Taint Propagation for Java

Vivek Haldar

Deepak Chandra

Michael Franz

{vhaldar, dchandra, franz}@uci.edu

Information and Computer Science

University of California, Irvine

Outline

Explain the threat

What we do

What others do

What we want to do

What we *really* want to do

GOAL: **Harden web applications against attacks from improperly validated input**

GOAL: Harden **web applications** against attacks from improperly validated input

GOAL: Harden

web applications

against

from

validation

HTTP interface

Plain text

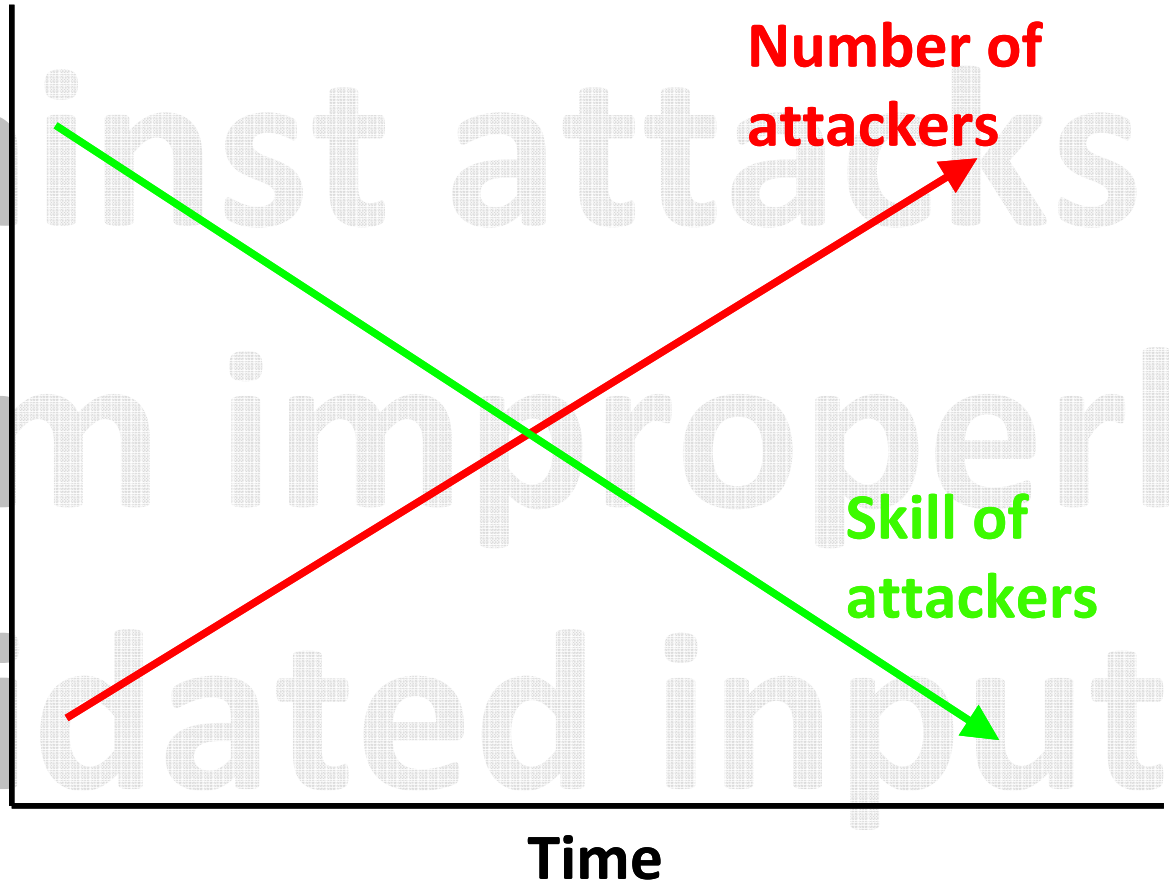
Open to the world!

