More Low-level software vulnerability protection mechanisms

CS642: Computer Security

Spring 2019

DNSpionage

- Attack: on UEA, Lebanon
  - Redirect domain name lookup (e.g., www.google.com) to attacker server
  - Redirect user traffic to attacker machines
  - Capture email passwords
  - Capture encryption certificates
  - Decrypt intercepted email

DNS hijacking

- Idea: change mapping of domain names to IP addresses
  - These are stored in a server without much protection
  - Broke into Netnod domain name registry
- Obtain SSL/TLS certificates for these domains
  - Means clients will believe they are connecting securely
  - Means certificate authorities failed
- How normally prevent? DNSSEC puts digital signature on domain names
  - But SSL/TLS certificates were used to spoof DNSSEC

How can we help prevent exploitation of buffer overflows and other control flow hijacking?

Non-executable memory pages

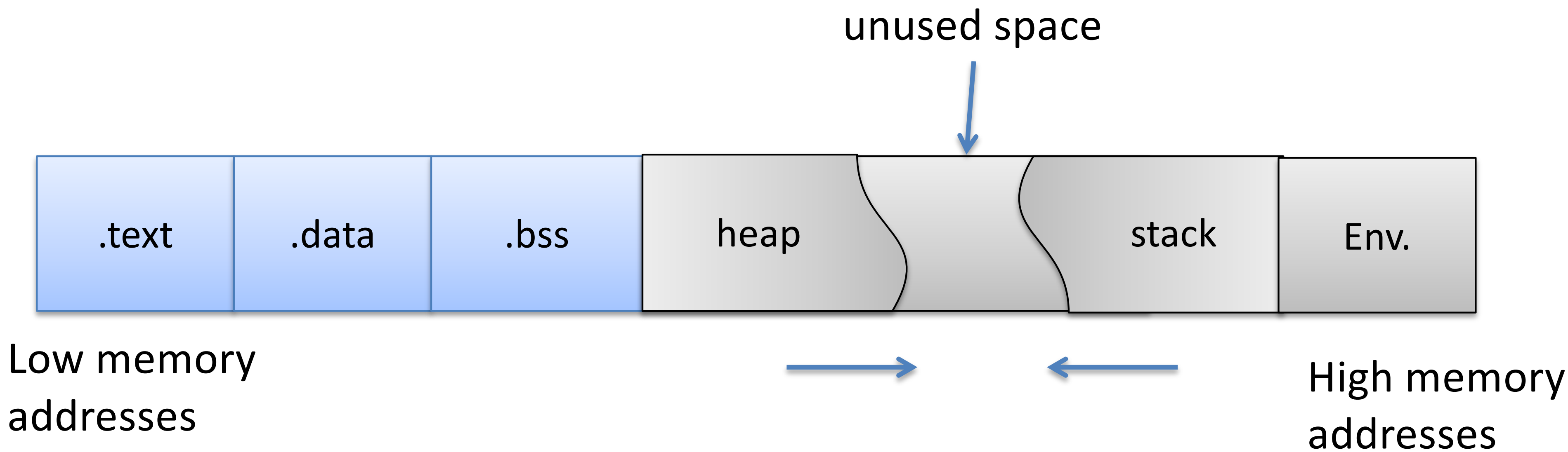
Return-into-libc exploits, Return-oriented programming

