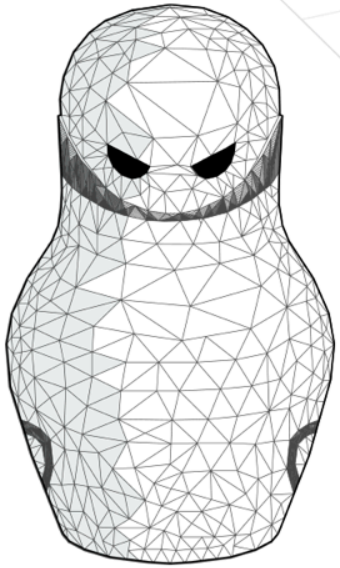


ZERO NIGHTS 2015

Mathematical model of input validation vulnerabilities and attacks

Ivan Novikov, Wallarm

About

@d0znpp (twitter, facebook, etc.)

Web application security researcher

Bug hunter

Author of “SSRF bible” (http://bit.ly/SSRF_bible)

(https://wallarm.com)

A problem of terminology

CWE

WASC

OWASP

...

X: I found **XSS**!
Y: **HTML** injection?
X: !!!!%^@^#%^%@!!

X: I found **XXE**!
Y: **XML** injection?
X: !!!!%^@^#%^%@!!

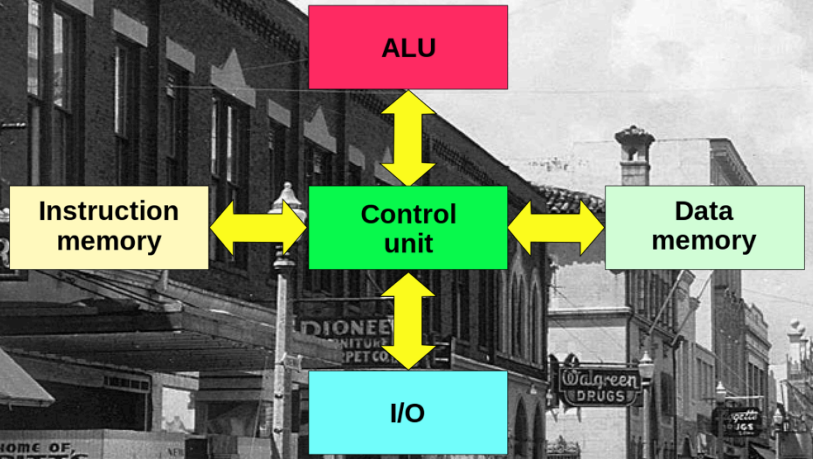
X: I found **CRLF** injection!
Y: Can you exploit it as splitting for XSS injection? Or just **open-redirect**?
X: !!!!%^@^#%^%@!!

