Hello, Interwebs

Hi, and thanks for reading this. As I mentioned a number of times during the talk this was one long, hard slog of a topic for me.

My intent was not to duplicate existing research (Johnson and Silberman @ BHUS05, others), but to try to make this topic comprehensible for the typical security professional, who (GASP! SHOCK! HORROR!) may not necessarily grasp all the hairy internals of exploit development, but still is tasked with protecting systems.

For the other 90% of us out there, our job is not to be leet, but rather not to get owned, something I hope to get a little bit better at every day. Since exploit mitigation is something that might bring us all a little bit closer to that, I wanted to explore the topic. Thanks much to BH for giving me the opportunity to do so, and to all of you for listening.

Thanks also to all the amazing people working on these technologies, especially the PaX team and Hiroaki Etoh of IBM.

-- shawn

P.S. It's actually Thompson that had the Phil Collins hair, not Ritchie. Sorry, Dennis.

28 75 6e 29 53 6d 61 73 68 69 6e 67 20 74 68 65 20 53 74 61 63 6b 0d 0a

(un)Smashing the Stack

4f 76 65 72 66 6c 6f 77 73 2c 20 43 6f 75 6e 74 65 72 6d 65 61 73 75 72 65 73
20 61 6e 64 20 74 68 65 20 52 65 61 6c 20 57 6f 72 6c 64

Overflows, Countermeasures and the Real World

Shawn Moyer :: Chief Researcher
---- SpearTip Technologies --->
blackhat [at] cipherpunx [dot] org

Hey, who is this guy?

ShawnM: InfoSec consultant, (quasi-) developer, husband, father, and raging paranoid with obsessive tendencies

Chief Researcher at SpearTip Technologies
Security Consultancy in Saint Louis, MO

Forensics, Assessment, MSSP, network analysis
Weddings, Funerals, Bar Mitzvahs

I like unsolvable problems, so I'm mostly interested in defense.
My hat color is... Fuschia.

#include std_disclaimer.h

I am not a ninja... I don't even play one on tv.

!=

Interest in exploit mitigation – no unified source of info.

Patching / IPS / vendor voodoo don't seem to work...

What is this stuff?

Exploit Mitigation

A range of compiler, OS, library, and kernel features
Intended to make successful exploitation more difficult

Primary aim: make mass exploitation less feasible
Limit exposure from memory corruption-based attacks

Nonexectuable stacks, ASLR, canaries, memory integrity
Compile-time and run time sanity checks for misuse
Bonus stuff: MAC models, static analysis, others

Why am I here, why are you listening?

We'll talk about

Attempt to deconstruct this topic for mere mortals

Some of the implementations out there

Complexity of the approaches

Plug some cool projects

Make this make sense for the Humans (us)

Let's just start at the beginning.

When dinosaurs roamed the earth...

This class of bugs has been around longer than (almost) everyone in this room.

Corruption of mismanaged memory space leads to control of execution flow. Hilarity ensues.

With us since at least the 1960s (!)
Intrinsic to Von Neumann model...
Code / Data abstraction, breaking the membrane

“Fandango on core”, “scribbling the stack”, “overrun screw”
“Stale / Dangling pointer bug” in ALGOL / FORTRAN lexicon

It used to be such a nice neighborhood…

Gene Spafford + RFC 1135, circa 1988:

"The Helminthiasis of the Internet"

Analysis of spread of the Morris worm

Stack-based overflow in fingerd gets() one vector

http://www.securityfocus.com/bid/2

(Trivia: Bid 1 was input val bug in Sendmail… Makes sense.)

In his analysis, Spaff warns against lack of bounds checking in C. Says gets(), scanf(), strcpy() and other unsafe calls should be avoided in the future.*

*Class, this will be on the final. Nineteen years is correct.

Meanwhile, back at the ranch...

Thomas Lopatic, circa 1995:

Hey, this is like the Morris worm fingerd bug...

Stack-based BoF found in NCSA HTTPd for HTTP GET

Mudge / L0pht, circa 1995:

Early walkthrough of process

How to construct shellcode, NOP sleds

Mesozoic Exploitation...

