Taint Analysis For Vulnerability Discovery

passket # gmail.com
http://passket.tistory.com

2010. 7. 3

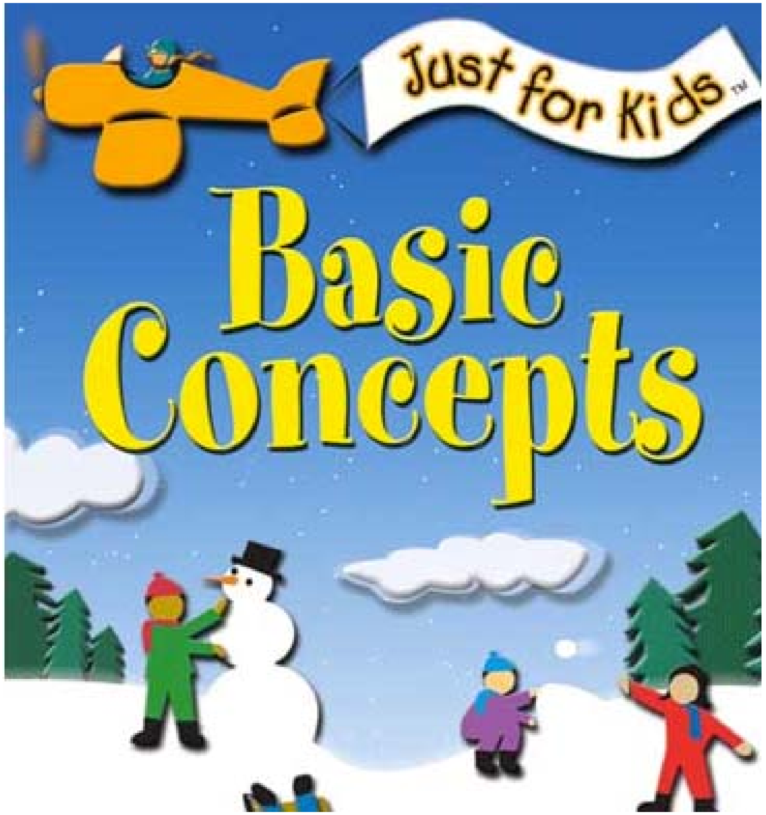
Motivation

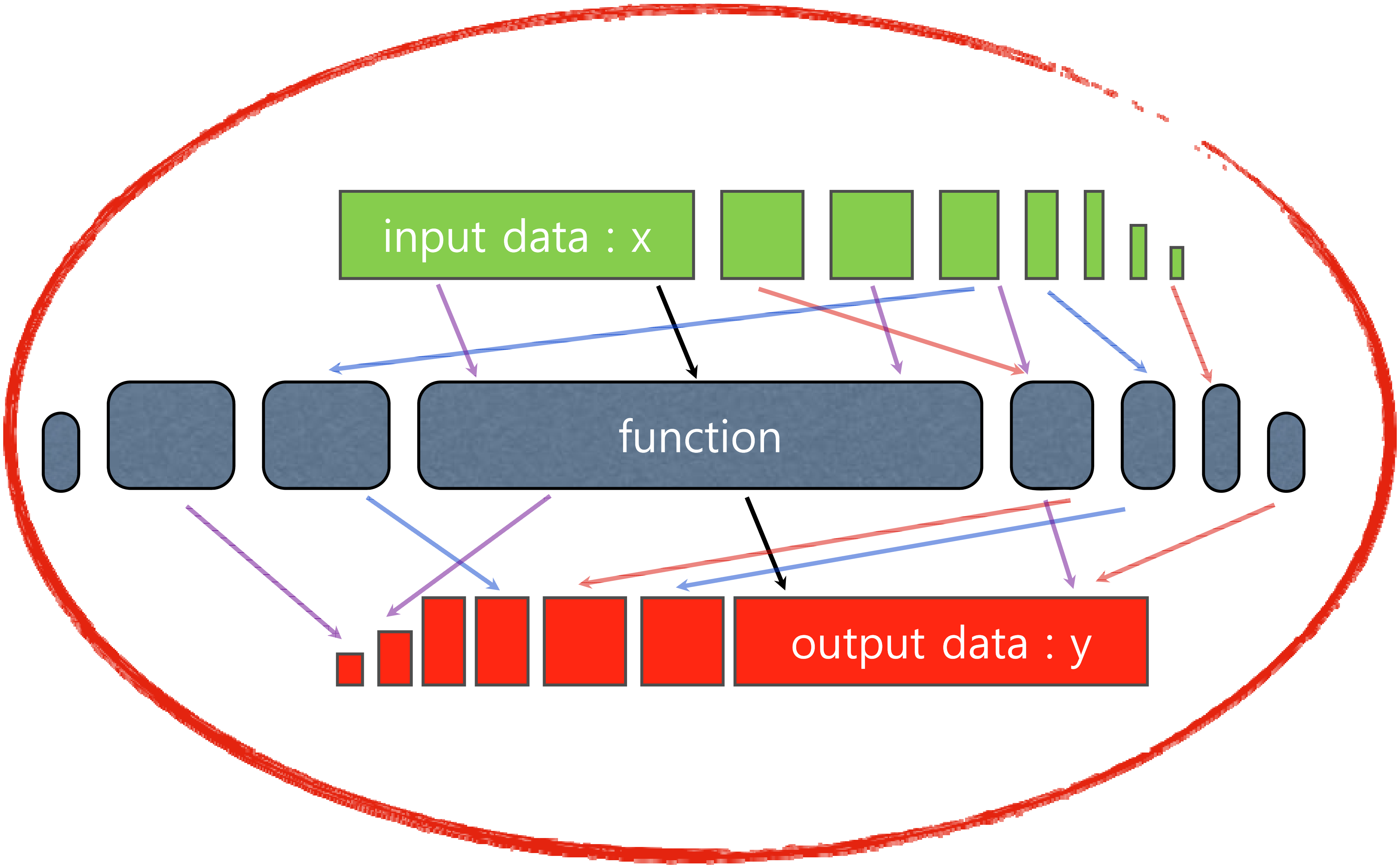
Where are vulnerabilities ?
How can you find the vulnerability ?
Is there vulnerability in my program ?
Where is my data in vulnerable program ?Unknown
vulnerability in commodity program ?
Does finding zero-day-vulnerablity make money ?

Outline

- Basic Concepts
- Tainted Propagation on x86
- Simple Test for Tainting
- Into The Abyss : in the wild world
- Future Work : Raison Framework
- References