Address space layout randomization

StackGuard, StackShield

Software fault isolation

Process memory layout



.text:
- machine code of executable

.data:
- global initialized variables

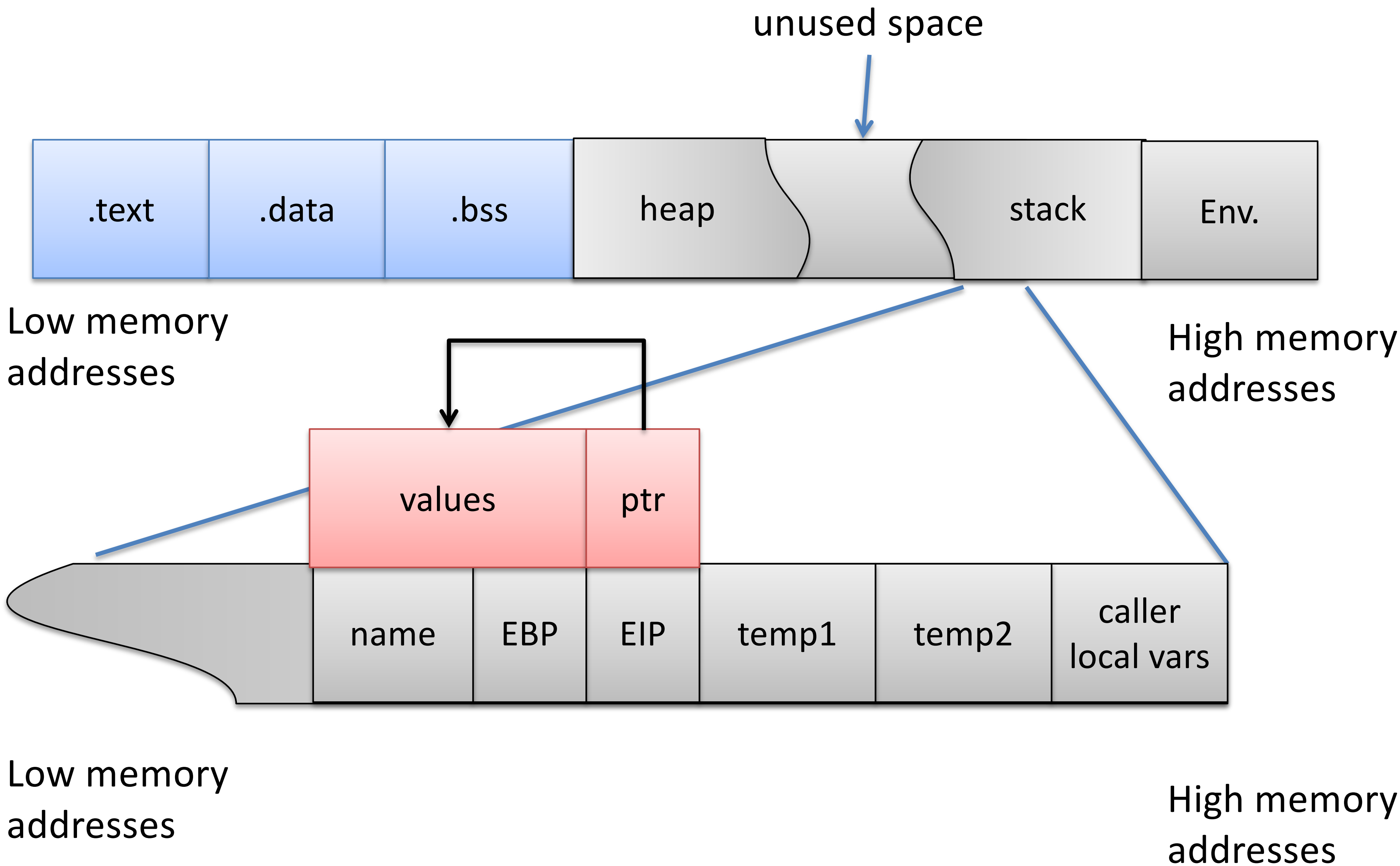
.bss:
- "below stack section"
- global uninitialized variables

heap:
- dynamic variables

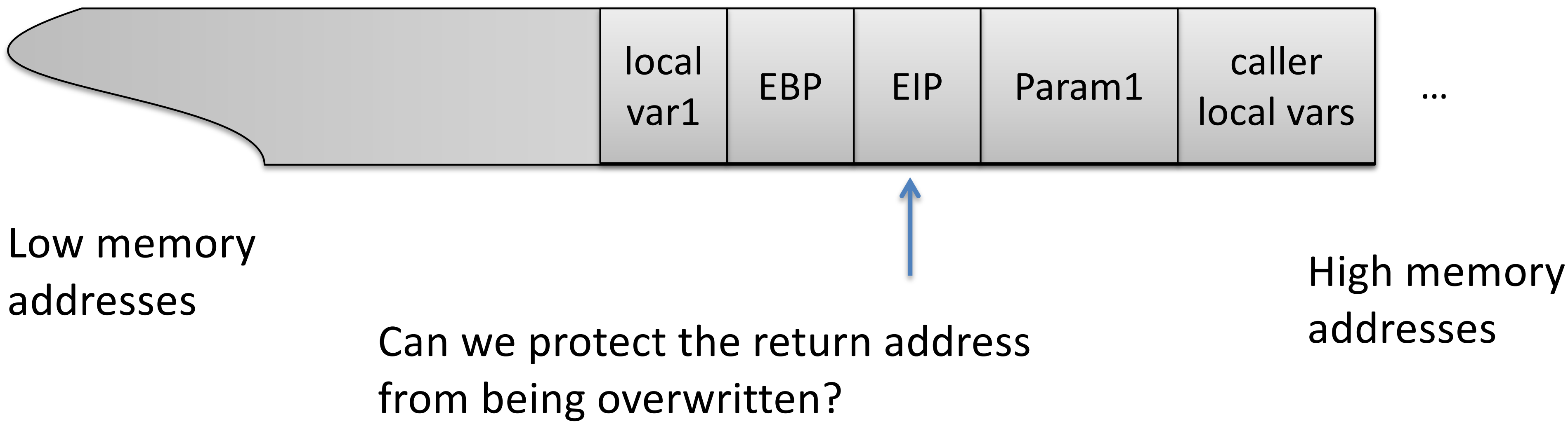
stack:
- local variables, track func calls

