WHERE s='x' AND true
```

Removing Symbols

The *template* of a program p is obtained by replacing each injected symbol in p with an ε.

```
SELECT * FROM orders WHERE s='x' AND true
SELECT * FROM orders WHERE s='ε' AND εεεε
```

The sole purpose of an ε symbol is to hold the place of an injected symbol; ε's are otherwise ignored.

Removing Symbols

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\mathtt{SELECT_{1}(SELECT)_{6}}$ $\mathtt{STAR_{8}(*)_{8}}$ $\mathtt{FROM_{10}(FROM)_{13}}$
$\mathtt{ID_{15}(orders)_{20}}$ $\mathtt{WHERE_{22}(WHERE)_{26}}$ $\mathtt{ID_{28}(s)_{28}}$ $\mathtt{EQUALS_{29}(=)_{29}}$
$\mathtt{STRING_{30}('x')_{32}}$ $\mathtt{AND_{34}(AND)_{36}}$ $\mathtt{TRUE_{38}(true)_{41}}$

Template Tokens:

$\mathtt{SELECT_{1}(SELECT)_{6}}$ $\mathtt{STAR_{8}(*)_{8}}$ $\mathtt{FROM_{10}(FROM)_{13}}$
$\mathtt{ID_{15}(orders)_{20}}$ $\mathtt{WHERE_{22}(WHERE)_{26}}$ $\mathtt{ID_{28}(s)_{28}}$ $\mathtt{EQUALS_{29}(=)_{29}}$
$\mathtt{STRING_{30}('')_{32}}$ $\mathtt{AND_{34}(AND)_{36}}$

Removing Symbols

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\mathrm{SELECT}_1(\mathrm{SELECT})_6$ $\mathrm{STAR}_8(*)_8$ $\mathrm{FROM}_{10}(\mathrm{FROM})_{13}$
$\mathrm{ID}_{15}(\mathrm{orders})_{20}$ $\mathrm{WHERE}_{22}(\mathrm{WHERE})_{26}$ $\mathrm{ID}_{28}(\mathrm{s})_{28}$ $\mathrm{EQUALS}_{29}(=)_{29}$
$\boxed{\mathrm{STRING}_{30}(\text{'x'})_{32}}$ $\mathrm{AND}_{34}(\mathrm{AND})_{36}$ $\mathrm{TRUE}_{38}(\mathrm{true})_{41}$

Template Tokens:

$\mathrm{SELECT}_1(\mathrm{SELECT})_6$ $\mathrm{STAR}_8(*)_8$ $\mathrm{FROM}_{10}(\mathrm{FROM})_{13}$
