Lecture 12

Pointless Tainting? Evaluating the Practicality of Pointer Tainting

Asia Slowinska, Herbert Bos

Operating Systems Practical

8 January, 2013

- buffer overflows
  - inject code – alter control flow
- attack non-control data
  - user identity
  - user privilege level
  - server configuration string
- non-control data attacks are more difficult to detect

- type-safe languages
- compiler extensions
- formal methods verification
- however ...
  - C/C++
  - source unavailable – recompilation not possible
- trojans
  - masquerade as useful programs
  - no exploit required
  - "stealthy spies" harder to detect

- pointer dereference
- control diversion attacks
  - execute instructions different from the ones it would normally execute
  - alter flow of control
- non control diversion attacks
  - memory corruption attacks against non-control data (non-function return address etc.)
  - privacy breaching malware (keyloggers and sniffers)
  - elevated privileges, unusual replies
  - address space layout randomization & stack guard don't work

- focused by non control diversion attacks
- also works against control-diverting attacks
- a form of dynamic information flow tracking (DIFT)
  - origin of data through a taint bit in a shadow memory unaccessible to software
  - check whether values derived from tainted origin ends up in places it should never be stored
- popular
  - apply on software without need of recompilation
  - (stated by advocates) incurs hardly false positives
  - one of the only techniques of detecting both control-diverting and non-control diverting attacks

Pointer tainting – not working

- keylogger detector
  - the method is flawed
  - incurs both false positives and negative
- existing applications not suitable for x86 architecture and Windows operating systems
- analyse fundamental limitations of the method when applied to detection of privacy-breaching malware
- fixing the method is breaking it

- manipulate data that is subsequently loaded in the processor's program counter
-
```
struct req {
    char reqbuf[64];
    void (*handler)(char *);
};

void do_req(int fd, struct req *r)
{
    // now the overflow
    read(fd, r->reqbuf, 64);
    r->handler(r->reqbuf);
}
```

Non-control diverting attacks

- modify security-critical data (do not alter control flow)
- non control data attacks

```
void serve (int fd)
{
    char *name = globMyHost;
    char cl name[64];
    char svr reply[1024];

    // now the overflow:
    read(fd,cl name,128);
    sprintf(svr reply,
        "hello %s, I am %s",
        cl name, name);
    svr send(fd,svr reply,1024);
}
```

- privacy breaching malware (trojans, keyloggers)

- dynamic taint analysis
- mark (in an emulator or hardware) all data coming from suspect sources
- taint is propagated
- source operands in ALU – destination is tainted
- copy source operands – taint propagates
- "cleaning" instructions (`xor eax,eax`)
- jump to "tainted" address – alarm is raised
- protection against control-diverting attacks, but not against non-control diverting attacks

- dereference of attack-manipulated pointers (same as control-diverting attacks)
- heap corruption – change links in lists
- format string attack
- basic tainting analysis raises alerts only for dereferences due to jumps, branches and function calls/returns

- "possibly malicious" program spying on users' behaviour – keyloggers
- basic taint analysis is weak in the face of translation tables
  - `x` is tainted
  - `y = a[x]` is not tainted
  - similar for `atoi`, `to_upper`, `strtol`
- taint analysis is powerless in the face of privacy-breaching malware

- designed to handle non-control diverting attacks
- limited pointer tainting (detecting non-control data attacks)
  - p is tainted
  - raise an alert on any dereference of p
  - inapplicable in the general case
  - LPT prescribe that taint of and index is cleaned
  - LPT cannot be used for tracking keystrokes
  - if p is tainted raise an alert on any dereference of p
- full pointer tainting (detect privacy breaching)
  - propagates taint
  - if p is tainted, any dereference of p taints the destination
  - looks ideal for privacy-breaching malware applications

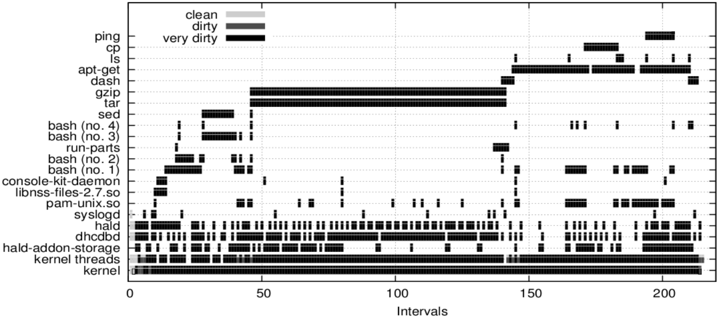
- Qemu 0.9
- Ubuntu 8.04.1, kernel 2.6.24-19-386
- Windows XP SP2
- depending on test, modify emulator to taint either
  - typed keyboard characters
  - network data
- inspect taintedness of register at context-switch times
- the more register are tainted the worse the problem
  - particularly serious for esp and ebp

- conservative measurements
  - register may be clean but not bytes in process' address space
  - check registers only at context-switch times
  - sufficient to present the problem of false positives
- taintedness in Linux
  - `schedule()`
  - `context_switch()` – monitor taintedness inside the kernel
- taintedness in Windows
  - `cr3` inspection – contains the physical address of the top-level page directory
  - `cr3` change $\rightarrow$ a new process is scheduled