Aleph One, circa 1997:

Snapshot of attack landscape in late 90's ("eggs", NOPs)

Tutorial on memory segments, data/code/heap/bss/stack

Solar Designer, circa 1997:

First documented example I found of ret2libc

No shellcode, use preloaded function in memory

Cenozoic Exploitation…

Conover / w00w00, circa 1999:

“w00w00 on heap overflows”

Stack is but one target, grasshopper…

Shellcode written to the heap, fuction ptr overwrites

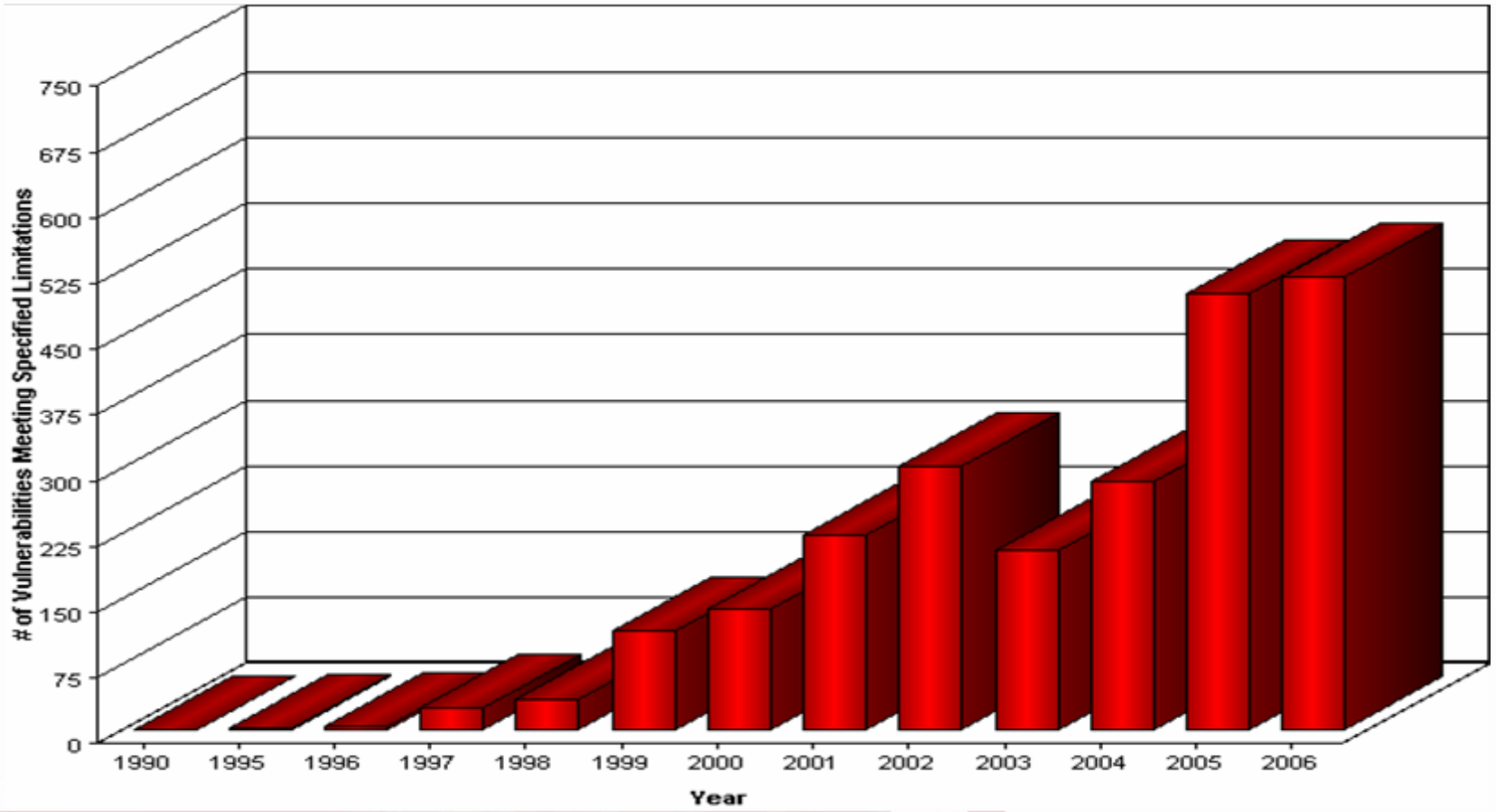
Okay, okay, enough already! Someone’s been reading the Phrack archives…

Why do you care?

Approaches evolved, but vectors are the same.

