A Taint Mode for Python via a Library

Juan José Conti
jjconti@gmail.com | FRSF UTN
Alejandro Russo
russo@chalmers.se | Chalmers

OWASP AppSec Research 2010
Stockholm, Sweden - June 21-24

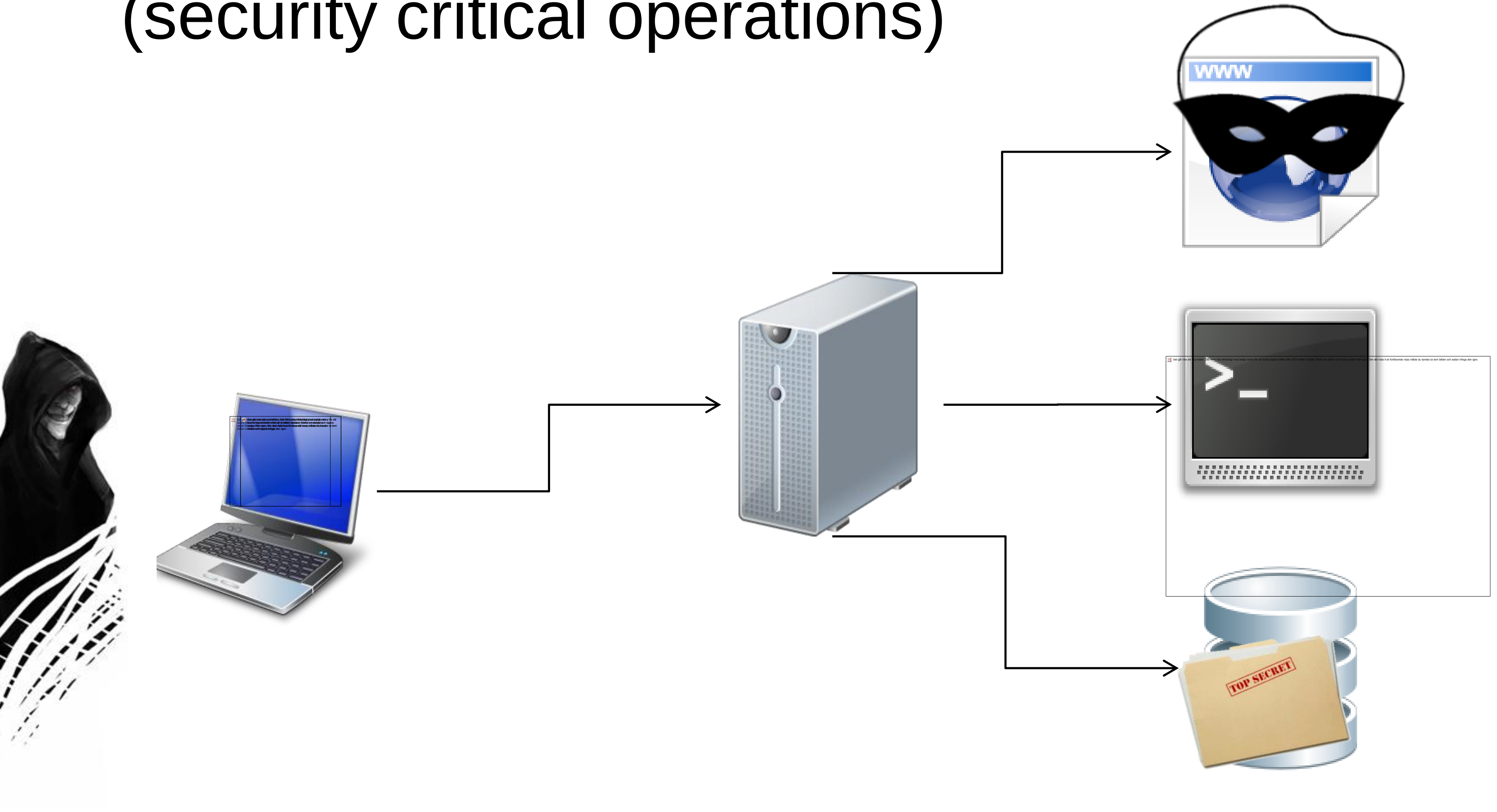
OWASP TOP 10

* A1: Injection
* A2: Cross-Site Scripting (XSS)
* A3: Broken Authentication and Session Management
* A4: Insecure Direct Object References
* A5: Cross-Site Request Forgery (CSRF)
* A6: Security Misconfiguration
* A7: Insecure Cryptographic Storage
* A8: Failure to Restrict URL Access
* A9: Insufficient Transport Layer Protection
* A10: Unvalidated Redirects and Forwards

Attackers goal: **craft input data** to gain some **control** over certain **operations**