Env:
- environment variables,
- arguments to program

Typical return ptr overwrite exploit



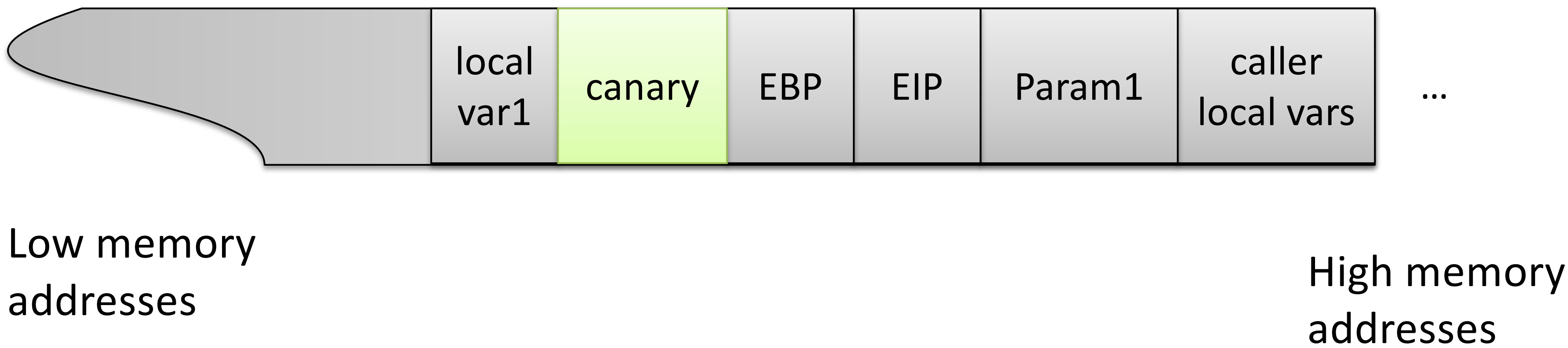
Protecting the stack



Two approaches:

- Detect manipulation (and then fail safe)
- Prevent it completely

Detection: stack canaries

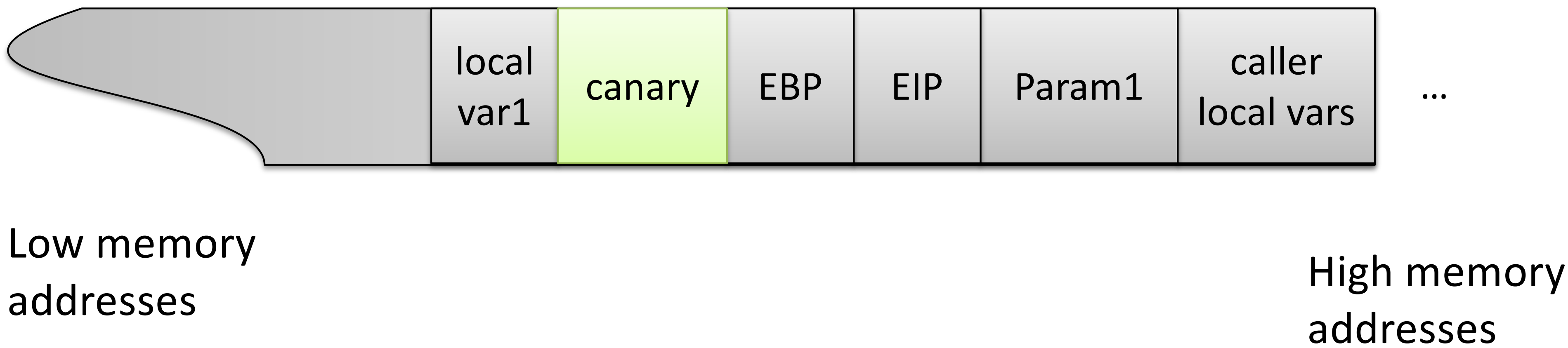


Canary value can be:

- Random value (choose once for whole process)
- NULL bytes / EOF / etc. (string functions won't copy past canary)