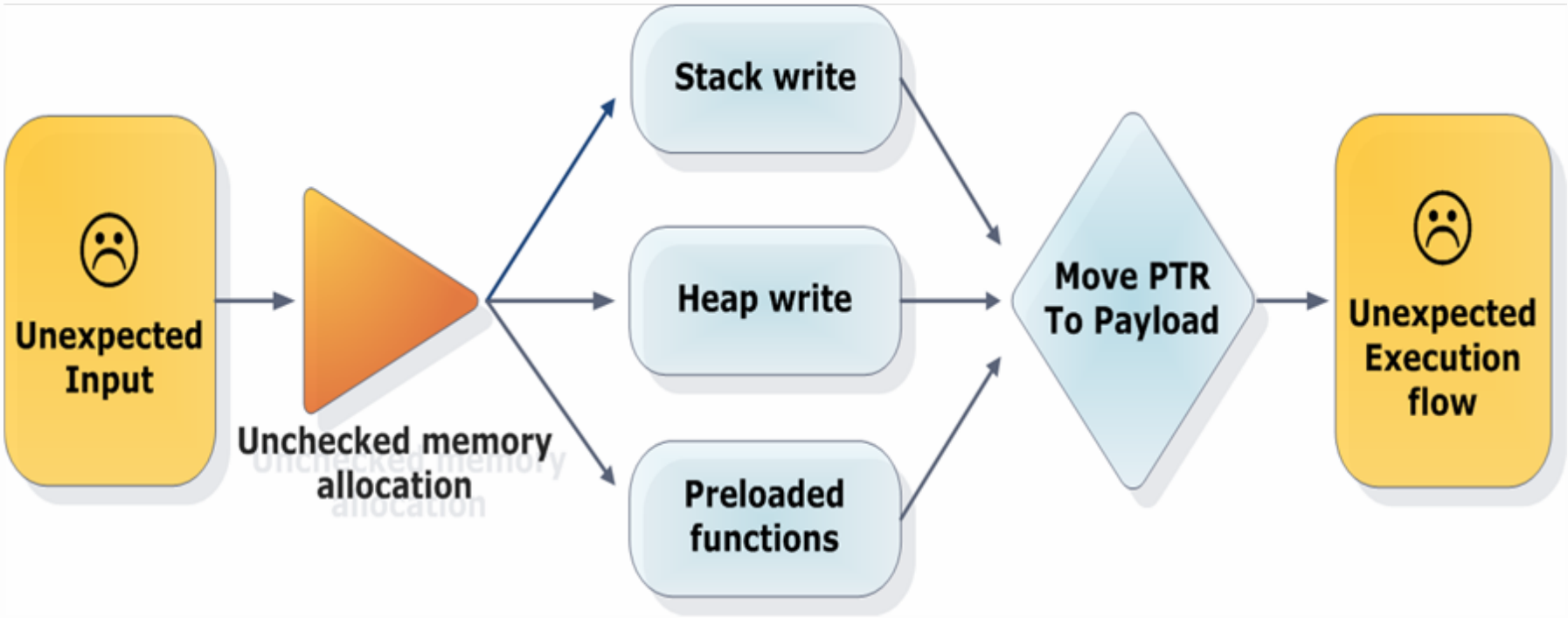
Ah... There's the ffwd button...

Things continue to get ugly. NVD remote BoF's up to end of 2006.



10 lbs. of crap in a 5 lb. bag

Okay, so obviously we all get this stuff, right?