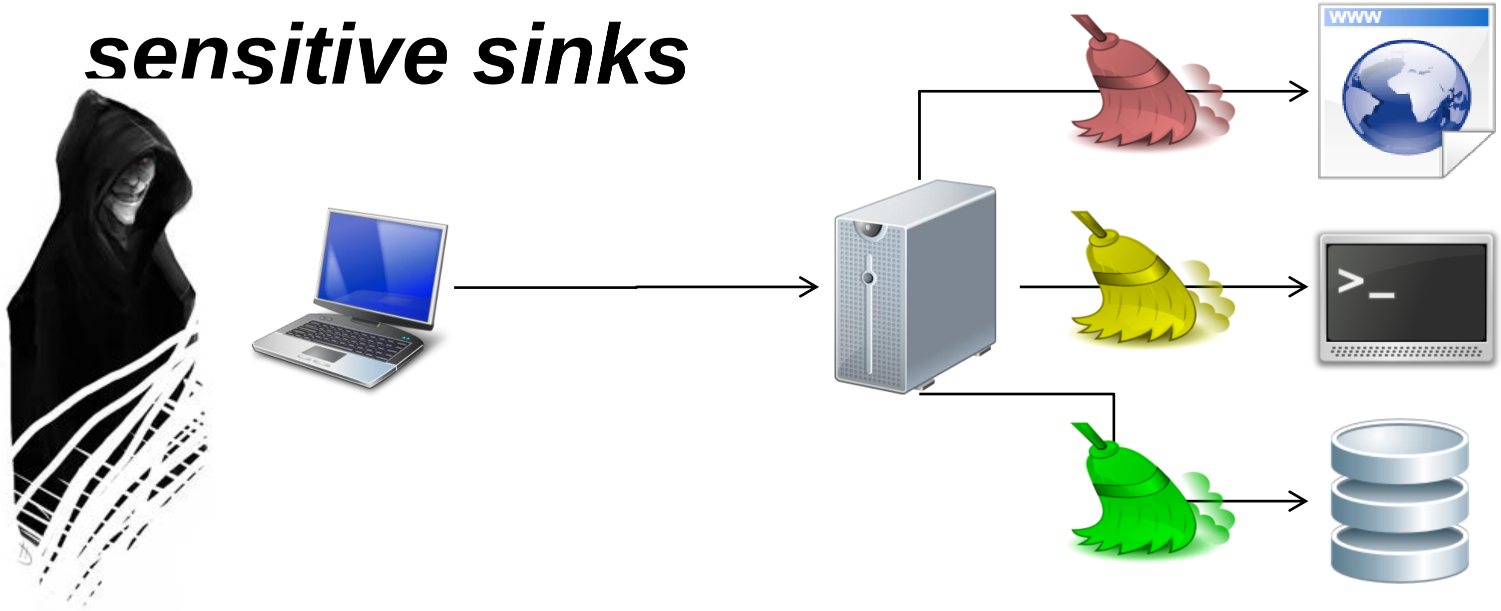
Scenarios

- Web applications with sensitive sinks (security critical operations)



Security Policy

- Data received from a client is considerer **untrustworthy** (or *tainted*)
- Untrustworthy data can be made trustworthy (or *untainted*) by a ***sanitization process***
- *Untrustworthy data (or tainted) can't reach* ***sensitive sinks***

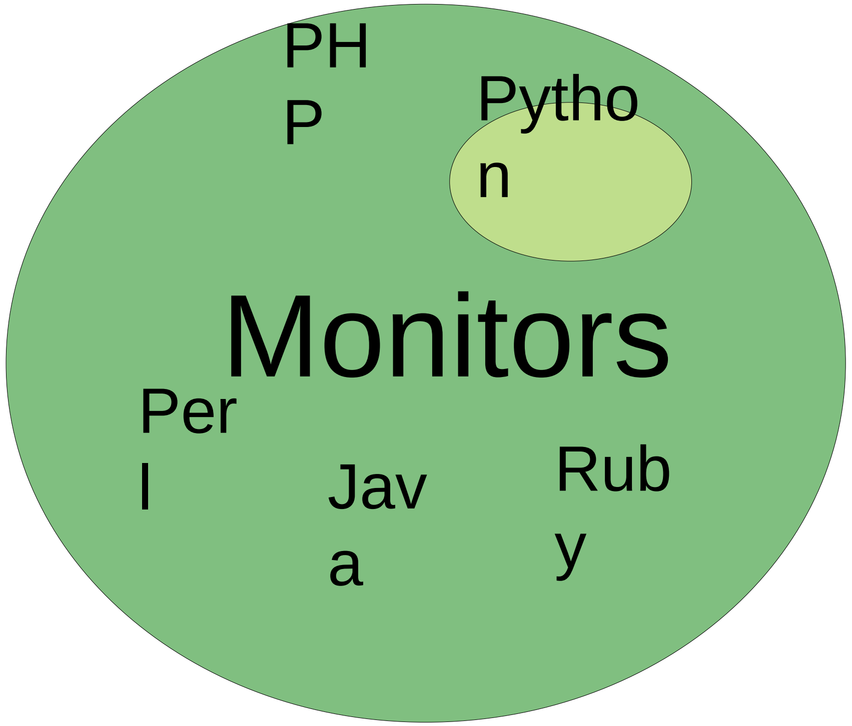


Different kind of attacks

“42 or 1=1”