On end of function, check that canary is correct, if not fail safe

Detection: stack canaries



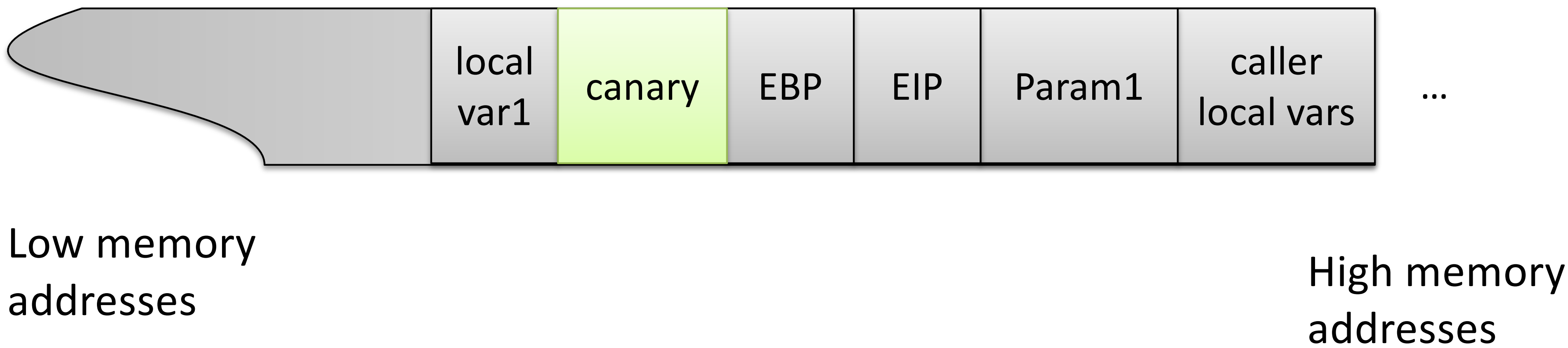
StackGuard:

- GCC extension that adds runtime canary checking
- 8% overhead on Apache

ProPolice:

- Modifies how canaries inserted
- Adds protection for registers
- Sorts variables so arrays are highest in stack

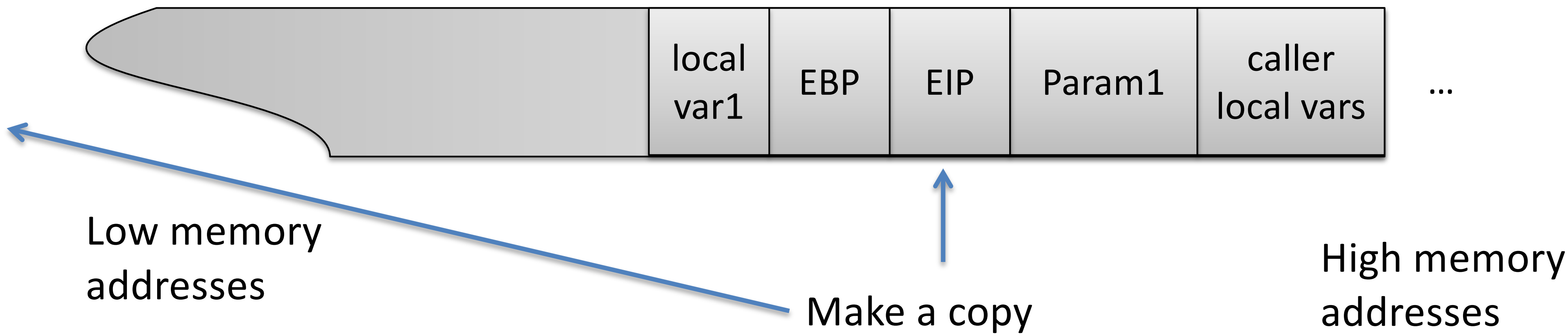
Detection: stack canaries



Discussion: How would you get around it?