1930s

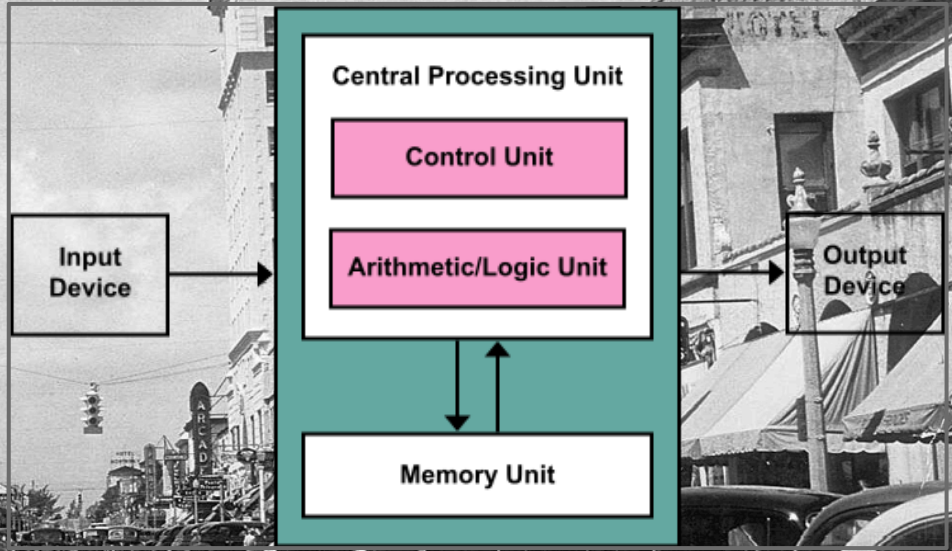

LADBROKE GVE
WESTBOURNE GVE
PADDINGTON STN
MARBLE ARCH
REGENT ST STRAND
CANNON ST BANK
ALDGATE BARKING
PICTORIAL
ALWAYS BEST
15A
GENERAL
UV 6758
RX 7014
RA 8910


HARLEY-DAVIDSON
MOTORCYCLES
BICYCLES

1940s



Harvard



von Neumann (Princeton)

“Common bus” in a client-server model

> SELECT id FROM users WHERE login=’admin’ AND password=’123456’

< OK

> SELECT id FROM users WHERE login=? AND password=?

> admin

> 123456

< OK

“Common bus” in a client-server model

> SELECT id FROM users WHERE login=’admin’ AND password=’123456’

< OK

> SELECT id FROM users WHERE login=? AND password=?

> admin

> 123456

< OK

Which conclusions?

Data and instructions

How to systemize it?

As always, mathematics can help us!

$$K_i \cap K_j = \emptyset, \forall i \neq j,$$
$$K_1 \cup \ldots \cup K_N = T.$$

$$C \cap V = \emptyset,$$
$$C \cup V = T.$$

$$O = \{o \in \mathcal{P}(T) : (o_i \cap C) = (o_j \cap C), \forall i \neq j\} \quad (3)$$