GOAL: Harden web applications

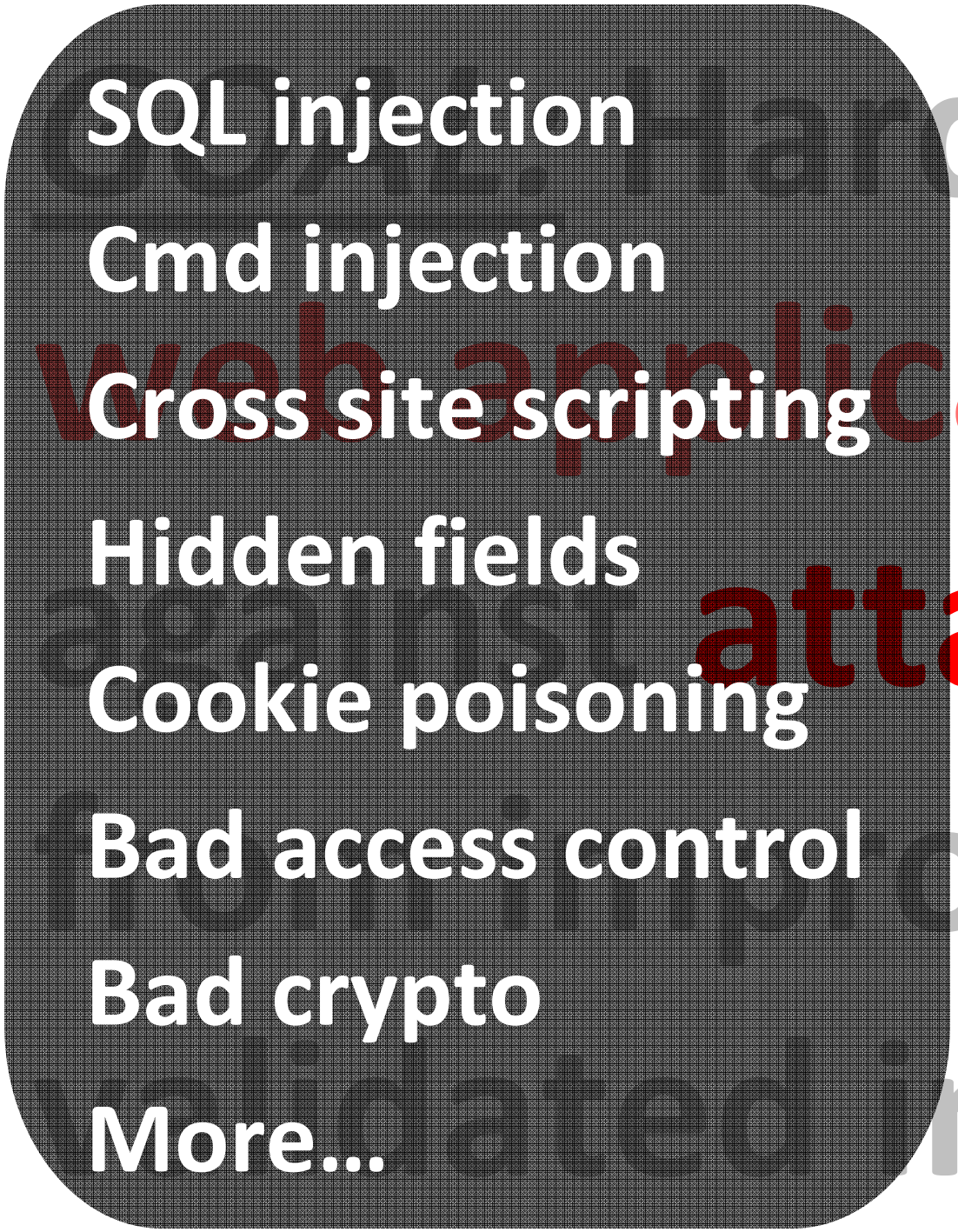
Use safe runtimes

Attacks on application logic, not failures of base language (buffer overflows)

GOAL: Harden **web applications** against attacks from improperly validated input



GOAL: Harden web applications against attacks from improperly validated input



den

ations

attacks

operly

nput

SQL injection

Cmd injection

Cross site scripting

Hidden fields

Cookie poisoning

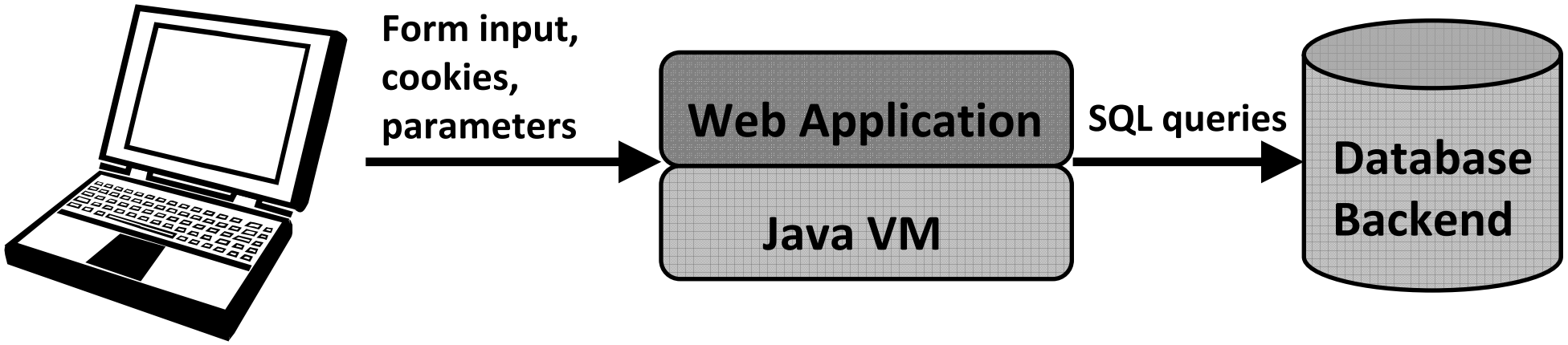
Most web application attacks from improperly validated input

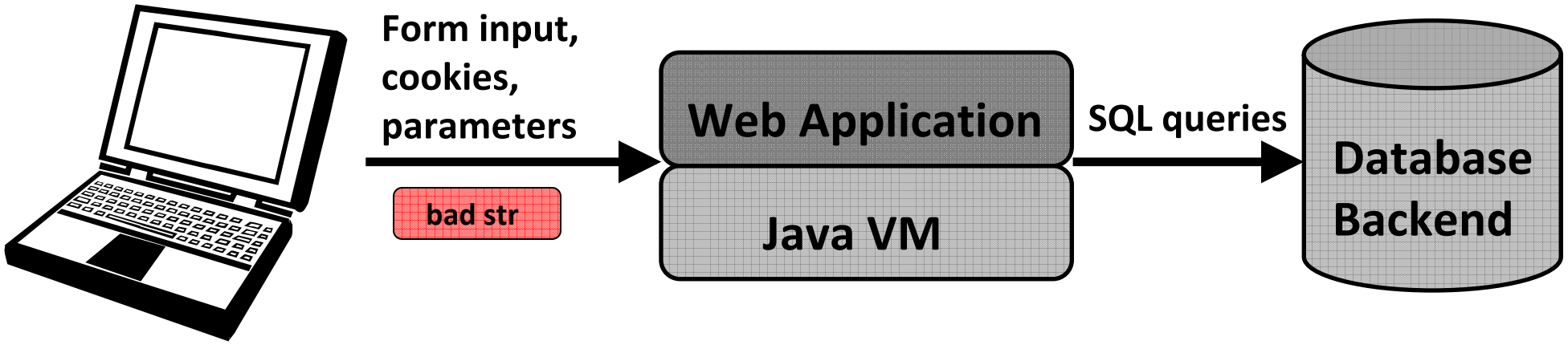
“The impact of using unvalidated input should not be underestimated. A huge number of attacks would become difficult or impossible if developers would simply validate input before using it. Unless a web application has a strong centralized mechanism for validating all input... vulnerabilities based on malicious input are very likely to exist.”