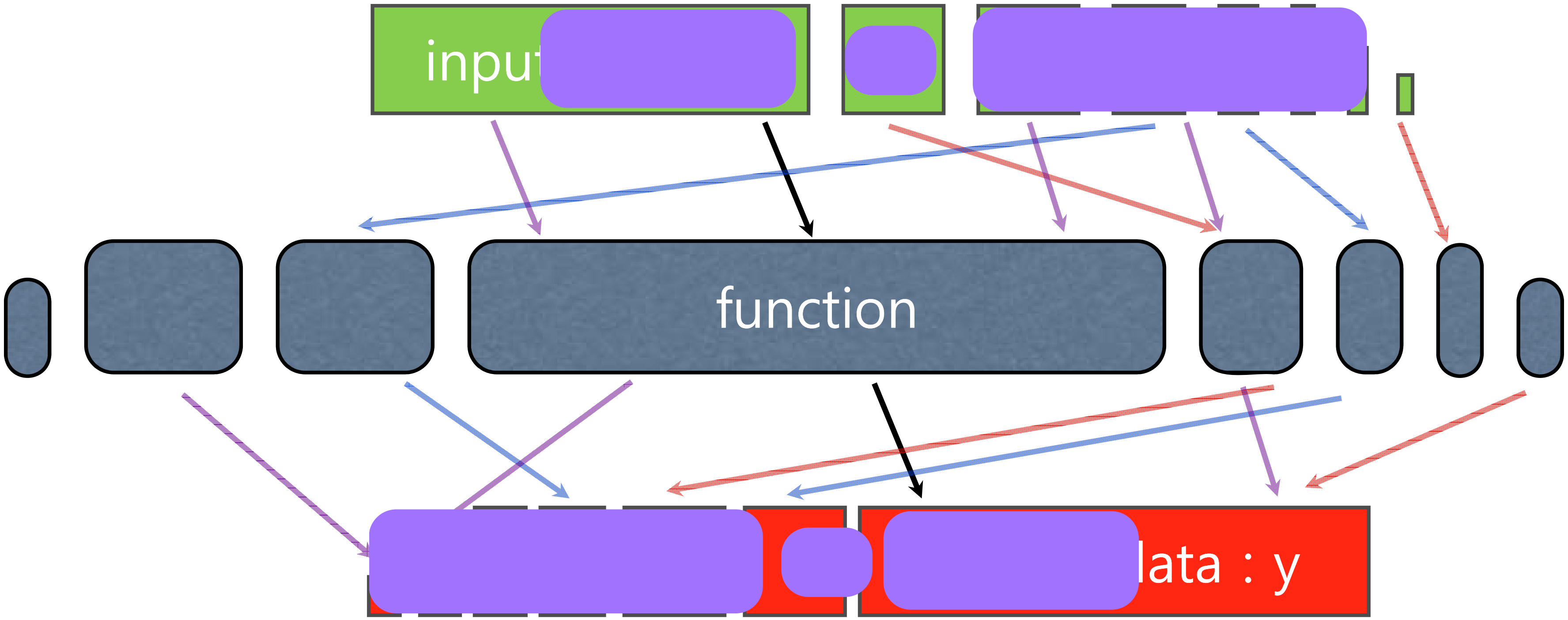
Basic Concepts





And Now we call this “system”

modify data : we call this “tainting”



we can analysis how tainted output driven
: we can call this “taint analysis”

How does taint analysis help Our works ?