http://www.phrack.org/issues.html?issue=56&id=5

Detection: copying values to safe location

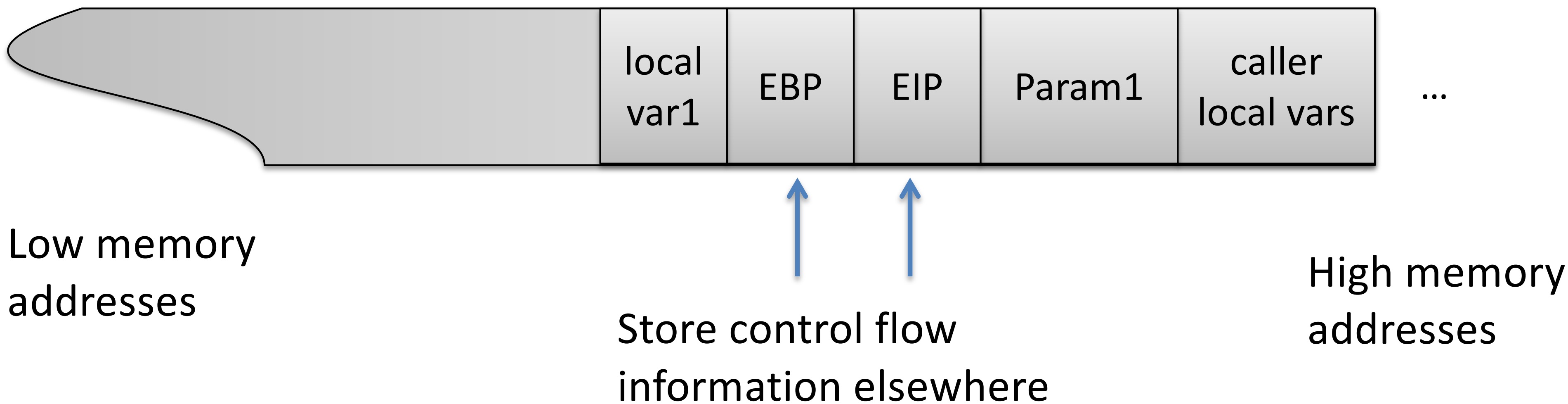


StackShield:

- Function call: copy return address to a safe location (beginning of .data)
- Check if stack value is different on function exit

Discussion: How would you get around this?

Prevention



StackGhost:

- Encrypting the return address
  - XOR with random value on function entrance
  - XOR with same value on function exit
- Per-kernel XOR vs. Per-process XOR
- Return address stack

Confinement (sand boxing)

- All the mechanisms thus far are circumventable
- Can we at least confine code that is potentially vulnerable so it doesn’t cause harm?

Simple example is chroot

chroot /tmp/guest
su guest

Now all file access are prepended with /tmp/guest

open(“/etc/passwd”, “r”)

Attempts to open
/tmp/guest/etc/passwd

Limitation is that all needed files must be inside chroot jail

Limitation: network access not inhibited

Escaping jails

open(“../../etc/passwd”, “r”)

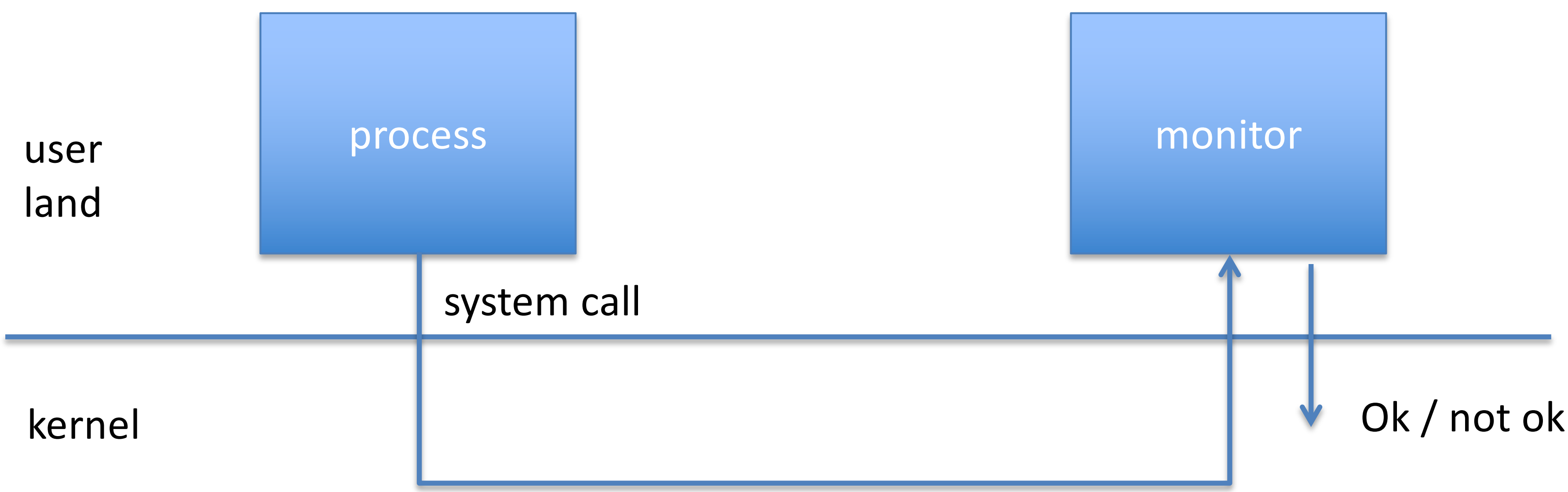
Attempts to open
/tmp/guest/../../etc/passwd

chroot should only be executable by root

create /aaa/etc/passwd
create /aaa/etc/sudoers
chroot /aaa
sudo …

System call interposition

- Malicious code must make system calls in order to do bad things
- So monitor system calls!