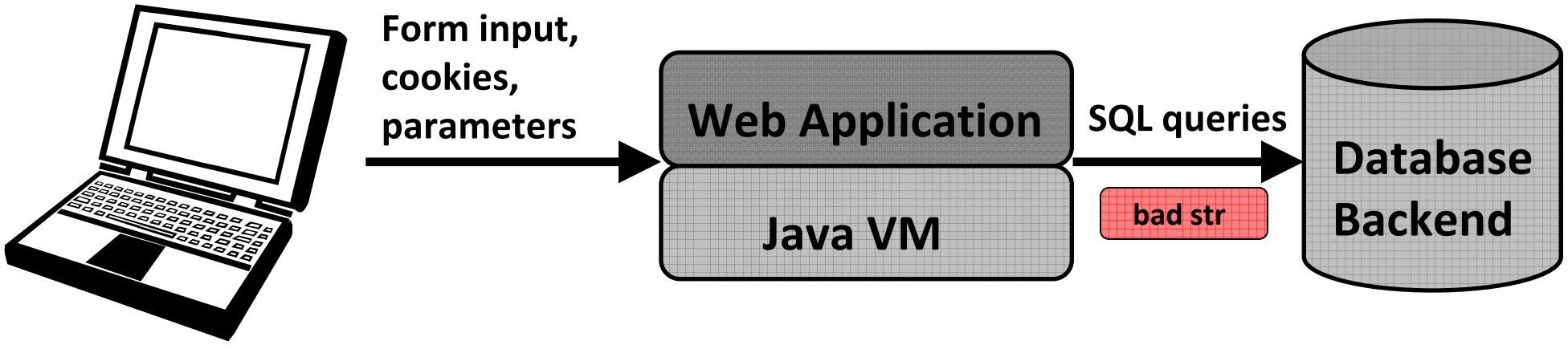
- The Ten Most Critical Web Application Security Vulnerabilities, 2004, Open Web Application Security Project.

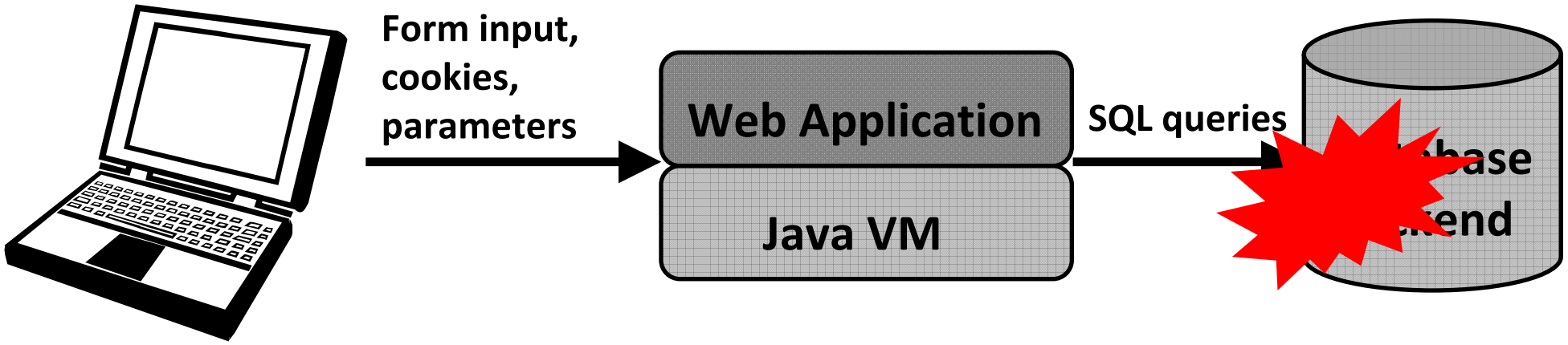
"The impact of using **unvalidated input** should not be underestimated. A **huge number of attacks would become difficult or impossible if developers would simply validate input before using it.** Unless a web application has a **strong centralized mechanism for validating all input**... vulnerabilities based on malicious input are very likely to exist."

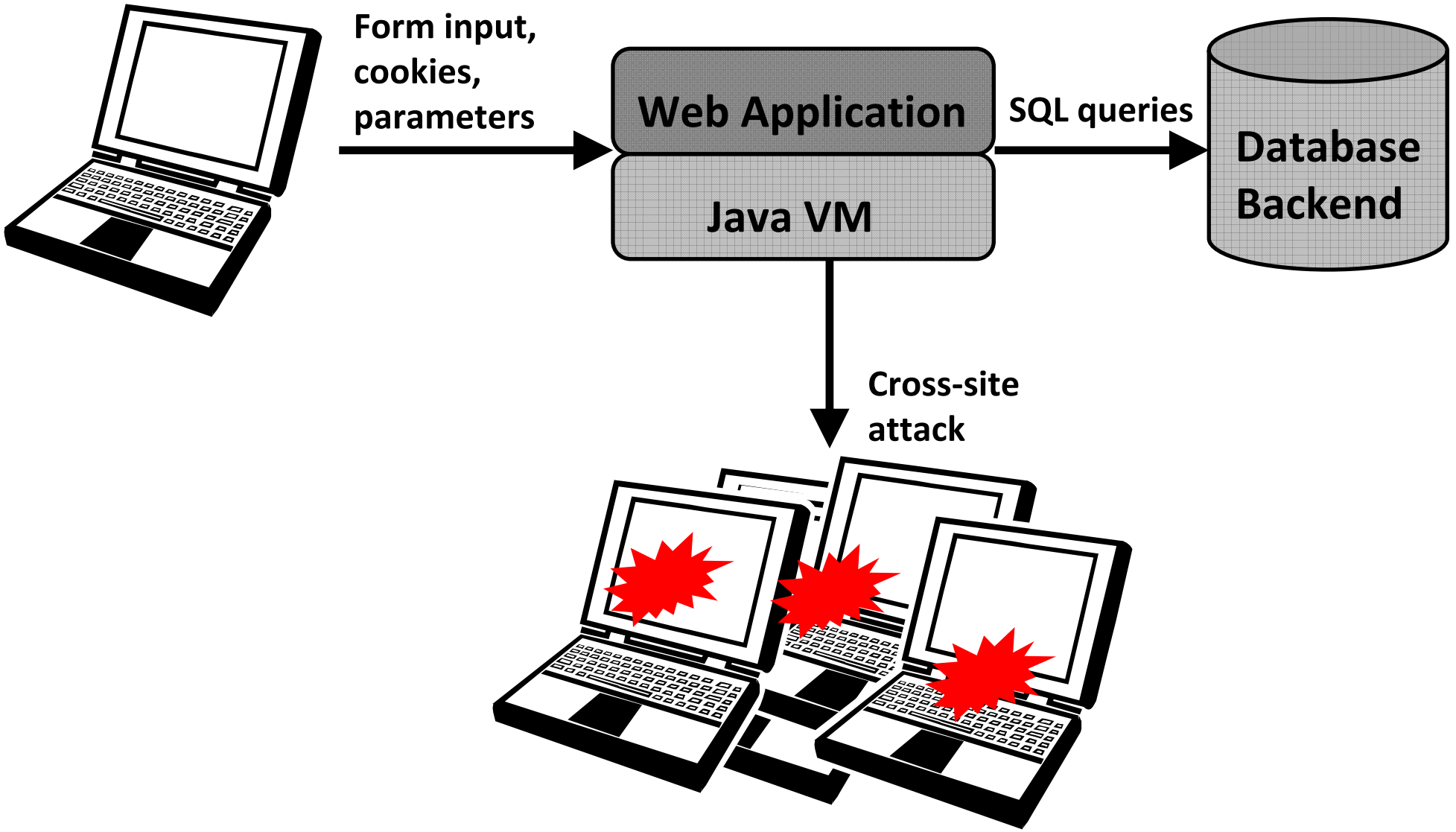
- The Ten Most Critical Web Application Security Vulnerabilities, 2004, Open Web Application Security Project.