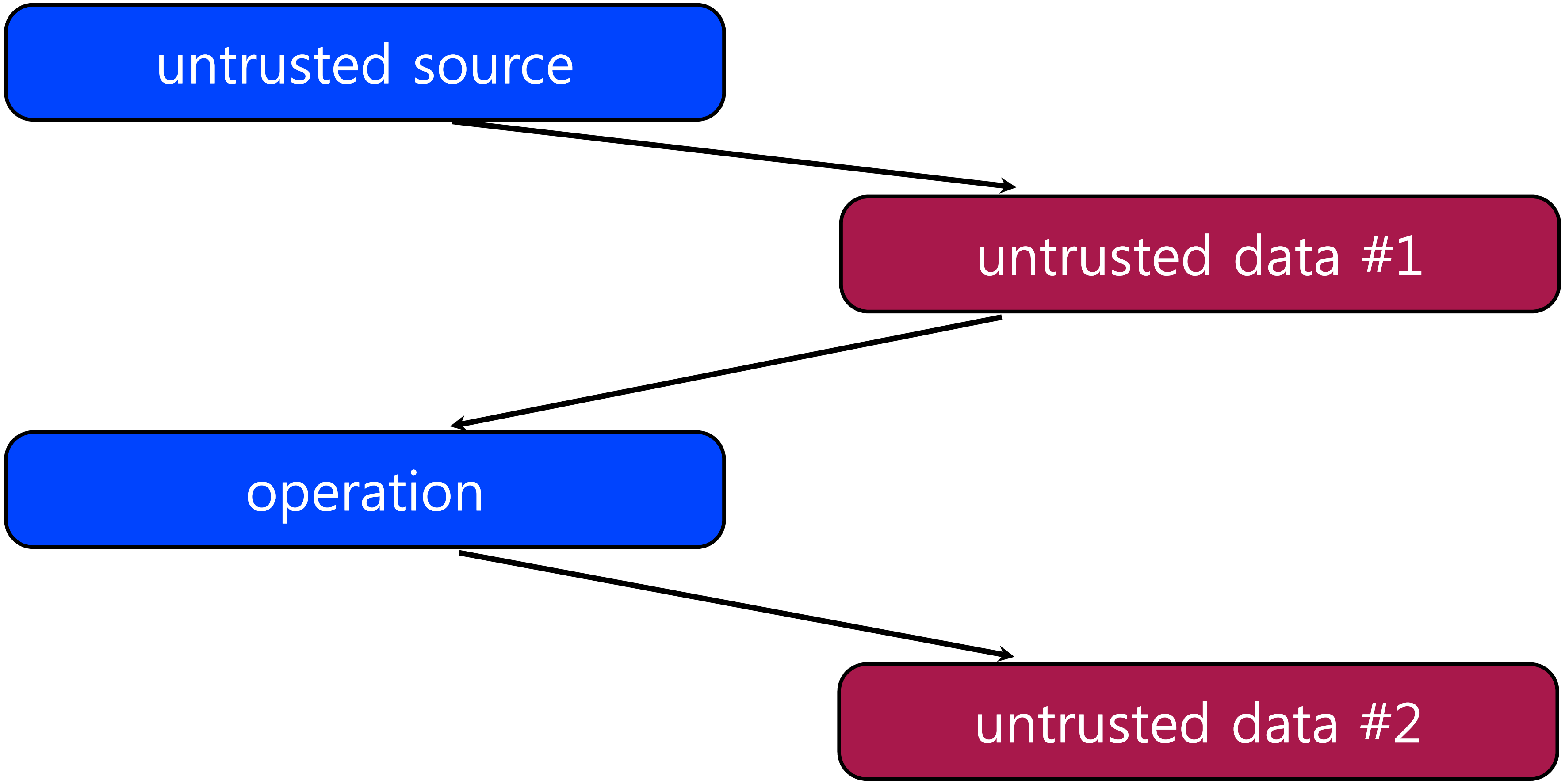
- Exploit Detections :
  - Find tainted EIP register
  - Find tainted Function Pointers
  - Find tainted Stack Arguments
  - Find tainted Data Structure using system
- Now we reverse upper follows
  - Finding Vulnerability

