




How They Do It: Unix Hacking 101

Hal Pomeranz
Deer Run Associates




Who Am I?

- 
- Independent security consultant
 - SANS Institute Senior Faculty
 - Technical Editor for *Sys Admin*
 - Unix Technical Advisor for the Center for Internet Security

Generally speaking, a guy who probably spends way too much time with Unix...



What's In This Course?

- 
- Overview of common techniques for breaking into Unix systems
 - Detailed coverage of stack smashing (buffer overflow) attacks and defenses
 - A look at what happens *after* a successful break-in: rootkit installation
 - Live demos, prevention techniques, etc.



What's Your Job?



ASK QUESTIONS!




Overview






Hierarchy of Vulnerabilities

- 
- Physical access
 - Captured (or weak/default) password
 - Deliberate malware
 - Software vulnerabilities (including race conditions, buffer overflows, et al)
 - Subverted "trust relationships" (`.rhosts` files, X Windows authentication, etc.)
 - Session hijacking




Physical Access = root

- 
- Single-user boot
 - Boot off of OS Media
 - Corrupt root file system
 - Steal the disk drives!

Keep those critical servers locked up!



Corrupt Root File System

- 
- Repeatedly power-cycle system
 - Root file system will eventually become inconsistent– manual `fsck` required
 - System will come up at root shell without asking for a password
 - Attacker can `fsck` file system and change root password, etc.




Weak Passwords

- We know reusable passwords are bad
- Better authentication systems are expensive to implement
- Common solutions are far from ideal:
 - Account expiration, password change
 - "Cracking" passwords and harassing users
- Unfortunately, many Unix systems lack ability to force "strong" passwords





Deliberate Malware

- 
- Common Unix vectors:
 - Trojan software distributions (often after break-ins at software archive servers)
 - Trap-doors in web programming libraries
 - Critical to verify software signature (with PGP, if possible) before installing
 - Consider testing on isolated network before production deployment




File System Race Conditions

- 
- Typically caused by programs writing to directories with "unsafe" permissions
 - Compounded by programs which choose "predictable" file names
 - Attacker creates a symlink which causes program to modify an unexpected file
 - Window of vulnerability can be very small— perhaps a few CPU cycles



1992 SunOS `/bin/mail` Hole

- 
- `/bin/mail` called by Sendmail to deliver mail into `/var/spool/mail`
 - `/bin/mail` checks mail file to see if it's a symlink and then opens file
 - Between the check and the open, attacker creates a symlink to `/.rhosts`




Other Predictable File Names

- `mktemp()` generates temporary file names based on template string
- Attacker can usually predict next file name
- Attacker makes link after `mktemp()` call and before `open()`
- Use `mkstemp()` instead or at least call `open()` with `O_CREAT|O_EXCL` set




Set-UID Script Race Condition

- 
- Make a symlink to set-UID script
 - Execute symlink
 - While kernel is loading interpreter, re-point symlink to root shell script
 - New script executes with set-UID permissions from old script!




Trust Problems – `.rhosts`

- 
- This mechanism replaces password authentication with IP/host-based auth:
 - IP addresses can be spoofed
 - DNS can be corrupted
 - Root privs on other hosts can be exploited
 - Attacker can often "reverse" `.rhosts` files and find other machines to break




X Windows Exploits

- 
- Attacker controlling your display can:
 - Get remote video output
 - Capture all keystrokes (read passwords)
 - "All or nothing" access model is a real problem here




Session Hijacking

- 
- Attacker "takes over" session in progress:
 - Attacker doesn't have to guess passwords
 - Pretty obvious to affected user, however
 - Two types of session hijacking attacks:
 - Local attacks snoop streams via kernel
 - Remote attacks from third-party machines




Protecting Yourself

- 
- Patching is important, but it only protects you from *known vulnerabilities*
 - Disabling services you're not using protects you from as yet unknown attacks
 - Use firewalls at the network and host level to control access and drop bogus traffic
 - Encrypt all network traffic with SSH, SSL, IPSEC, or other strong VPN



Extra Credit Items

- 
- Use a strong, two-factor authentication system for user access (expensive)
 - Abandon `.rhosts` files in favor of DSA authentication via SSH (requires re-tooling)
 - Monitor system configurations with a tool like AIDE or Tripwire[™] (mgmt issues)