$\mathrm{ID}_{15}(\mathrm{orders})_{20}$ $\mathrm{WHERE}_{22}(\mathrm{WHERE})_{26}$ $\mathrm{ID}_{28}(\mathrm{s})_{28}$ $\mathrm{EQUALS}_{29}(=)_{29}$
$\boxed{\mathrm{STRING}_{30}(\text{''})_{32}}$ $\mathrm{AND}_{34}(\mathrm{AND})_{36}$

Removing Symbols

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\mathrm{SELECT}_{1}(\mathrm{SELECT})_{6}$ $\mathrm{STAR}_{8}(*)_{8}$ $\mathrm{FROM}_{10}(\mathrm{FROM})_{13}$
$\mathrm{ID}_{15}(\mathrm{orders})_{20}$ $\mathrm{WHERE}_{22}(\mathrm{WHERE})_{26}$ $\mathrm{ID}_{28}(\mathrm{s})_{28}$ $\mathrm{EQUALS}_{29}(=)_{29}$
$\mathrm{STRING}_{30}(\text{'x'})_{32}$ $\boxed{\mathrm{AND}_{34}(\mathrm{AND})_{36}}$ $\mathrm{TRUE}_{38}(\mathrm{true})_{41}$

Template Tokens:

$\mathrm{SELECT}_{1}(\mathrm{SELECT})_{6}$ $\mathrm{STAR}_{8}(*)_{8}$ $\mathrm{FROM}_{10}(\mathrm{FROM})_{13}$
$\mathrm{ID}_{15}(\mathrm{orders})_{20}$ $\mathrm{WHERE}_{22}(\mathrm{WHERE})_{26}$ $\mathrm{ID}_{28}(\mathrm{s})_{28}$ $\mathrm{EQUALS}_{29}(=)_{29}$
$\mathrm{STRING}_{30}(\text{''})_{32}$ $\boxed{\mathrm{AND}_{34}(\mathrm{AND})_{36}}$

Removing Symbols

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\mathtt{SELECT_{1}(SELECT)_{6}\ STAR_{8}(*)_{8}\ FROM_{10}(FROM)_{13}}$
$\mathtt{ID_{15}(orders)_{20}\ WHERE_{22}(WHERE)_{26}\ ID_{28}(s)_{28}\ EQUALS_{29}(=)_{29}}$
$\mathtt{STRING_{30}('x')_{32}\ AND_{34}(AND)_{36}\ \boxed{TRUE_{38}(true)_{41}}}$

Template Tokens:

$\mathtt{SELECT_{1}(SELECT)_{6}\ STAR_{8}(*)_{8}\ FROM_{10}(FROM)_{13}}$
$\mathtt{ID_{15}(orders)_{20}\ WHERE_{22}(WHERE)_{26}\ ID_{28}(s)_{28}\ EQUALS_{29}(=)_{29}}$
$\mathtt{STRING_{30}('')_{32}\ AND_{34}(AND)_{36}}$

Expanding Tokens

Notation: If token t can be expanded into token t', we write $t \preccurlyeq t'$.