Other benefits

- Solve Reachability Problems
  - How can I makes PDF files to execute code block #937 in PDF reader ?
- Zero-day Detection
  - Include other bug class
- Helping Fuzzer Mutations

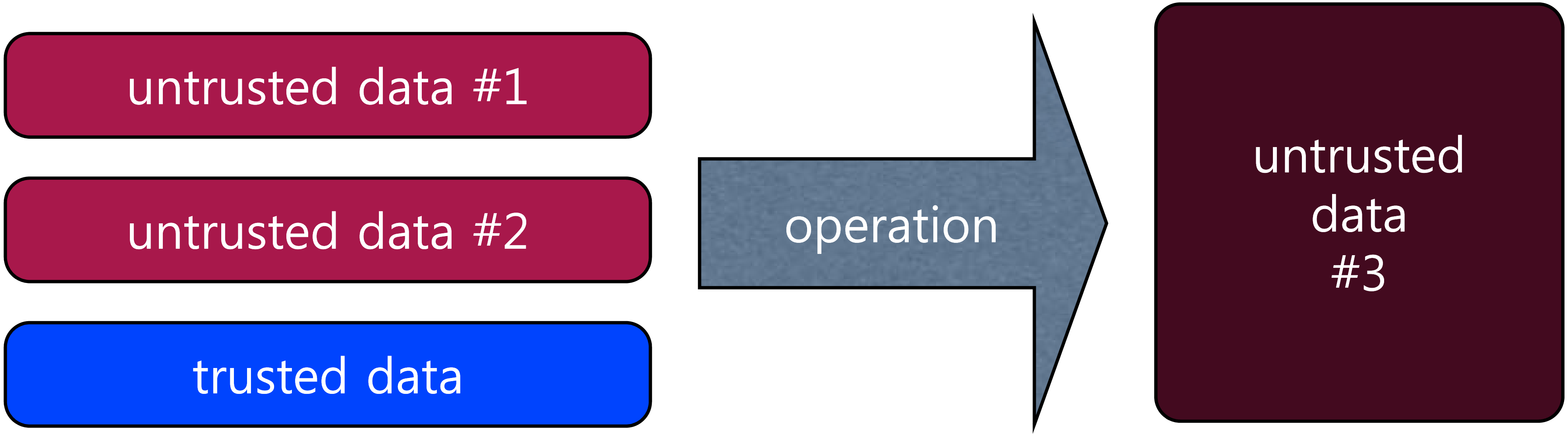
Tainted Object

- The Object from untrusted source



Tainted Object

- The Object from untrusted operation, data

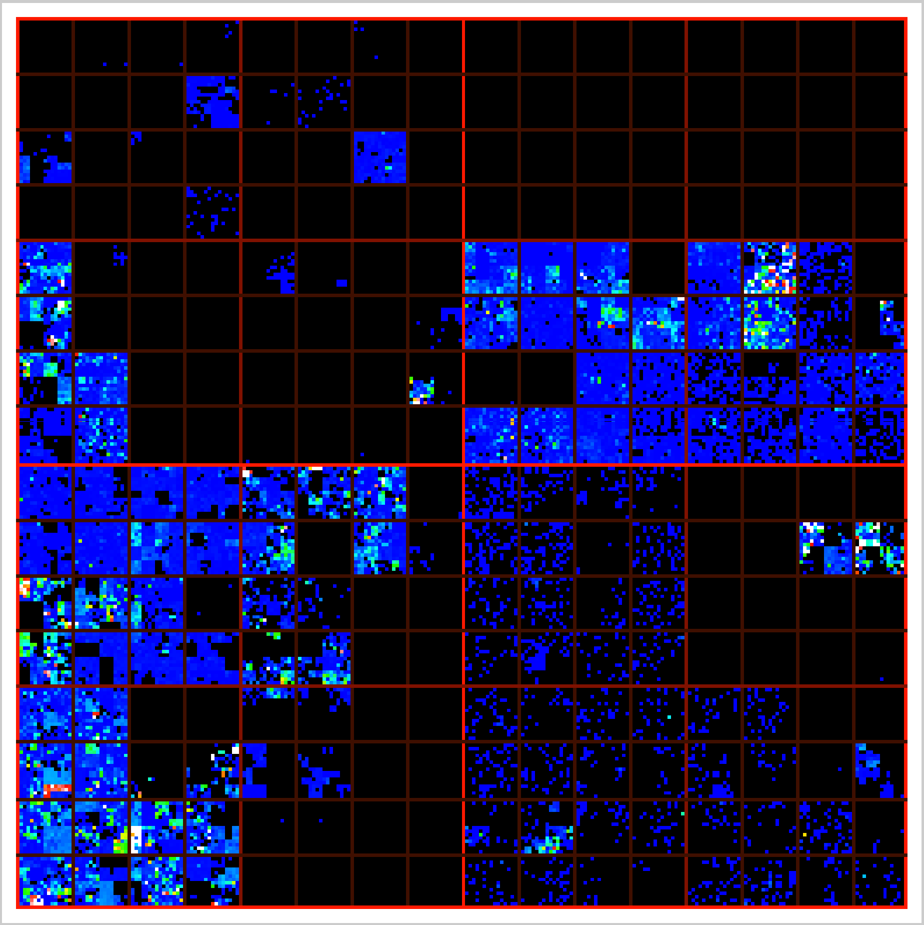


Tainted Object

- Untrusted Sources
  - Files, Inputs, Network Reads, ...