Stack Smashing

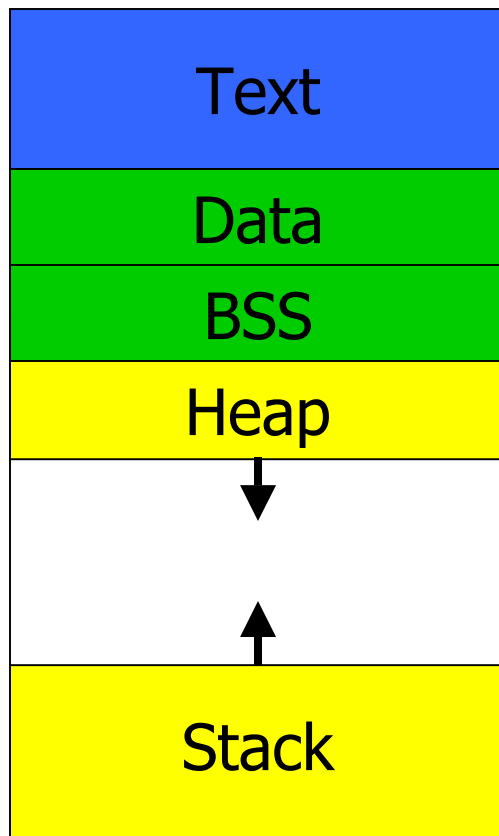




The History of Buffer Overflows

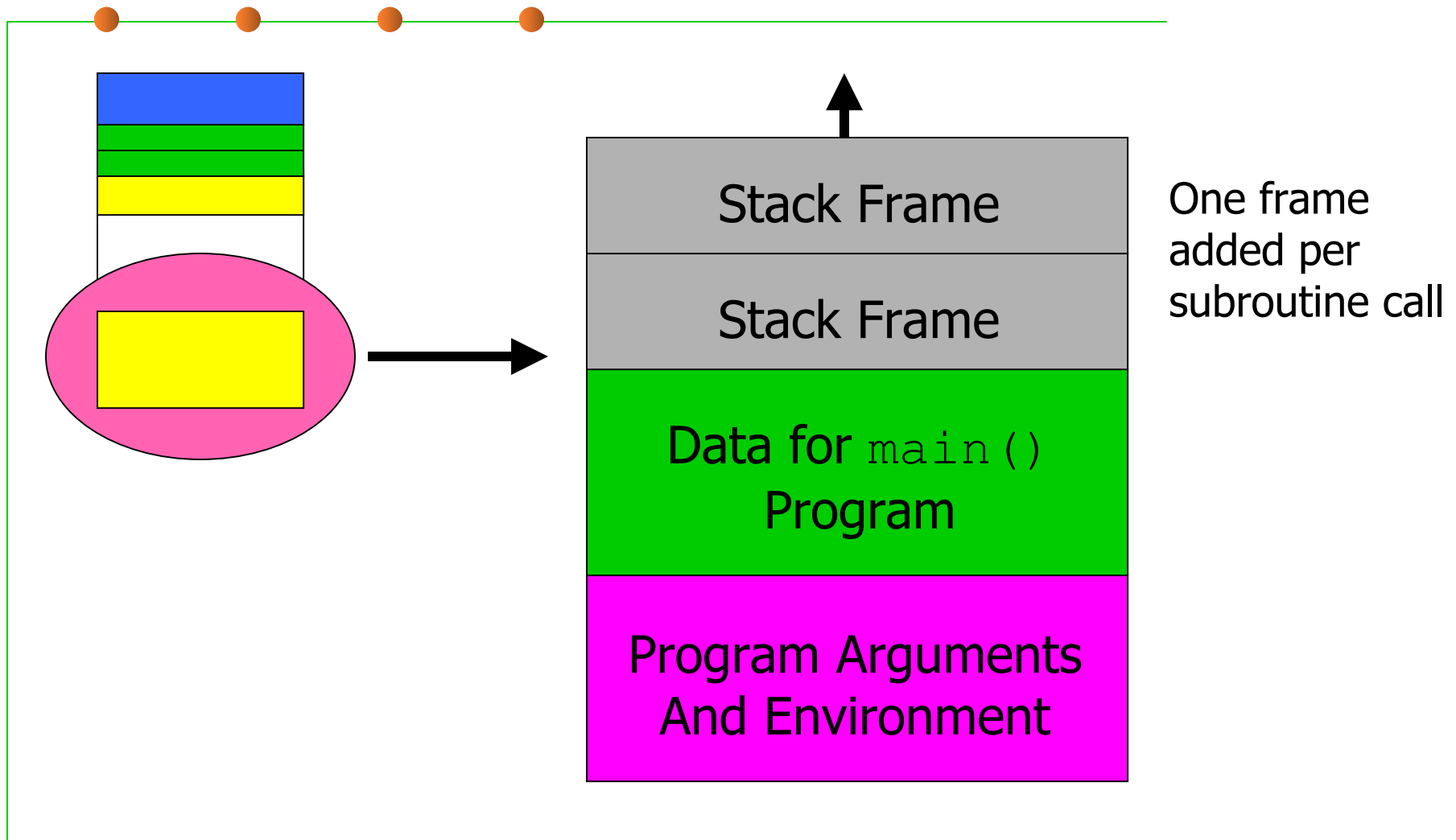
- 
- First wide-spread buffer overflow attack was the 1988 Morris Worm (**fingerd**)
 - Mudge's white paper in 1995 apparently popularized the term "buffer overflow"
 - Buffer overflow exploits on Linux/Solaris used to motivate Y2K DDoS attacks
 - Fully automated worms now automatically infecting systems via these vulnerabilities