Form input,
cookies,
parameters
Web Application
Java VM
SQL queries
Database
Backend


Form input,
cookies,
parameters
bad str
Web Application
Java VM
SQL queries
Database
Backend


Form input,
cookies,
parameters
Web Application
Java VM
SQL queries
bad str
Database
Backend


Form input,
cookies,
parameters
Web Application
Java VM
SQL queries


Form input,
cookies,
parameters
Web Application
Java VM
SQL queries
Database
Backend
Cross-site
attack

Underlying system

VM with object labels

Tracked through execution

Define policies on labels

- **Access control**
- **Information flow**

Tracking taintedness

Tainting

Un-tainting

Blocking

Tracking taintedness

Tainting

Sources

Un-tainting

Blocking

Tracking taintedness

Tainting

Un-tainting

Blocking

Sources

Any untrusted input

Form.getParameter()

Tracking taintedness

Tainting

Sources

Un-tainting

Checkers

Blocking

Tracking taintedness

Tainting

Sources

Un-tainting

Checkers

Blocking

String matching

String.matches(regex)

Tracking taintedness

Tainting

Sources

Un-tainting

Checkers

Blocking

Sinks

Tracking taintedness

Tainting

Sources

Un

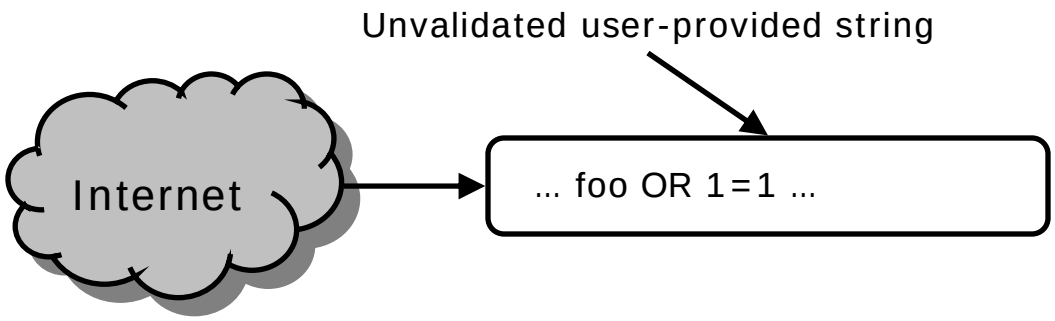
heckers

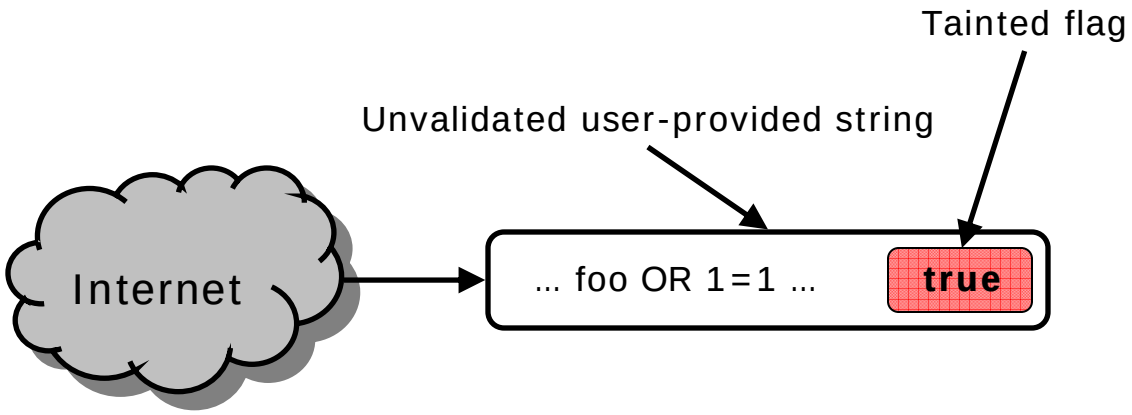
Security sensitive methods

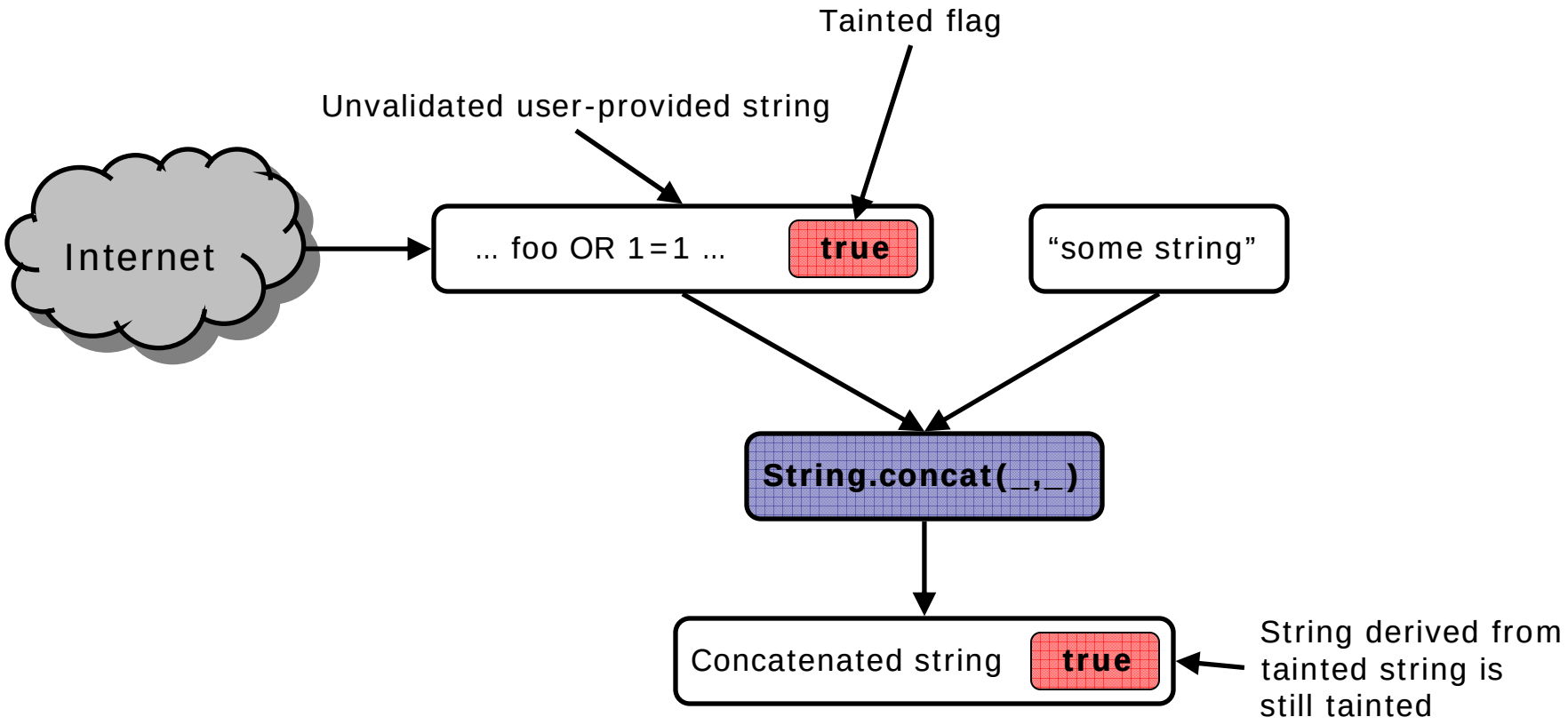
Stmt.execQuery(str)