“<script> alert('hello') </script>”

Other taint analysis



Closest related work [18]

- Modify interpreter
- Only strings
- Binary tainted attribute

+ NO changes in code

+ Less false alarm than SA

- Overhead
- Modification of the interpreter

Taint analysis

- Mark untrusted inputs, sanitizations functions and sensitive sinks.
- **Untainting** data when sanitized
- **Detect** when tainted data reaches sensitive sinks
- **Propagate** taint information

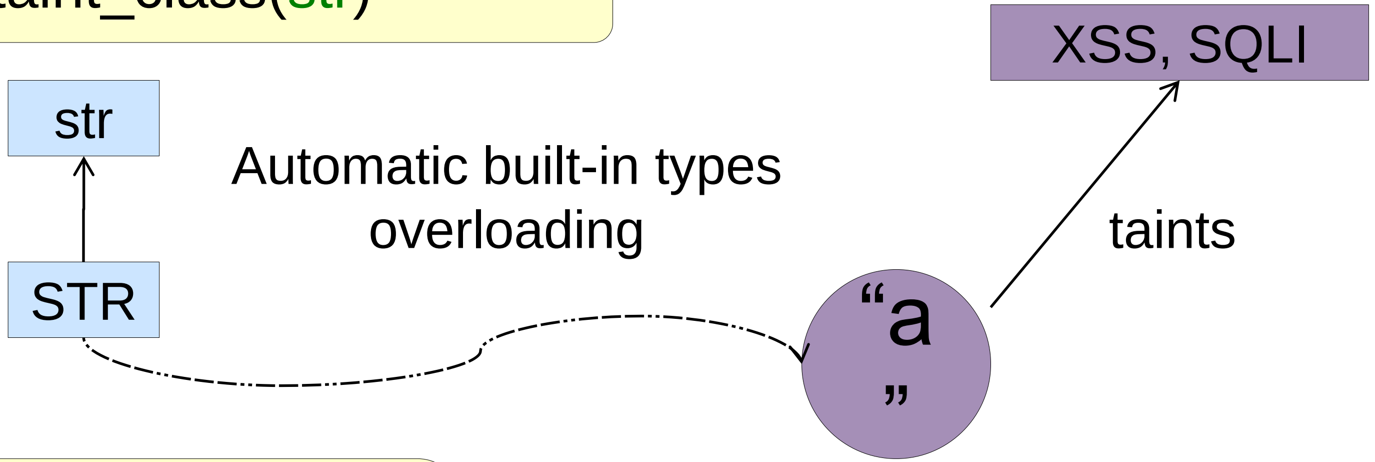
Taint Propagation

```
a # tainted
b # clean
c = a + b # now c is tainted too


a * 8
a[3:10]
“is %s clean?” % a
a.upper()
```

How does the library work?

```
STR = taint_class(str)
```



```
c   =  a  +  b
STR = STR + str
STR = STR.__add__
```

```
c   =  a.upper()
STR = STR.upper
```

Automatic built-in functions overloading

```
len = propagate_func(len)
c = len(a)
INT = len(STR)
```

Taint Mode in Python (API)

- Untrusted sources

```
from web import input
input = untrusted(input)
```

```
@untrusted
def user_function():
    ...
```

Taint Mode in Python (API)

- Sensitive sinks

```
db.select = ssink(SQLI)(db.select)
```

```
@ssink(OSI)
def user_function(cmd):
    ...
```

Taint Mode in Python (API)

- Sanitization functions

```
sanitize = cleaner(SQLI)(sanitize)
```

```
@cleaner(OSI)
def user_function(cmd):
    ...
```

Little demo

(using web.py)

Conclusions and future works

- It is possible to provide a light-weight (300 LOC) taint analysis lib for Python
- No need to modify the interpreter
- Is it possible to do a similar module for other languages? **Ruby?**
- Evaluation on popular web applications
  - Integrate our library into **Google App Engine** and web frameworks

More information

A Taint Mode for Python via a Library

Juan José Conti and Alejandro Russo

OWASP AppSec Research 2010

Stockholm, Sweden - June 21-24

http://www.cse.chalmers.se/~russo/juanjo.htm

http://www.juanjoconti.com.ar/taint/