Process Memory

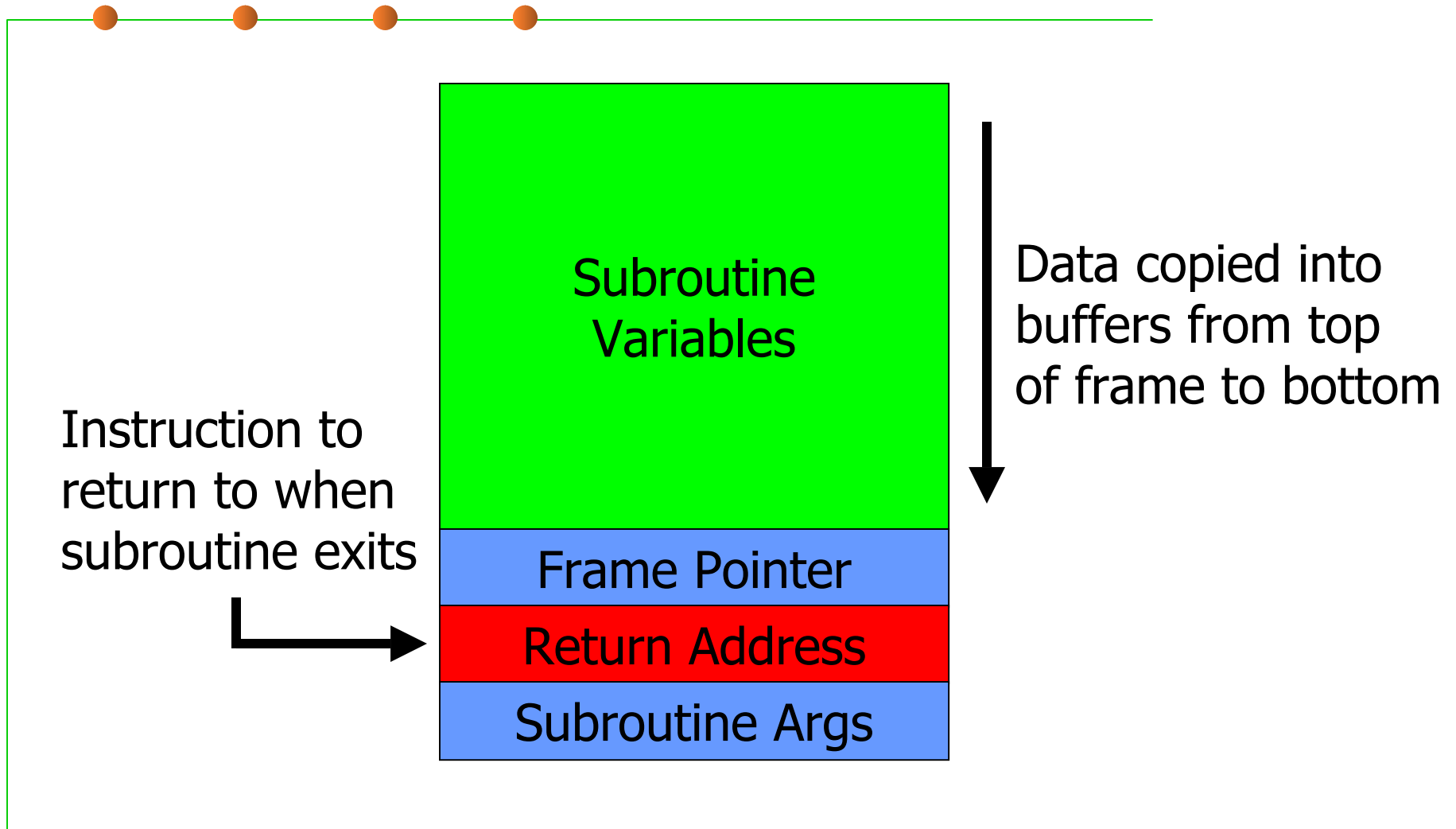


- ← Program instructions (read-only)
- ← "Constants" – literal strings/numbers
- ← Global and persistent (static) variables
- ← Dynamically allocated data
- ← Free/unallocated memory
- ← Local subroutine data