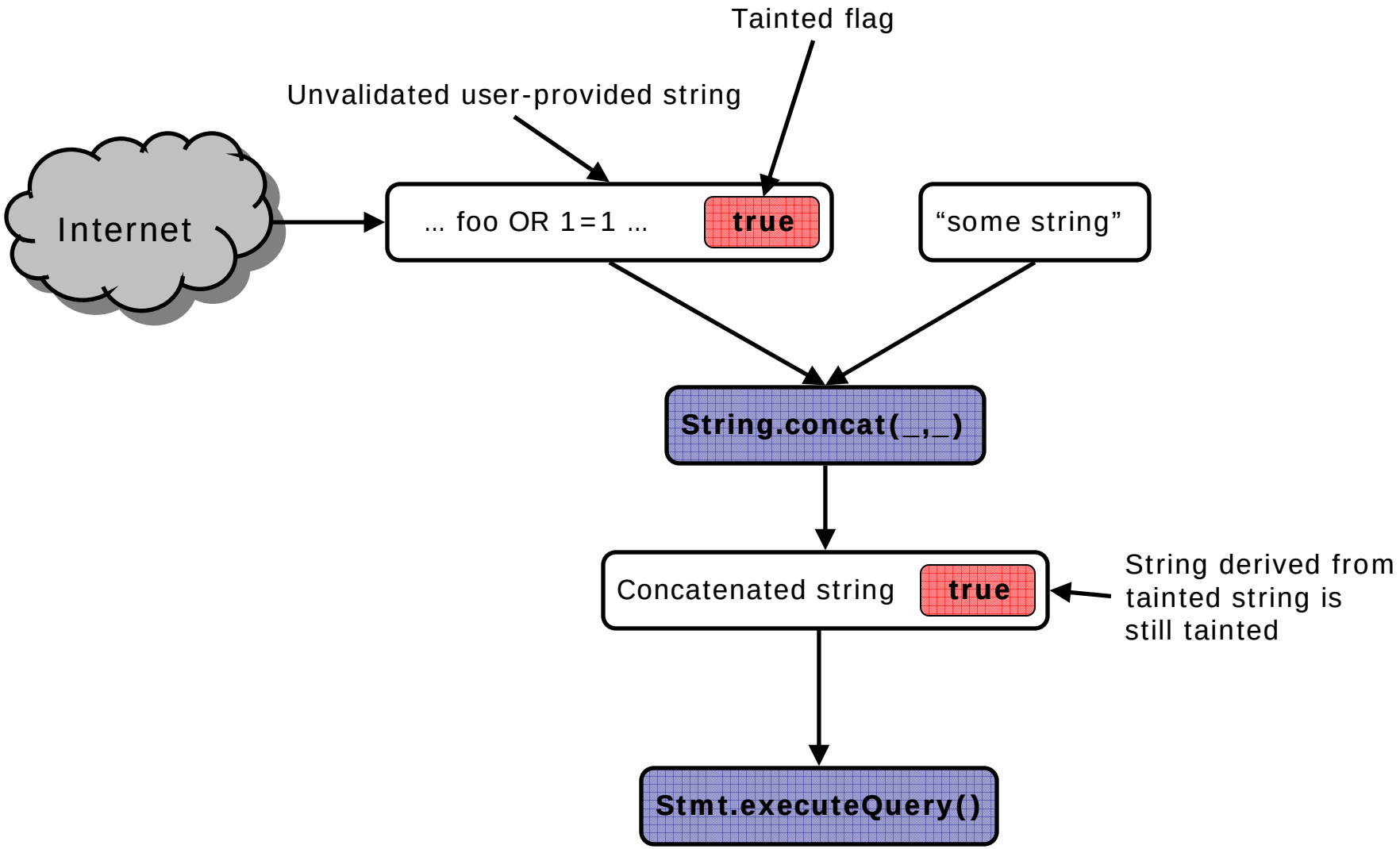
Blocking

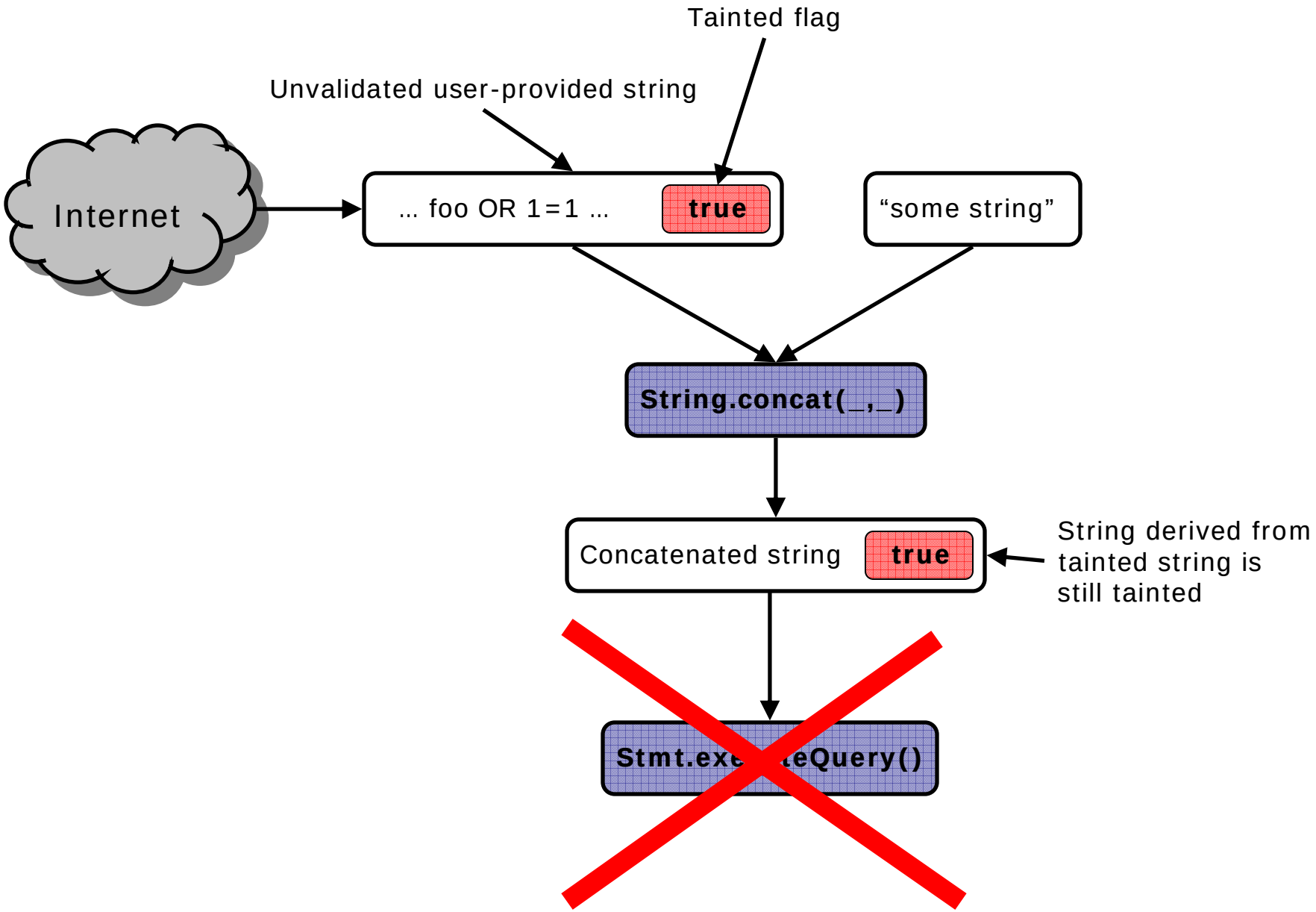
Sinks

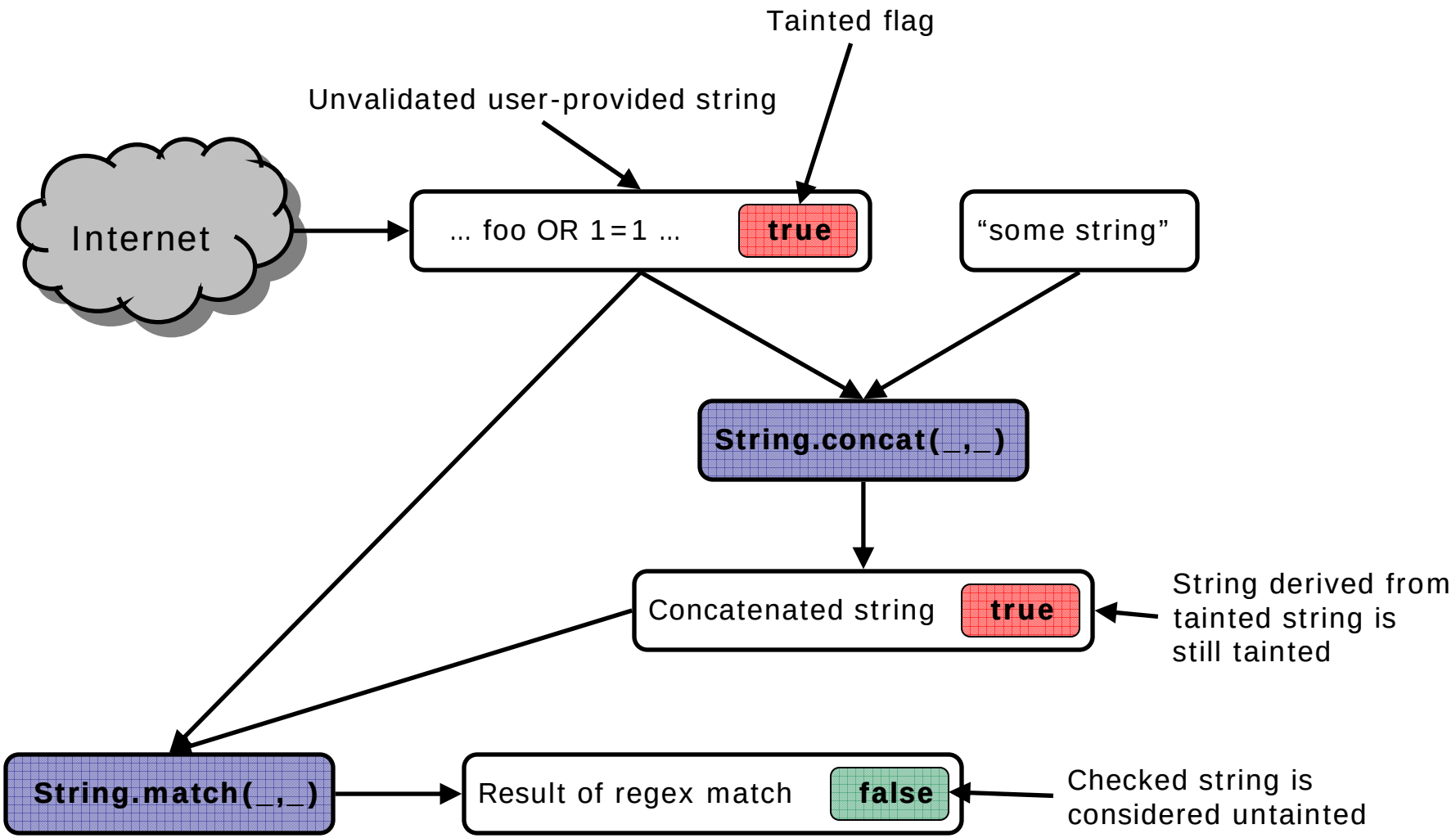

Unvalidated user-provided string
Internet
... foo OR 1=1 ...


Tainted flag
Unvalidated user-provided string
Internet
... foo OR 1=1 ...
true


Tainted flag
Unvalidated user-provided string
Internet
... foo OR 1=1 ...
true
“some string”
String.concat(_,_)