$T = \{t\}$

- C

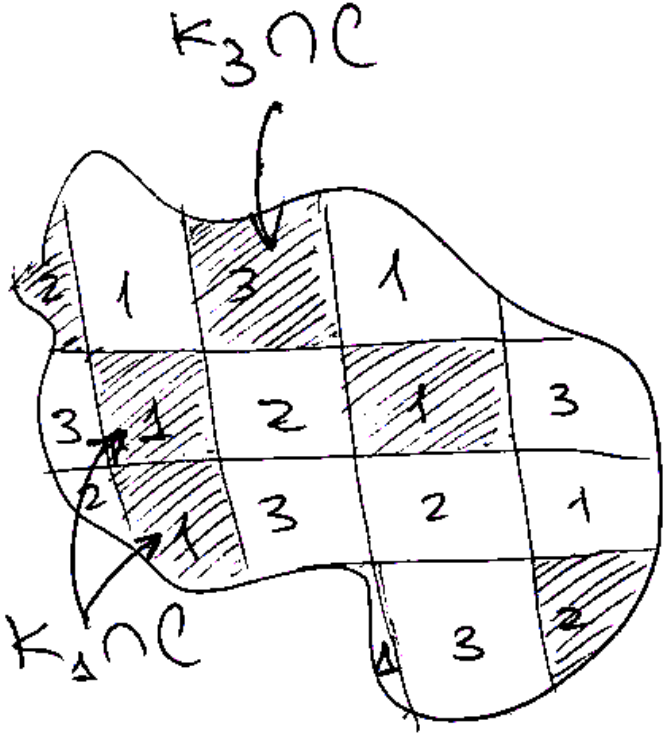
- V

$K_1 \cap K_2 =$

$= K_2 \cap K_3 =$

$= K_3 \cap K_1 = \varnothing$

$C \cap V = \varnothing$



$T = \{t\}$

- C

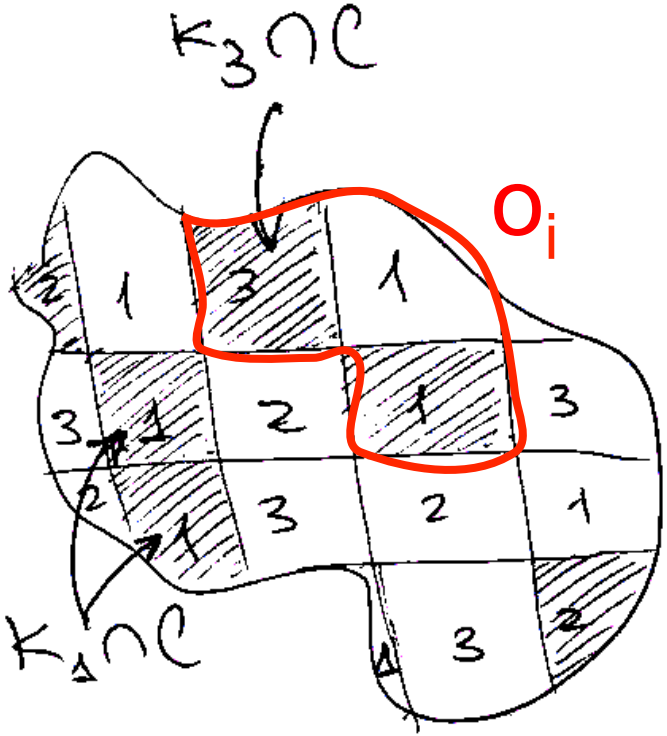
- V

$K_1 \cap K_2 =$

$= K_2 \cap K_3 =$

$= K_3 \cap K_1 = \emptyset$

$C \cap V = \emptyset$



$T = \{t\}$

- C

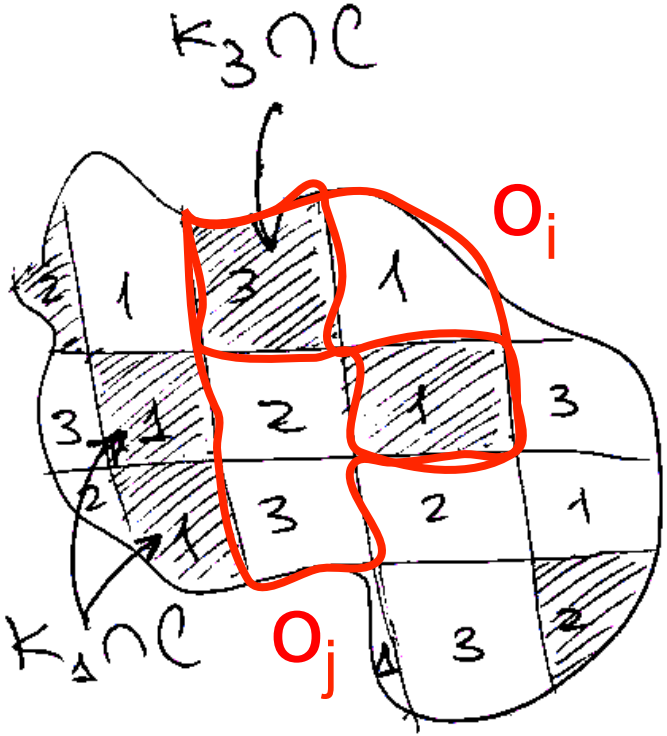
- V

$K_1 \cap K_2 =$

$= K_2 \cap K_3 =$

$= K_3 \cap K_1 = \emptyset$

$C \cap V = \emptyset$



V-objects of K_j type **injection** for O definition

$$\begin{cases} \exists o_i \in O = \{t\} : \forall t \in V, t \notin K_j, \\ \exists o_j \in O = \{t\} : \exists t \in V, t \in K_j. \end{cases}$$

Exists o in O have V-objects not in K_j-type AND

Exists another o in O have V-objects in K_j-type

V-objects of K_j type **execution** for O definition

$$\forall o_i \in O = \{t\} : \exists t \in V, t \in K_j.$$

All o in O have V-objects in K_j-type

Implementations

Turing machine

Formal grammars

Logic flows

DNA

Viruses

...