The Stack




Stack Frame

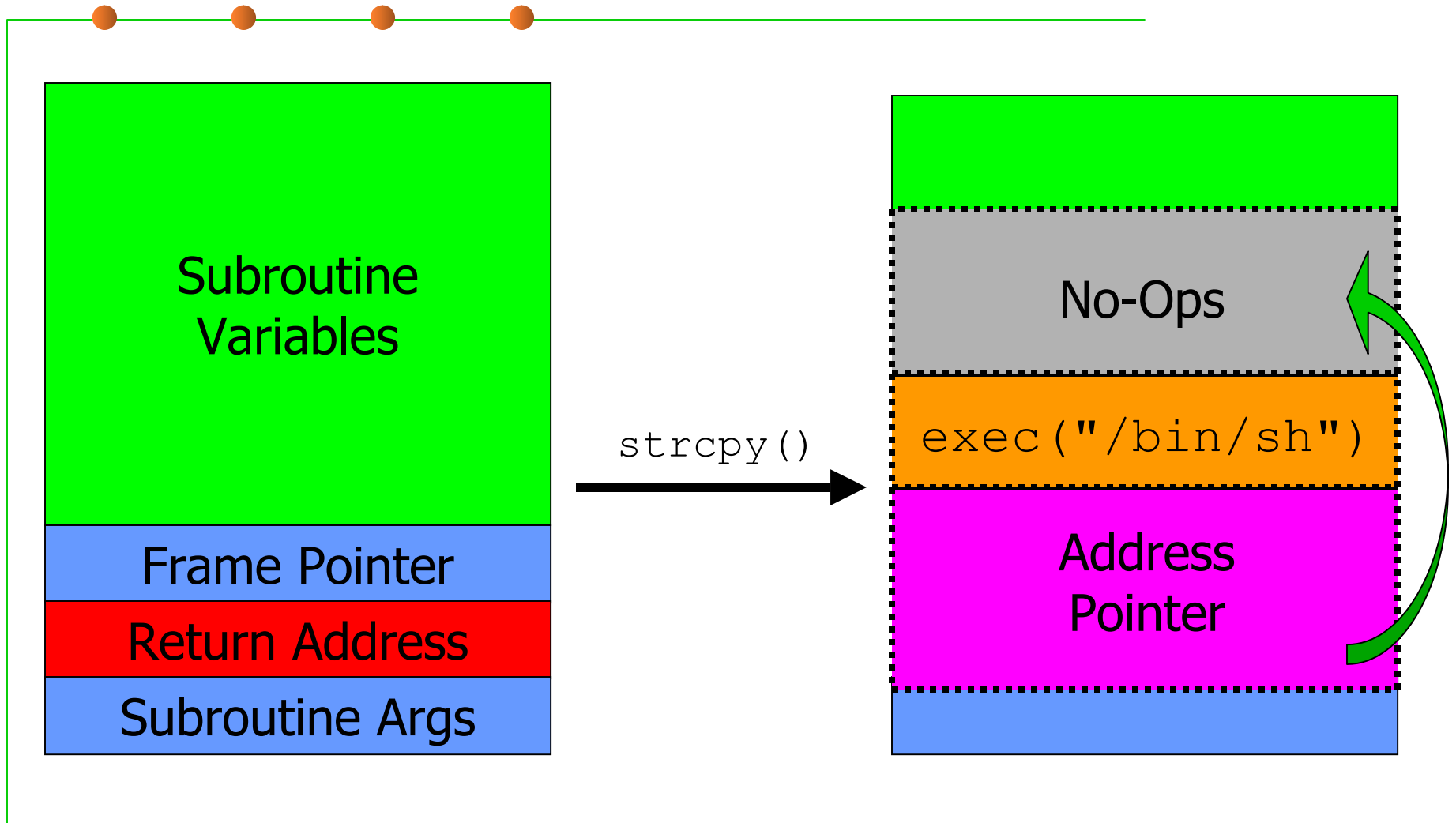




Classic Buffer Overflow


- 
- Attacker constructs a string containing:
 - No-ops for padding
 - Machine code to `exec("/bin/sh")`
 - Bogus instruction addr pointing into subroutine data area
 - Subroutine is coerced into copying this string into its data area, overwriting end of buffer
 - Subroutine exits and program follows bogus address to execute shell
 - If program runs as root, attacker gets root shell

Classic Buffer Overflow (2)