Janus

Wagner et al.

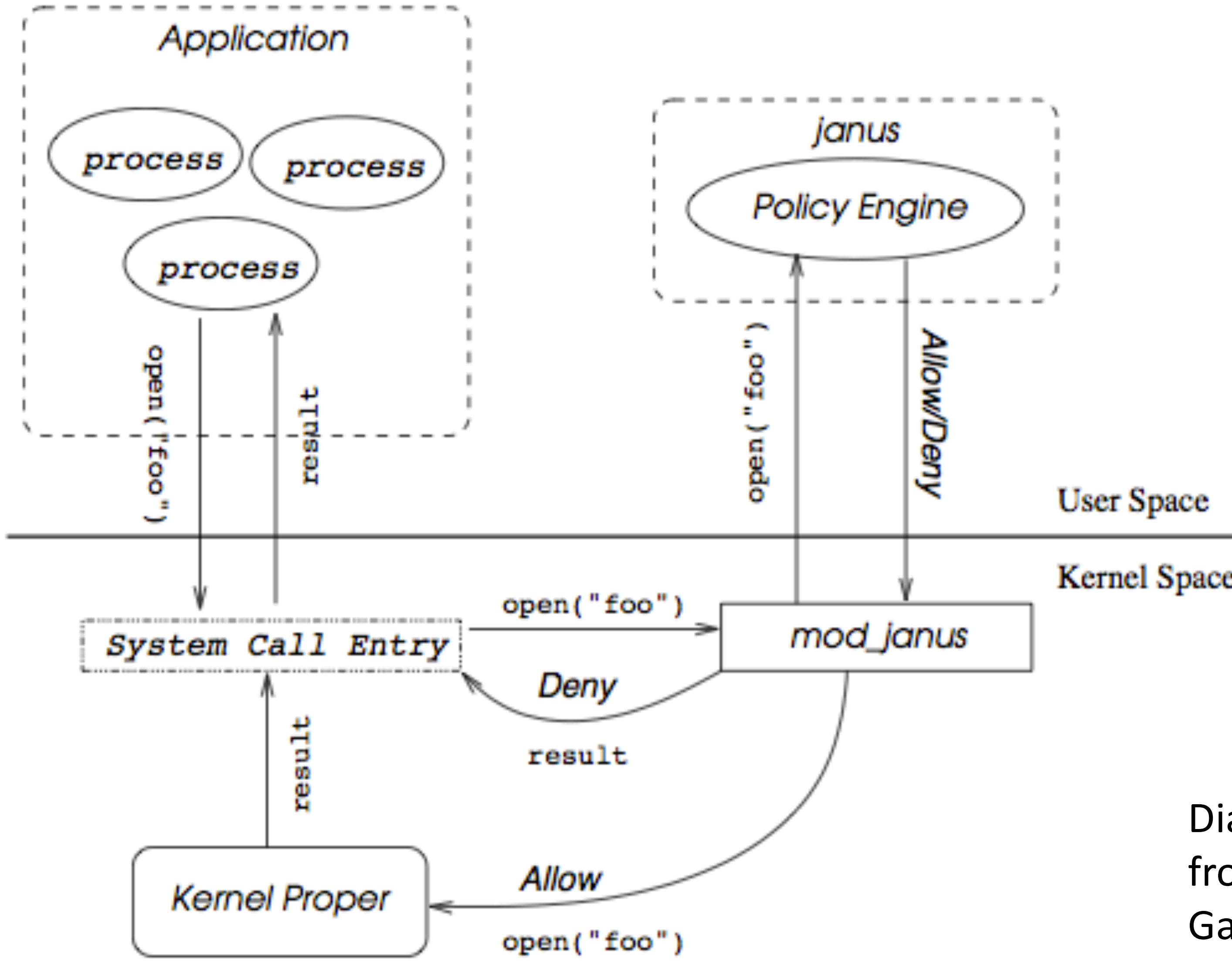


Figure 1. System Call Interposition in Janus

Diagram from Garfinkel 2003

Software-fault isolation example: Google Native Client

Goal: run native code from a web browser safely

Examples are Quake and XaoS ported over

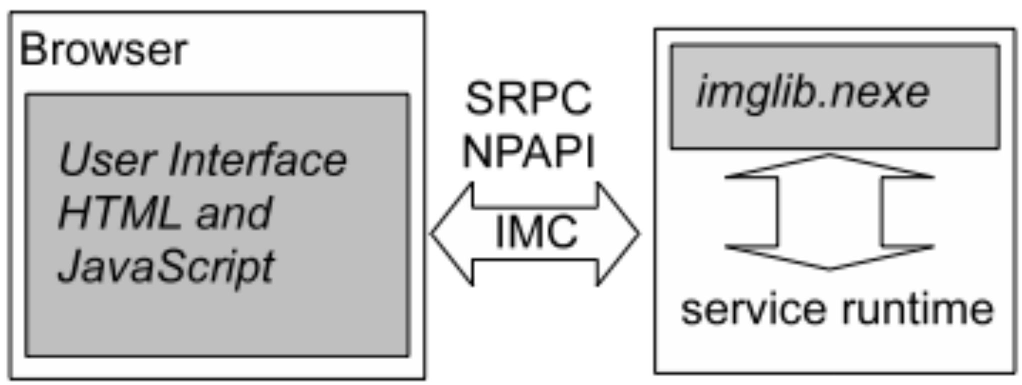


Figure 1: Hypothetical NaCl-based application for editing and sharing photos. Untrusted modules have a grey background.

From Yee et al. 2009

Software-fault isolation example: Google Native Client

Inner sandbox

- require code to abide by alignment and structure rules, allowing disassembly.
  Instruction on 16-byte boundaries (no jump inside instruction
- Fail if any disallowed instructions
- All user addresses in a range
  - No write outside range

Validator quickly checks that a binary abides by these rules