Concatenated string
true
String derived from tainted string is still tainted


Tainted flag
Unvalidated user-provided string
Internet
... foo OR 1=1 ...
true
“some string”
String.concat(_,_)
Concatenated string
true
String derived from tainted string is still tainted
Stmt.executeQuery()


Tainted flag
Unvalidated user-provided string
Internet
... foo OR 1=1 ...
true
“some string”
String.concat(_,_)
Concatenated string
true
String derived from tainted string is still tainted
Stmt.exe eQuery()

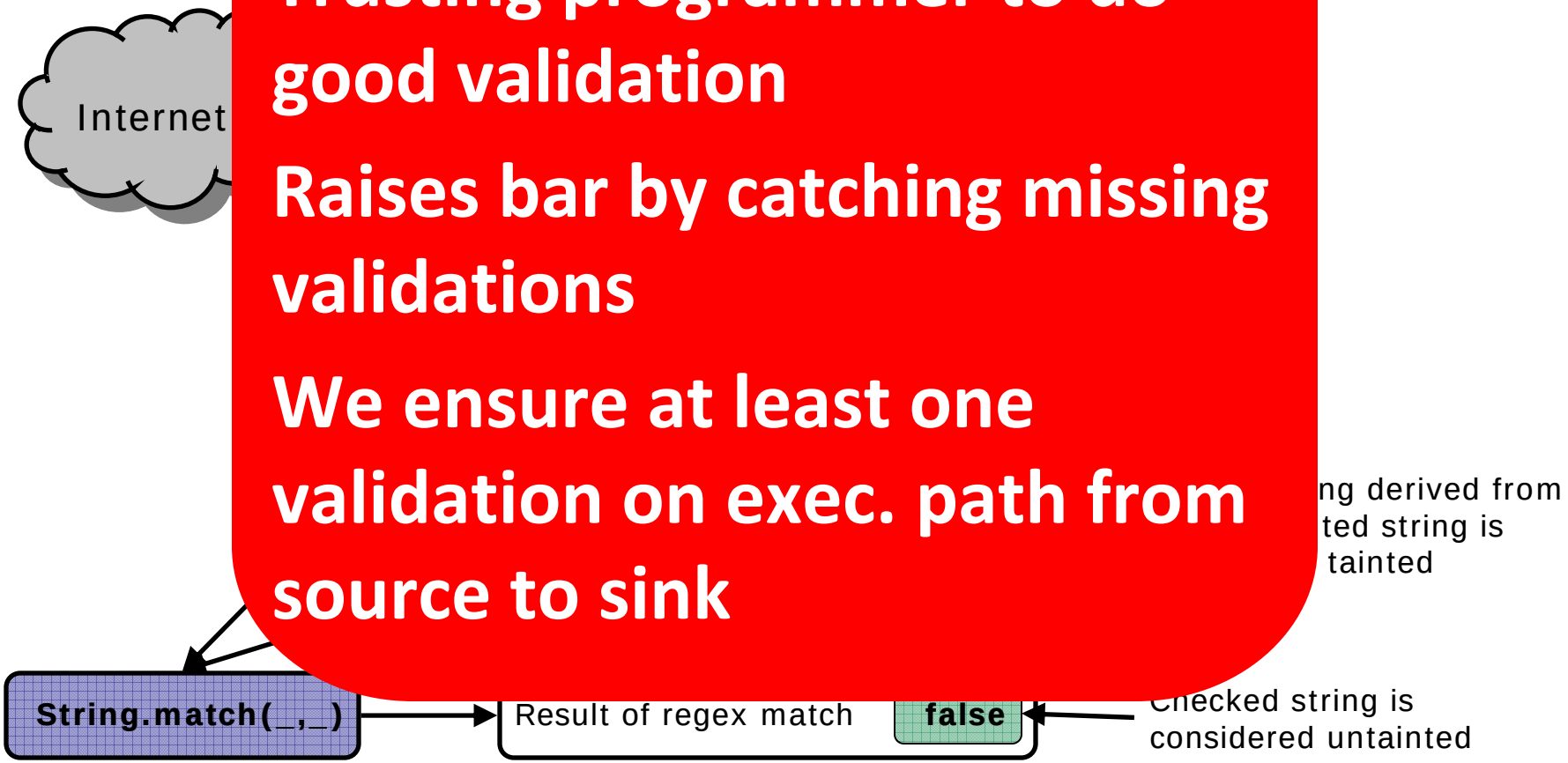

Tainted flag
Unvalidated user-provided string
Internet
... foo OR 1=1 ...
true
“some string”
String.concat(_,_)
Concatenated string
true
String derived from tainted string is still tainted
String.match(_,_)
Result of regex match
false
Checked string is considered untainted