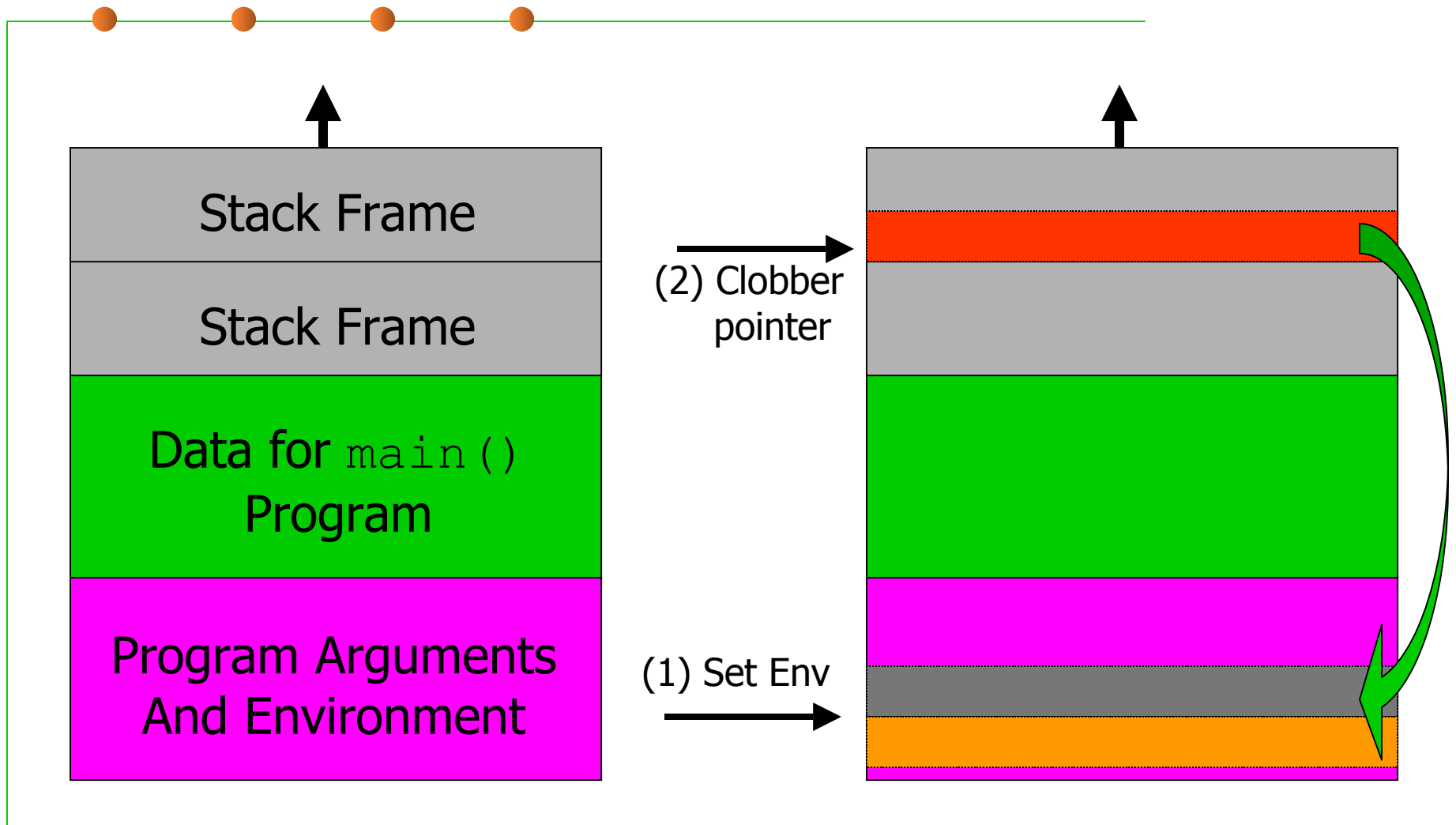




Modified Attack


- 
- Sometimes subroutine buffer is too small to hold exploit code
 - Exploit code can be put into an environment variable
 - Subroutine return address set to environment area at bottom of stack
 - Requires attacker have local machine access...

Modified Attack (2)





In General...

- 
- Attacker places code in program memory, changes address pointer to jump to code
 - Places to put code:
 - Stack (per previous examples)
 - Heap
 - Data/BSS (if writable)
 - Ways to modify address pointer:
 - Stack overflow (per previous examples)
 - Format string attacks (see next slide)

Format String Attacks


- Programmers are sometimes lazy:

```
printf("%s", str);           # correct  
printf(str);                # lazy
```

- If attacker can set `str`, then attacker can embed output specifiers
- In particular, `%n` writes a numeric value to a specified memory address