Software-fault isolation example: Google Native Client

Outer sandbox

- system call interposition to monitor
- similar to Janus / ptrace

Native client spec perf

	static	aligned	NaCl	increase
ammp	200	203	203	1.5%
art	46.3	48.7	47.2	1.9%
bzip2	103	104	104	1.9%
crafty	113	124	127	12%
eon	79.2	76.9	82.6	4.3%
equake	62.3	62.9	62.5	0.3%
gap	63.9	64.0	65.4	2.4%
gcc	52.3	54.7	57.0	9.0%
gzip	149	149	148	-0.7%
mcf	65.7	65.7	66.2	0.8%
mesa	87.4	89.8	92.5	5.8%
parser	126	128	128	1.6%
perlbmk	94.0	99.3	106	13%
twolf	154	163	165	7.1%
vortex	112	116	124	11%
vpr	90.7	88.4	89.6	-1.2%

Table 4: SPEC2000 performance. Execution time is in seconds. All binaries are statically linked.

Native client Quake perf

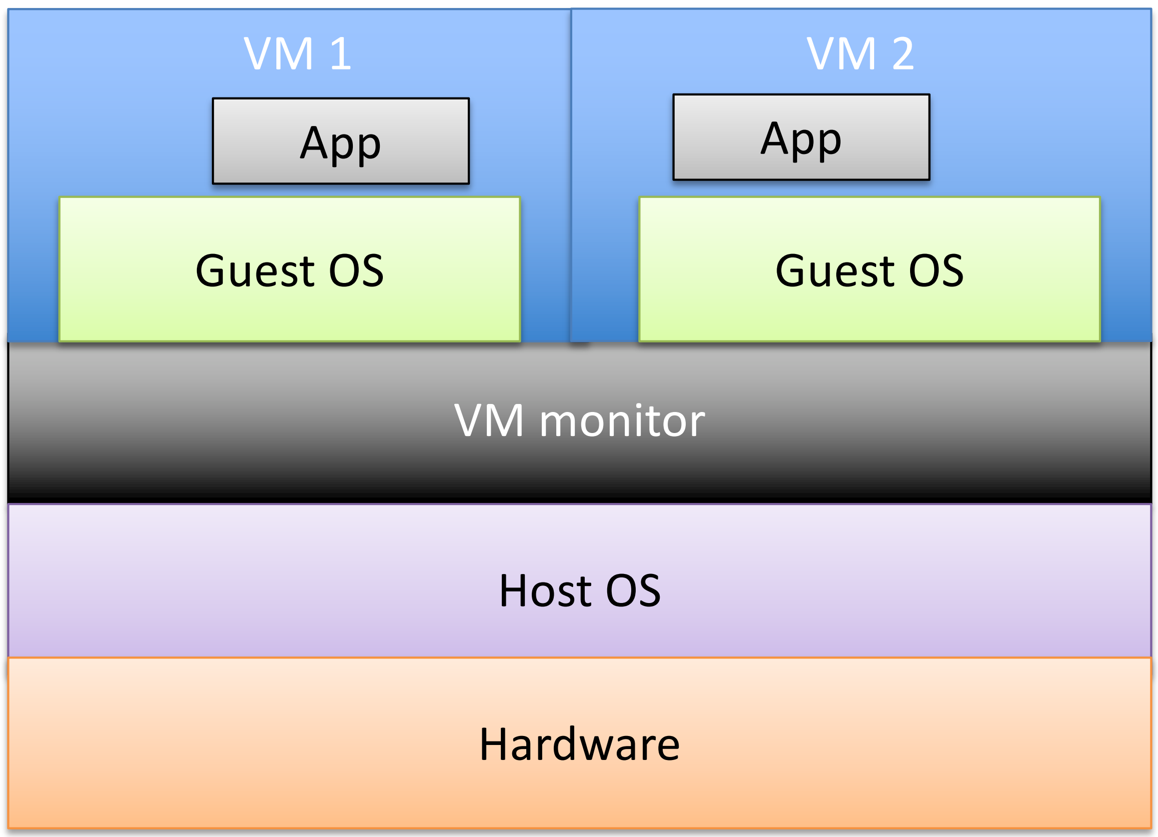
Run #	Native Client	Linux Executable
1	143.2	142.9
2	143.6	143.4
3	144.2	143.5
Average	143.7	143.3