Untainting is a *heuristic!*

Trusting programmer to do good validation

Raises bar by catching missing validations

We ensure at least one validation on exec. path from source to sink



Tracking taintedness

Application **Runtime**

Tracking taintedness

Application

Runtime

Instrument bytecode

Specify sources and sinks

Sources mark Strings tainted

Sinks must not consume tainted data

Tracking taintedness

Application

Instrument bytecode

Specify sources and sinks

Sources mark Strings tainted

Sinks must not consume tainted data

Runtime

String, StringBuilder, StringBuffer classes

String *manipulation* preserves and propagates taintedness

String *checking* removes taintedness

Tracking taintedness

Application

Instr

Speci

Sourc

tainte

Sinks mi

tainted data

Runtime

ates

String *checking* removes taintedness

Test case: WebGoat

Successfully catches errors

Tracking taintedness

Application

Instrumen

Specify sou

Sources ma
tainted

Sinks must n
tainted data

Runtime

gBuilder,
r classes

pulation
nd propagates
s

String *checking* removes
taintedness

Performance overhead negligible

Solution space

Static

Dynamic

Solution space

Static

Dynamic

[Wagner, Lam, Livshits, Evans…]

Completeness

No runtime overhead

(Most) need source

Dynamic class loading?

False positives

Solution space

Static

Dynamic

[Wagner, Lam, Livshits, Evans…]

Completeness

No runtime overhead

(Most) need source

Dynamic class loading?

False positives

[Perl taint mode, Ruby…]

Can apply to deployed systems with bugs!

Solution space

Static

Dynamic

[Wagner, Lam, Livshits, Evans…]

Completeness

No runtime overhead

(Most) need source

Dynamic class loading?

False positives

[Perl taint mode, Ruby…]

Can apply to deployed systems with bugs!

Future Work

Log attacks

carry more than a “tainted” flag

Generalize tainting

multiple taint levels

Finer-grained tainting?

at character level?

Future Work

Log attacks

carry more than a "tainted" flag

Generali

multiple

Finer-gr

at chara

String datatype is more than series of chars – need more metadata

OUTLOOK:

Insecurity

=

Failure of abstraction

OUTLOOK:

Buffer overflows

= Insecurity

=

Failure of abstraction

OUTLOOK:

Buffer overflows

=

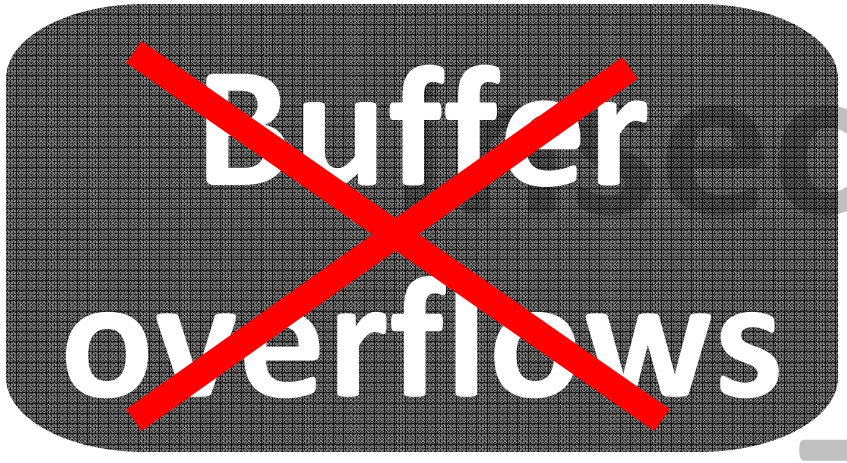
Type unsafe C

Insecurity

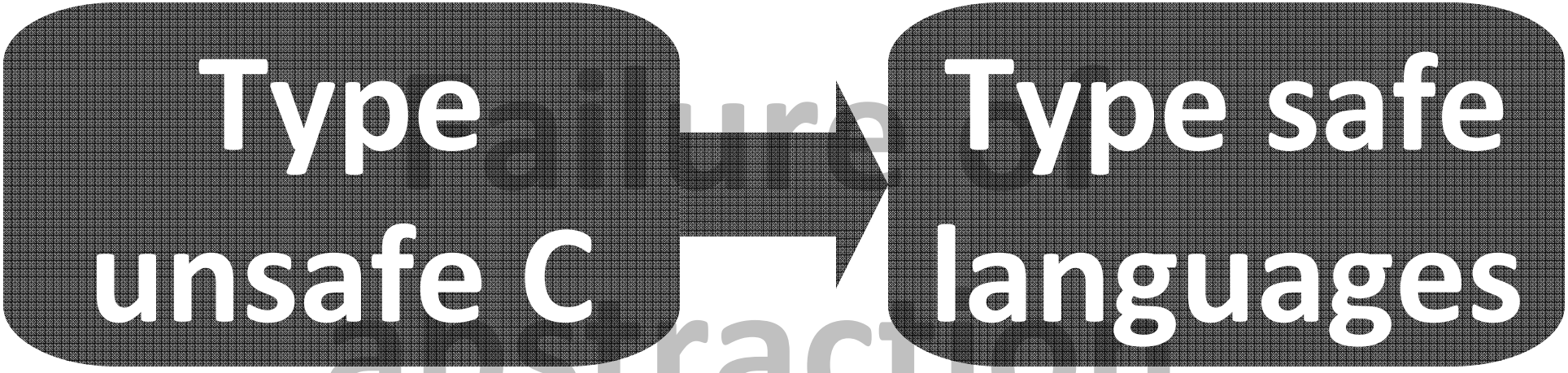
=

failure of abstraction

OUTLOOK:



=



OUTLOOK:

Input validation

Insecurity

=

What abstraction?

Failure

abstraction

Summary

Unvalidated input is the most common attack on webapps

Dynamic method significantly raises bar for exploits

Catches real errors

Negligible performance hit

Summary

Unvalidated input is the most common attack on webapps

Dynamic method significantly raises bar for exploits

Catches real errors

Negligible performance hit

Catch me later for a demo

THANK YOU

vhaldar@uci.edu
http://www.ics.uci.edu/~vhaldar