$$\text{'x'}: \quad \mathtt{STRING}_1(\text{''})_3 \preccurlyeq \mathtt{STRING}_1(\text{'x'})_3$$

$$1234: \quad \mathtt{INT}_2(23)_3 \preccurlyeq \mathtt{INT}_1(1234)_4$$

$$12.34: \quad \mathtt{INT}_1(1234)_5 \not\preccurlyeq \mathtt{FLOAT}_1(12.34)_5$$

The NIE Property

If the injected symbols of an output program only insert or expand noncode tokens, then we say that the program satisfies the NIE (**N**oncode **I**nsertion/**E**xpansion, pronounced “knee”) property.

The NIE Property

Definition Overview

If a program satisfies the NIE property, then it should be possible to get to the sequence of tokens the program from the sequence of tokens in the program's template by only inserting or expanding noncode tokens.

The NIE Property

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\mathtt{SELECT}_{1}\mathtt{(SELECT)}_{6}$ $\mathtt{STAR}_{8}\mathtt{(*)}_{8}$ $\mathtt{FROM}_{10}\mathtt{(FROM)}_{13}$
$\mathtt{ID}_{15}\mathtt{(orders)}_{20}$ $\mathtt{WHERE}_{22}\mathtt{(WHERE)}_{26}$ $\mathtt{ID}_{28}\mathtt{(s)}_{28}$ $\mathtt{EQUALS}_{29}\mathtt{(=)}_{29}$
$\mathtt{STRING}_{30}\mathtt{('x')}_{32}$ $\mathtt{AND}_{34}\mathtt{(AND)}_{36}$ $\mathtt{TRUE}_{38}\mathtt{(true)}_{41}$

Template Tokens:

$\mathtt{SELECT}_{1}\mathtt{(SELECT)}_{6}$ $\mathtt{STAR}_{8}\mathtt{(*)}_{8}$ $\mathtt{FROM}_{10}\mathtt{(FROM)}_{13}$
$\mathtt{ID}_{15}\mathtt{(orders)}_{20}$ $\mathtt{WHERE}_{22}\mathtt{(WHERE)}_{26}$ $\mathtt{ID}_{28}\mathtt{(s)}_{28}$ $\mathtt{EQUALS}_{29}\mathtt{(=)}_{29}$
$\mathtt{STRING}_{30}\mathtt{('')}_{32}$ $\mathtt{AND}_{34}\mathtt{(AND)}_{36}$

The NIE Property

Program: `SELECT * FROM orders WHERE s='x' AND true`