Example #1. XML instructions execution

POST /xmlrpc HTTP/1.1

…

<?xml version=”1.0”?>

<!DOCTYPE [<!ENTITY % a SYSTEM “http://...”> %a; %o; %l;]>

<a>test

Example #1. XML external entities injection

```
POST /xmlrpc HTTP/1.1

…

<?xml version=”1.0”?>

<!DOCTYPE [<!ENTITY % a SYSTEM “http://...”> %a; %o; %l;]>

<a>test</a>
```

And what? What does it do in practice?

No precedents (regexps, fingerprints) anymore for:

- IDS/IPS (such as WAF)
- Fuzzers
- DAST
- ...

First steps

libinjection — open source library to detect SQL injection based on tokenizer

- Only first 5 tokens
- Token fingerprints for all known attacks (about 9k)
- Hardcoded tokenizer — hard to maintain

It seems to be a good idea to mark each token as “**data**” or “**instruction**”.
As a result, it’ll be possible to detect attacks without signatures!

Parser/tokenizer bugs provides bypasses anyway

http://pastebin.com/8i40qQAx

http://pastebin.com/Wy0fhegr

http://pastebin.com/7zd51x0h

```
'id=1 union distinctROW select 1 from users&type=fingerprints
'id=snoopdogg 1.e.``1.e.id union select 1 from users&type=fingerprints
'id=``1.e.id union select 1 from users&type=fingerprints
'id={f`id`} union select 1 from users&type=fingerprints
'id=1 union select!<1,1 from users&type=fingerprints
'id=1 union select@ $,1 from users &type=fingerprints
'id=1 union select!.a.a,1 from (select 1 a from users limit 1)a &type=fing
'id=1 union select*,1 from users &type=fingerprints
'id=1 union select*,1 from (select 1 from users limit 1)p join (select 2)b
```

NEW! Bypasses for mod_security, libinjection and other WAFs
by @sergey_lakantar , @lightos , @d0znpp , @NGalbreath , @black2fan

And finally... More syntaxes!

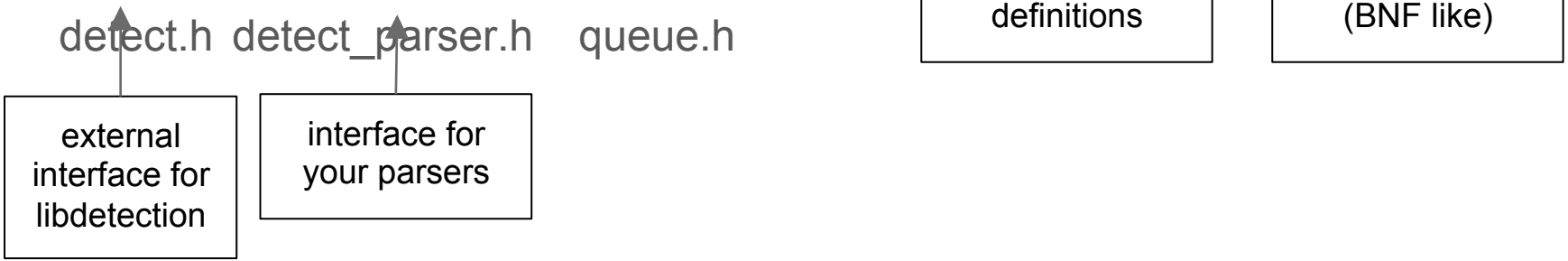
libdetection PoC is available at https://github.com/wallarm now

./lib/sqli:

CMakeLists.txt sqli.c sqli.h sqli_lexer.re2c
sqli_parser.y

./include/:

detect.h detect_parser.h queue.h



Some tests

/dev/random strings (average length 255 bytes)

i7-4710HQ (1 core used)

	libinjection	*wallarm PoC (libdetection)*
/dev/random (255b aver.len)	391k/s	**953k/s (+243%)**
libinjection ./data/* attacks	**530k/s**	**539k/s (~ the same)**
AAA...x1024	182k/s	**200k/s (+9%)**

Just PoC. Grammar and lexer are simple!

Thanks!

@wallarm, @d0znpp

https://github.com/wallarm