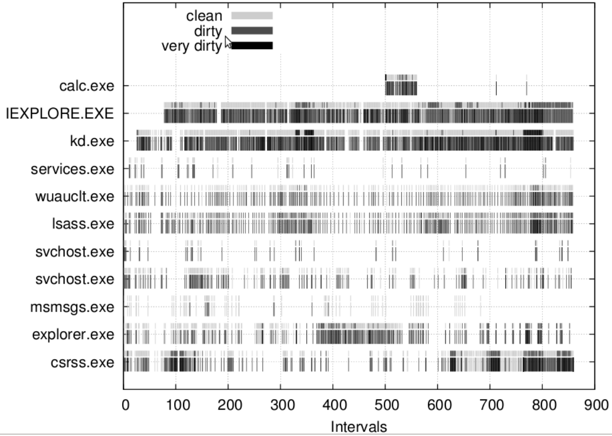
- taint data from network
- alerts raised for benign actions like configuring the machine's IP address
- LPT propagates taint when combining and untainted base pointer and a tainted index
- dereferencing causes an alert

- simple keystroke tracing – all taint that is applied
- simple C program – reads a user typed character from the command line

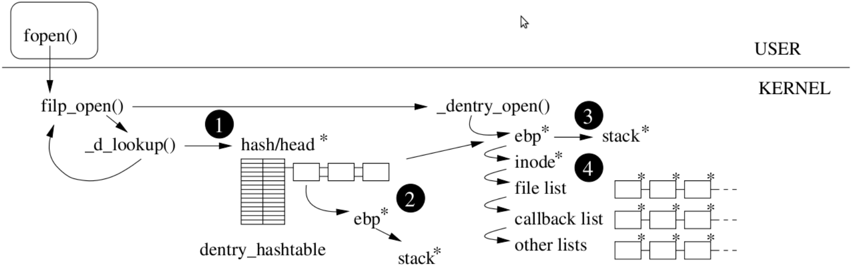
Taint explosion for FPT (2)



Taint explosion for FPT (3)



- containment measures required
- pollution of the kernel
- problematic usage of esp and ebp


fopen()
USER
KERNEL
filp_open()
_dentry_open()
_d_lookup()
1
hash/head *
dentry_hashtable
2
ebp*
stack*
3
ebp*
stack*
inode*
4
file list
callback list
other lists

False positives and root causes of taint pollution

- tainting of ebp and esp
  - LPT raises alarms quickly
  - FPT spreads taint indiscriminately
- pointers are tainted in the same way
  - A tainted, what about `B = (A+0x4)`?
- if taint is applied only for detecting memory corruption attacks, taint may leak due to table lookups

- pure LPT and FTP does not have many false negatives
- however ...
  - LPT will miss modification of non-control data by means of a direct buffer overflow
- miss implicit information flows
  - `if (x == 0) y = 0; else y = 1`
- reduce false positive $\rightarrow$ opportunities for false negatives will increase significantly

- both LPT and FPT
- basic idea – never apply pointer tainting to tainted values of ebp and esp
- ebp is used as a general purpose register
- clean ebp when value is large enough to represent a frame pointer
- although taint is slowed down, it still propagates quickly

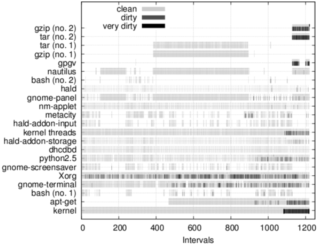
LPT-specific containment techniques

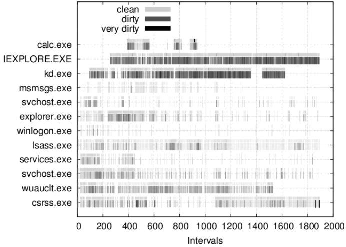
- prevent taint from leaking due to table lookups
- detect and sanitise table accesses
  - impractical on x86 – no specific instructions for pointer arithmetic
- bounds checks – safe even if index is tainted provided the index was properly bounds-checked
  - identified by a `cmp` instruction
  - suffers from false positives and false negatives
- pointer injection detection
  - use a P bit to mark valid pointers
  - applied on SPARC v8 architecture
  - false positives possible – overflow a buffer, modify and index, add index to a legitimate address
  - not easily applicable to x86

FTP-specific techniques

- white lists and black lists
  - white list all places where tainting should be propagated
  - black list all places where tainting should not be propagated
  - unfeasible for large applications
  - heavy impact on performance
- landmarking
  - an address is "ready to be used for a dereference"
  - dereferencing a landmark – propagate taint
  - derived values have to be modified with tainted data
  - opportunities for false positives and false negatives abound

Landmarking and esp/ebp protection





- prone to false negatives
- only slow down the outburst of false positives
- difficult to distinguish access to a translation table from access to a `next` field in a linked list
- without a priori information it's impossible to successfully apply FPT (on current hardware)

- pointer injection (P bit) seems promising
- have to get it to work on common hardware
- possible for Linux on SPARC
- open challenge to do it for x86

- pointer tainting – considered one of the most powerful techniques to detect keyloggers and memory corruption attacks on non-control data
- proved problematic – large number of false positives
- FPT is probably not suited for detecting keyloggers
- unclear whether LPT can be applied to automatically detect memory corruption attacks on x86

- exploit
- DIFT
- taint analysis
- pointer tainting
- control diversion
- control data
- non-control data
- memory corruption
- keylogger, trojans
- x86 (Linux & Windows)
- limited pointer tainting (LPT)
- full pointer tainting (FPT)
- false positives, false negatives
- esp/ebp protection
- pointer injection detection
- landmarking

- Asia Slowinska, Herbert Bos – Pointless Tainting? Evaluating the Practicality of Pointer Tainting
- Asia Slowinska, Herbert Bos – Pointer tainting still pointless: (but we all see the point of tainting)