Template: `SELECT * FROM orders WHERE s='ε' AND εεεε`

Program Tokens:

$\texttt{SELECT}_{1}\texttt{(SELECT)}_{6}$ $\texttt{STAR}_{8}\texttt{(*)}_{8}$ $\texttt{FROM}_{10}\texttt{(FROM)}_{13}$
$\texttt{ID}_{15}\texttt{(orders)}_{20}$ $\texttt{WHERE}_{22}\texttt{(WHERE)}_{26}$ $\texttt{ID}_{28}\texttt{(s)}_{28}$ $\texttt{EQUALS}_{29}\texttt{(=)}_{29}$
$\texttt{STRING}_{30}\texttt{('x')}_{32}$ $\texttt{AND}_{34}\texttt{(AND)}_{36}$ $\texttt{TRUE}_{38}\texttt{(true)}_{41}$

EXPAND

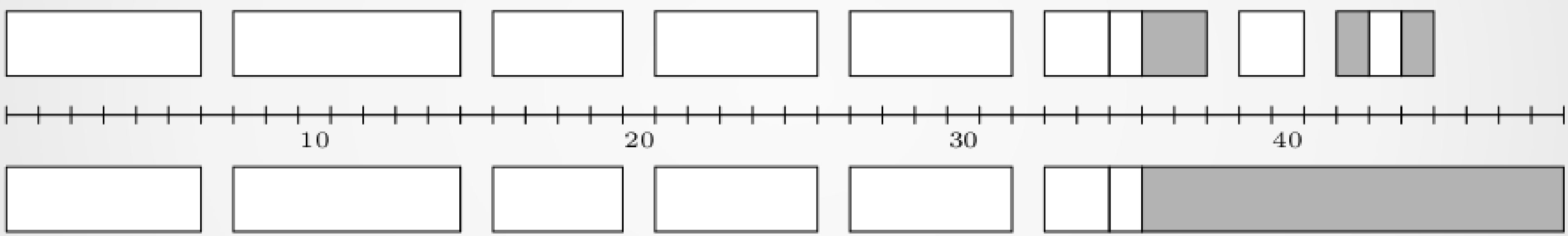
INSERT

Template Tokens:

$\texttt{SELECT}_{1}\texttt{(SELECT)}_{6}$ $\texttt{STAR}_{8}\texttt{(*)}_{8}$ $\texttt{FROM}_{10}\texttt{(FROM)}_{13}$
$\texttt{ID}_{15}\texttt{(orders)}_{20}$ $\texttt{WHERE}_{22}\texttt{(WHERE)}_{26}$ $\texttt{ID}_{28}\texttt{(s)}_{28}$ $\texttt{EQUALS}_{29}\texttt{(=)}_{29}$
$\texttt{STRING}_{30}\texttt{('')}_{32}$ $\texttt{AND}_{34}\texttt{(AND)}_{36}$

Example

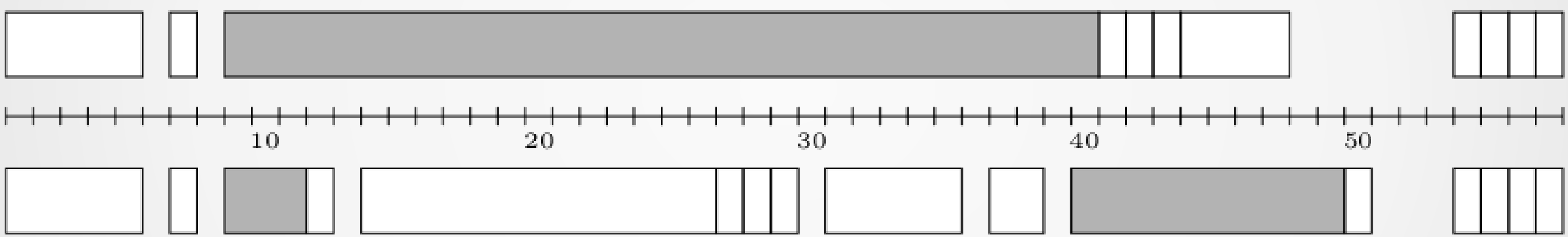
`SELECT balance FROM accts WHERE pw='' OR 1=1 --'`



`SELECT balance FROM accts WHERE pw='`$\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon$`'`

Example

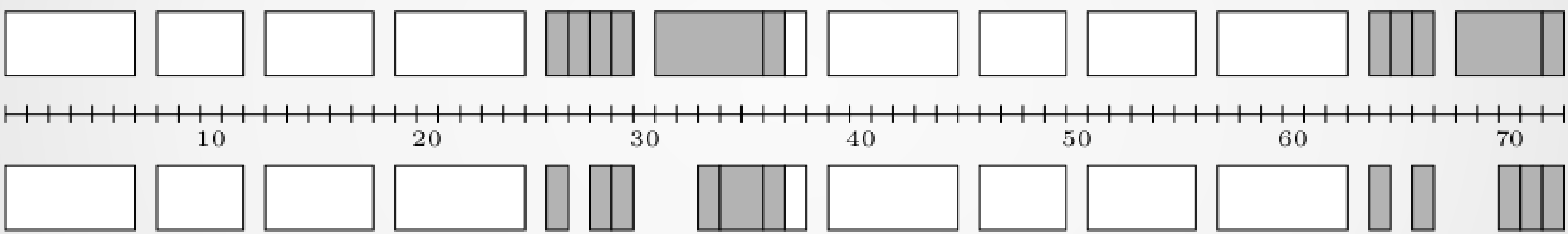
```
$data = ‘\‘; securityCheck(); $data .= ’&f=exit#’;\n f();
```




```
$data = ‘ε’; securityCheck(); $data .= ‘&f=exit#’;\n f();
```

Example

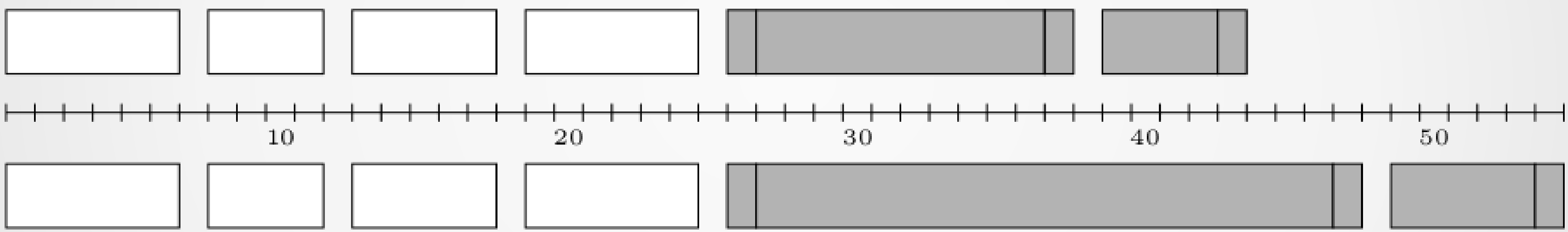
INSERT INTO trans VALUES (1,- 5E-10); INSERT INTO trans VALUES (2, 5E+5)



INSERT INTO trans VALUES (ε,- $\varepsilon\varepsilon$-10); INSERT INTO trans VALUES (ε, $\varepsilon\varepsilon$+5)

Example

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




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