- Tainted Objects
  - Memory Locations, Process Registers

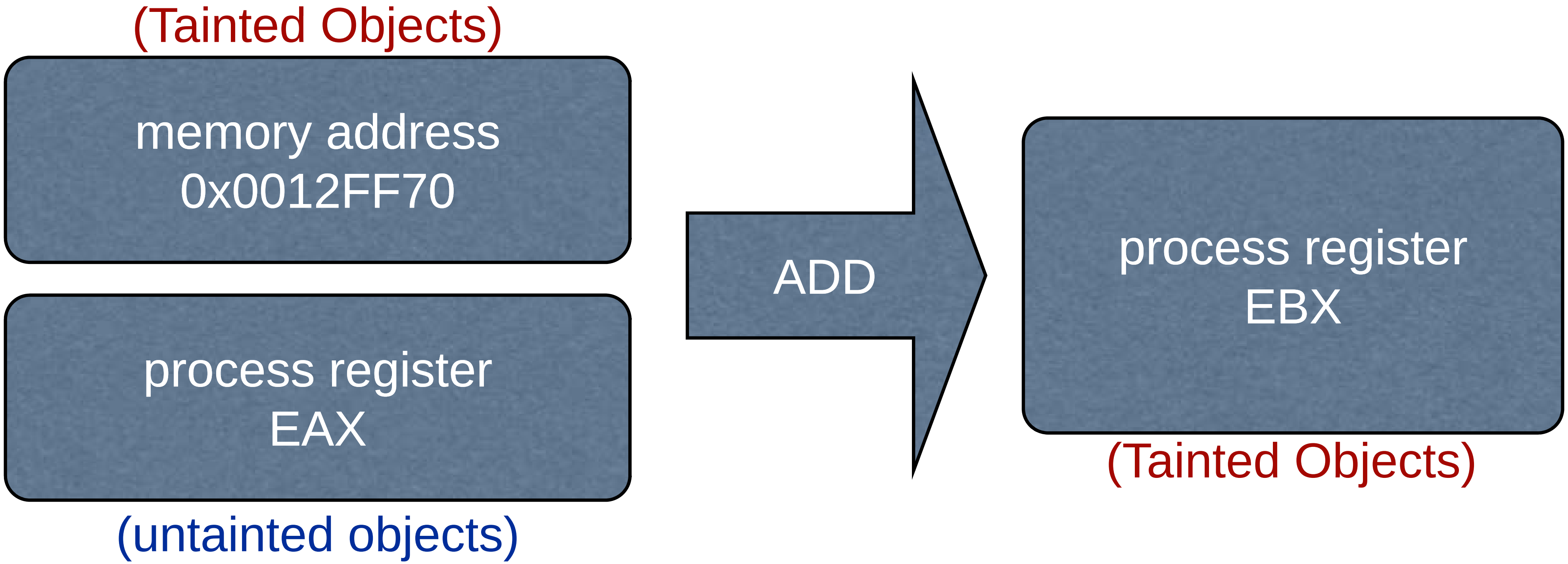
Taint Propagation on x86



Taint Propagation

- Taint Propagation is analysis for tainted object derivation activities.
- If a tainted object X derive to Y
  - we say “Y is the tainted object”
  - so, we assign this : X → T(Y)
- Taint operation is transitive
  - X → T(Y), and Y → T(Z) then, X → T(Z)

Taint Propagation



(Tainted Objects)

memory address 0x0012FF70

process register EAX

(untainted objects)

ADD

process register EBX

(Tainted Objects)

Operation on x86 which derived in tainited

- Assignment Operations
  - operation move X to Y
- Arithmetical Operations
  - operation perfumes arithmatic calculus from X
- Stack Push/Pop Operations
  - similar with Assignment Operations

Operation on x86
which derived in tainited

- Boolean Operation
  - must consider if the result of the operation depend on the value of tainted object
  - ex) AND Operation

A(tainted)	B	A && B
0	0	0(untainted)
0	1	0(untainted)
1	0	0(untainted)
1	1	1(tainted)

  - special case : X xor X is always untainted

Operation on x86 which derived in tainited

- We analysis whole program process
  - Finally, if we find tainted special object, we find a new bugs
  - special object : EIP register, function pointers, etc.

implementations of propagation

- Just trace using breakpoints
  - only memory locations
- Just trace using exceptions
  - only memory locations
- How do we trace process registers ?
  - emulation or virtualization, It is only way to propagations

implementations of propagation

- After we figure out the tainted object, every instruction has to execute after emulation.
  - So, we can figure out new tainted object.
- Or, register handler to process register using virtualization
  - this requires fully implementation for cpu emulating and memory access

<u>Simple Test For Tainting</u>

Into the Abyss :
in the wild world

welcome to wild world!