Table 8: Quake performance comparison. Numbers are in frames per second.

More sandboxing: virtualization

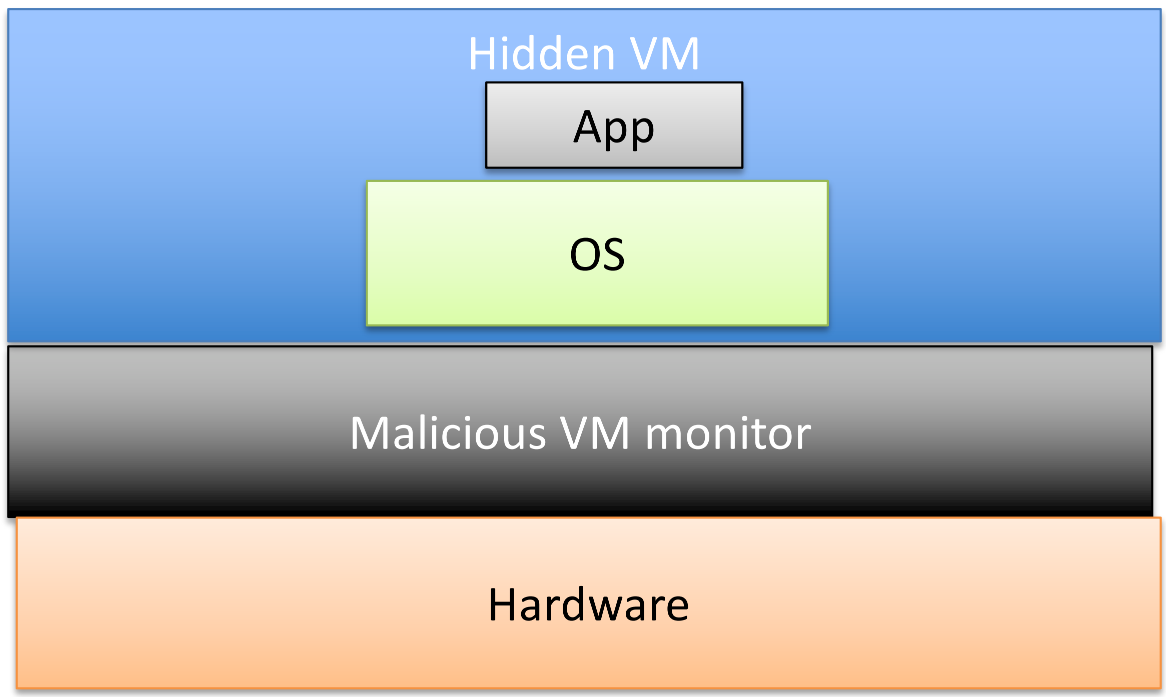
- Modern virtual machines (VMs) often used for sandboxing

NSA NetTop



More sandboxing: virtualization

- Malicious use of virtualization: blue pill virus



Discussion:
state of low level software security

- Do you think Native Client is fool proof?
- What about VM-based sandboxing?

- How does all this make you feel?