- Attacker can overwrite return address pointer or other numeric value




Fixing Buffer Overflows

- 
- Fix the programs
 - Too many
 - New bugs being written daily
 - Fix the programmers
 - Too many
 - New programmers being made daily
 - *Fix the stack*



Fixing the Stack

- 
- Code normally resides in read-only text area at beginning of program memory
 - Well-behaved programs should never execute instructions off data stack
 - Modify kernel— attempts to execute from stack cause program to abort
 - Requires cooperation of CPU hardware

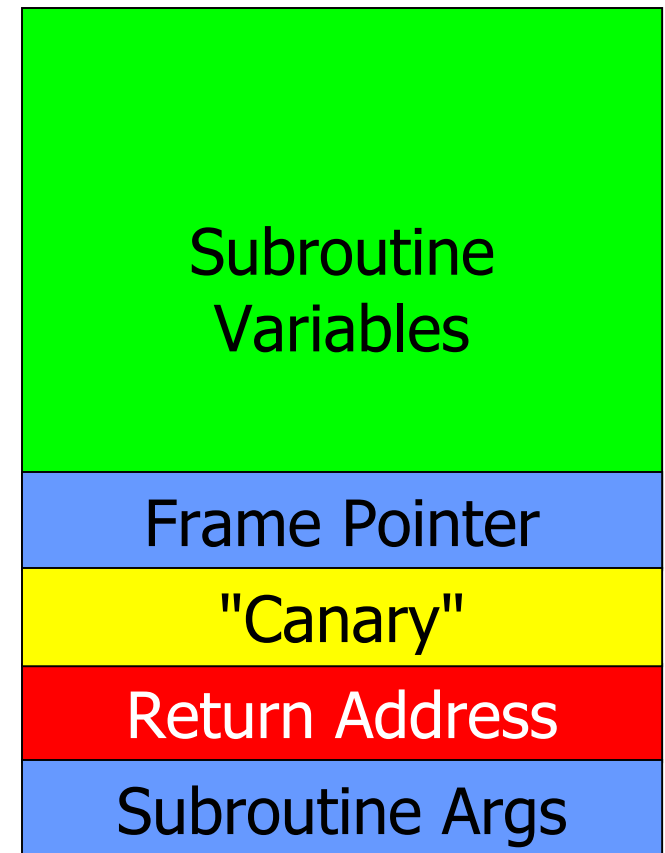


Different Implementations

- For Solaris, HP-UX, et al:
 - "Stack protection" only prevents executing code off of stack pages
 - Can be thwarted by putting malicious code into the heap area
- OpenBSD (W^X) and Adamantix (PaX):
 - All writable pages marked as non-executable (including heap area)
 - Can break applications written in high-level languages (notably Java)