- Problem 1 : multithread or message-driven
- Problem Ⅱ : a lot of logs
- Problem Ⅲ : still can’t find ?

for the real world tainting

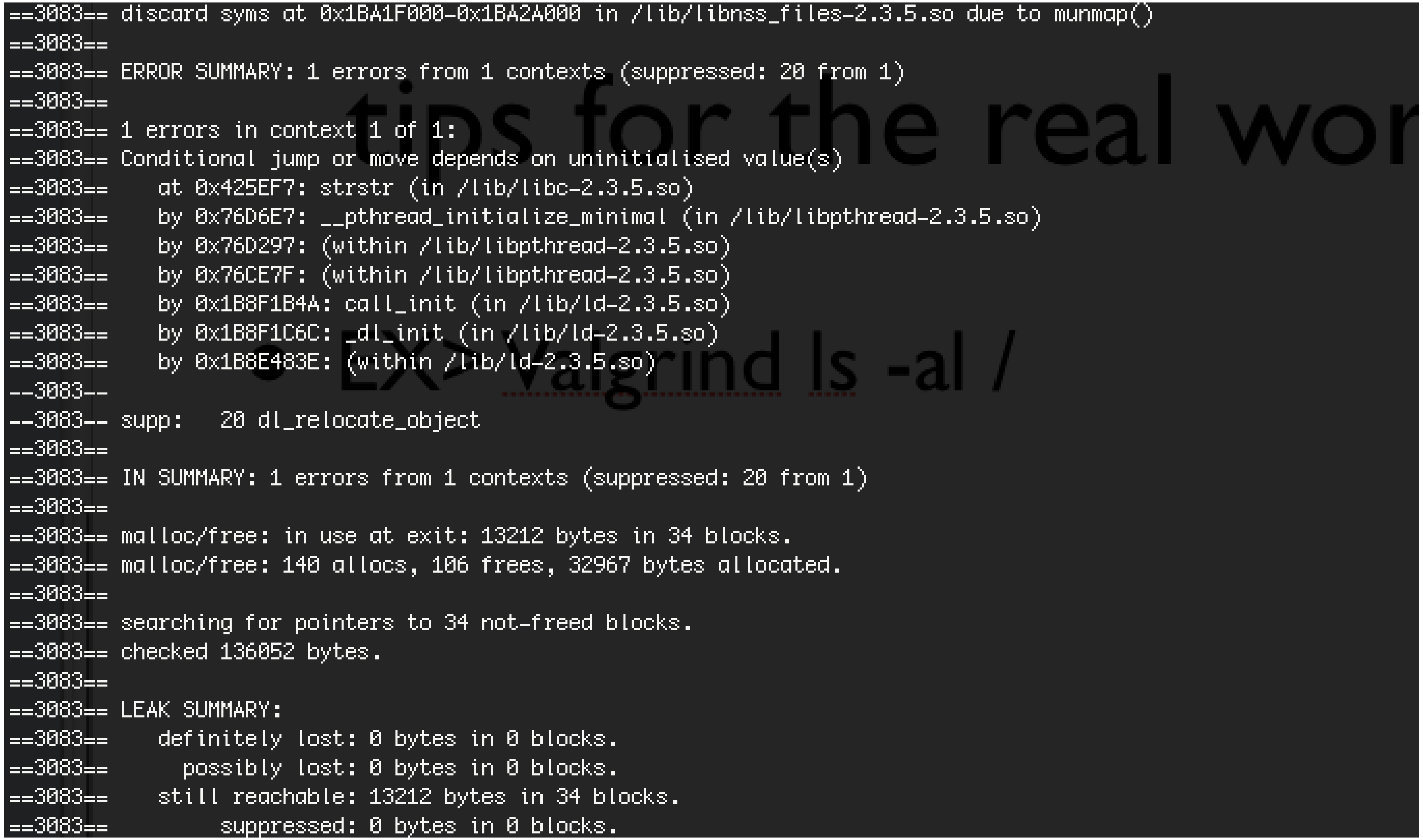
- Multithreaded or Message-Driven Program makes your fuzzer into hang over
  - Cuz, There is no automated end of program
  - So, you make fully virtualization for program
- There are tons of log
  - Is it same with mutation fuzzing ?
  - no waaaay, keep in going analysis tightly

tips for the real world tainting

- Using debugger : paimei is good for it
- Construct your own emulation for program
- Sometimes just use other guy's code
  - why not ? valgrind + wine + windows app.
  - concentrate your major subject : finding bugs.

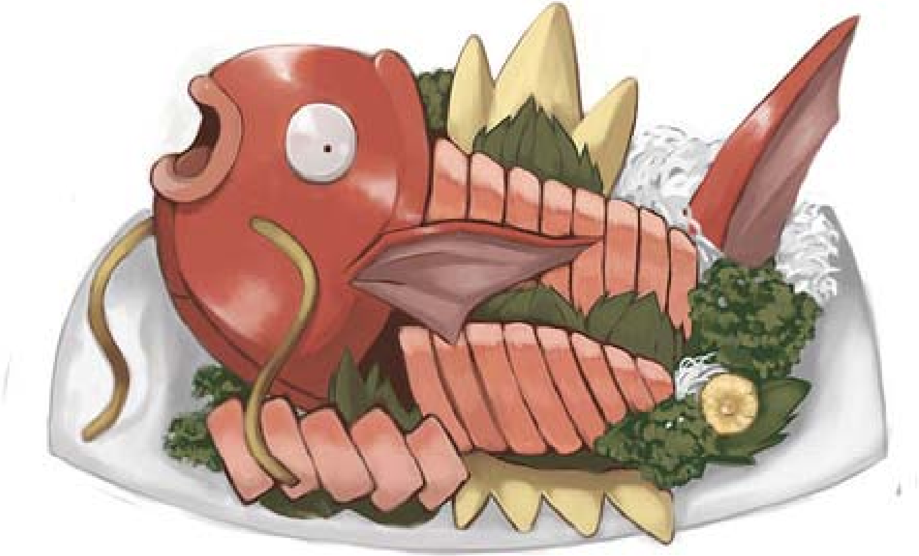
tips for the real world tainting

- EX> Valgrind ls -al /



```
==3083== discard syms at 0x1BA1F000-0x1BA2A000 in /lib/libnss_files-2.3.5.so due to munmap()
==3083==
==3083== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 20 from 1)
==3083==
==3083== 1 errors in context 1 of 1:
==3083== Conditional jump or move depends on uninitialised value(s)
==3083==    at 0x425EF7: strstr (in /lib/libc-2.3.5.so)
==3083==    by 0x76D6E7: __pthread_initialize_minimal (in /lib/libpthread-2.3.5.so)
==3083==    by 0x76D297: (within /lib/libpthread-2.3.5.so)
==3083==    by 0x76CE7F: (within /lib/libpthread-2.3.5.so)
==3083==    by 0x1B8F1B4A: call_init (in /lib/ld-2.3.5.so)
==3083==    by 0x1B8F1C6C: _dl_init (in /lib/ld-2.3.5.so)
==3083==    by 0x1B8E483E: (within /lib/ld-2.3.5.so)
--3083--
--3083-- supp:   20 dl_relocate_object
==3083==
==3083== IN SUMMARY: 1 errors from 1 contexts (suppressed: 20 from 1)
==3083==
==3083== malloc/free: in use at exit: 13212 bytes in 34 blocks.
==3083== malloc/free: 140 allocs, 106 frees, 32967 bytes allocated.
==3083==
==3083== searching for pointers to 34 not-freed blocks.
==3083== checked 136052 bytes.
==3083==
==3083== LEAK SUMMARY:
==3083==    definitely lost: 0 bytes in 0 blocks.
==3083==      possibly lost: 0 bytes in 0 blocks.
==3083==    still reachable: 13212 bytes in 34 blocks.
==3083==         suppressed: 0 bytes in 0 blocks.
```

Extras



Raison Framework

automated exploit framework

still under-constructing.....

references

- “LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks” - Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan zhou, Youfeng Wu - University of Illinois

- “BitBlaze: A New Approach to Computer Security via Binary Analysis” - Dawn Song

- “Dytan: A generic dynamic taint analysis framework” – James Clause, Wanchun Li, and Alessandro Orso. Georgia Institute of Technology.

- “Understanding data lifetime via whole system emulation” – Jim Chow, Tal Garfinkel, Kevi Christopher, Mendel Rosenblum – USENIX – Stanford University

- “Taint analysis” - edgar barbosa, H2HC 2009

- “valgrind” - http://valgrind.org/

- “paimei & pydbg” - http://pedram.redhive.com/PyDbg/docs/

- “PyEmu” - http://code.google.com/p/pyemu/