Compiler-Based Solutions

- Compiler inserts a random "canary" value before address pointer
- Canary value checked when subroutine exits – abort if changed
- Attacks which clobber return address pointer also clobber canary
- Performance hit due to extra code to insert/check canary values





How to Defeat Stack Protection

- 
- Put malicious code into heap data area (at least for Solaris and HP-UX)
 - Force subroutine return to other call in text segment, like `system()`
 - Overwrite expected subroutine args:
 - `system()/exec()` call wrong program
 - `open()` opens unexpected file




After the Attack






Back Doors

- 
- Having broken a system, the attacker wants to make it easy to get back in
 - Often `sshd` or `telnetd` replaced, but any networked or SUID binary will do
 - New version gives root shell when special account and/or password used
 - Multiple back doors usually left behind to "fake out" system admins




rootkits

- 
- Nowadays, attackers want to install IRC servers, sniffers, DDOS tools, et al
 - Need much more intricate tools to cover their tracks
 - rootkits are pre-packaged bundles of software with tools and Trojans
 - New trend is rootkits that "harden" the system and remove other rootkits!




Hiding Files

- 
- `ls`, `dir`, `find`, etc. all replaced to hide attackers' files
 - Some rootkits use configuration files to change behavior "on-the-fly"
 - Binaries are crafted to have the same size (and same timestamps) as originals
 - Some files are installed "immutable" to make them harder to remove




Hiding Processes

- 
- `ps`, `top`, `lsdf` commands replaced with versions to hide certain processes
 - Also need to modify `kill`, `pkill`, `killall` and similar programs




Hiding Network Connections

- 
- New `syslogd` and `tcpd` will not log connections from certain hosts/domains
 - `ifconfig` will not report that network interfaces are in promiscuous mode
 - `netstat` will not show connections from certain hosts/domains




Problems for Rootkit Authors

- 
- Too many binaries to replace:
 - Hard work
 - Greater chance of detection
 - Might miss one
 - Checksumming (with AIDE or Tripwire) allows for easy detection




The Next Wave – Kernel Hacks

- 
- Suppose you could replace the underlying kernel routines?
 - Single modification corrupts every program on the system!
 - Modern Unix systems support dynamic kernel modules– no reboot required!
 - New kernel rootkits can even attack via `/dev/kmem`– no loadable modules!




What Do You Do?

- 
- Analyze system by first booting from OS CD-ROM
 - All tools should be brought over on read-only media (e.g., CD-ROM)
 - Reinstall system from scratch – don't necessarily trust your backups



And Don't Forget...

- 
- Assume any passwords on compromised system have been cracked off-line
 - If you believe packet sniffer was installed, then **all** site passwords must change




Wrap Up





That's All Folks!

- 
- Any final questions/comments?
 - Please fill out your eval forms!
 - Thanks for listening!

Plenty of useful URLs to follow...



General Resources

- SANS "Reading Room"
<http://www.sans.org/rr/>
- CERT/CC "Tech Tips"
http://www.cert.org/tech_tips/
- Center for Internet Security
<http://www.CISecurity.org/>
- BUGTRAQ Mailing List Archives
<http://www.securityfocus.com/>



Details on Stack Smashing

- The classic paper by Aleph1:
<http://www.phrack.org/show.php?p=49&a=14>
- Format string attacks
<http://www.securityfocus.com/guest/3342>
- Heap and BSS overflow paper:
<http://w00w00.org/files/articles/heaptut.txt>
- libc redirection attacks:
<http://hackersplayground.org/papers/stack.txt>



Other Resources

- Layered defense from OpenBSD 3.[34]:
<http://www.openbsd.org/papers/csw03.mgp>
- Compiler-based stack protection:
StackGuard – *<http://www.immunix.org/>*
IBM – *<http://www.trl.ibm.com/projects/security/ssp/>*
- Session hijacking tools:
Hunt – *<http://lin.fsid.cvut.cz/~kra/#HUNT>*
Ettercap – *<http://ettercap.sourceforge.net/>*



Rootkit Information

- 
- Links to detailed rootkit overviews:

<http://www.chkrootkit.com/>

http://biocserver.bioc.cwru.edu/~jose/shaft_analysis/node-analysis.txt

- Functional kernel rootkits (Linux):

<http://la-samhna.de/library/rootkits/list.html>

<http://www.phrack.org/show.php?p=58&a=7>