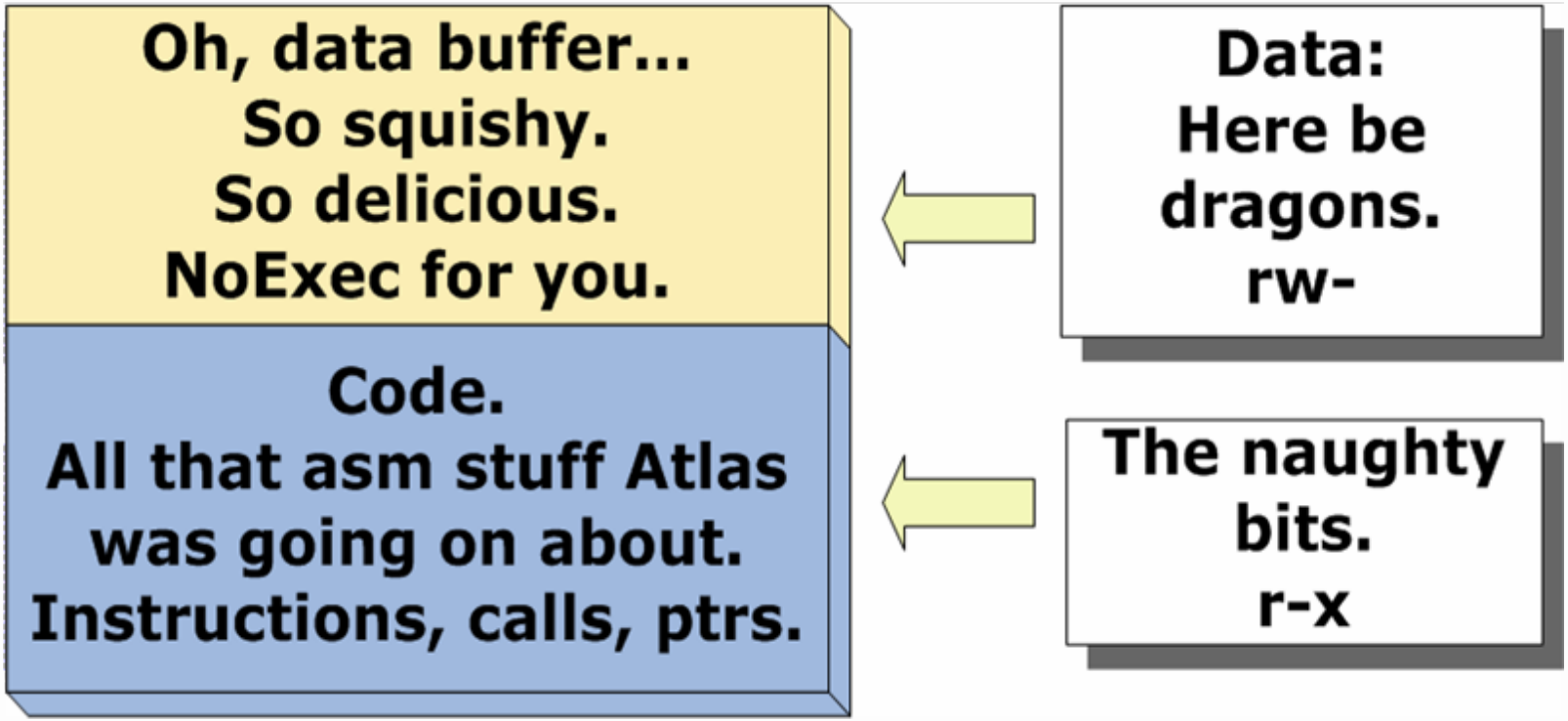
Brring, brring. “Cluephone. It’s for you.”

Hey…
Wait a minute.

Data is data, code
is code, right?

Eventually, the cluephone rings....

Gotta keep 'em se-pa-rated...



Brrring, brrring...

Solaris / uSPARC "noexec_user_stack" bit:

/etc/system "*no_exec_user_stack=1*" (or via ndd)

Similar implementations on HP/UX and other big iron

Nowadays, nX bit, DEP (XD? Meh.):

Via PAE (Page Address Extension) *et al*, bit 63 0/1

Requires software hooks! A CPU does not make you safer.

Brrring, brrring, BRRRRINGGG...

Emulation in software:

Solar Designer's StackPatch was the first.

W^X, ExecShield, PaX, DEP, others.

Software implementations (32-bit) are less fine-grained

Segment-based rather than page-based (line in the sand)

In general, nonexecutable stacks are problematic for apps that expect trampolines, JIT compilers, emulators, anything else that constructs code on the stack.

New countermeasures beget new attacks

Whoops! With a noexec stack, the vuln is still there, you've just reduced what's possible.

ret2libc:

First described by Solar in 1997

Most "0wn DEP / nX" talks / articles boil down to this

Call existing, preloaded function(s): *system()*, *execve()*, etc

Call mprotect() if OS honors it, then set allocation as rwx

Requires(?) known address, useful function in mem (hence libc)

Heap-based overflows:

Much more interesting now that noexec is everywhere

See Sotirov's talk, Justin of IOA, atlas' DC talk, Ollie W.

nX counter-countermeasures (cont.)

Piromposa / Enbody:

"Hannibal attack" or multistage overflow

Overwrite function pointer to an arbitrary address

Insert shellcode at predefined address via *argv()*, etc

Skape / Skywing:

Forcible OptOut for process in MS's DEP via ret2libc

DEP configurable via ProcessExecuteFlags at runtime

MEM_EXECUTE_OPTION(ENABLE|DISABLE)

Summary: set "/noexecute=AlwaysOn" in boot.ini(!)

PaX uses file header instead. Higher bar?

nX counter-countermeasures (cont.)

"Opt in / out" ugliness:

All implementations (DEP, gcc) allow disabling of checks via mprotect(), VirtualProtect(), -fno-stackprotector, /O

Tunable via ELF header with chpax

Skywing's NoExecute Hall of Shame:

http://www.nynaeve.net/?p=135

Personally, DEP / noexec has killed:

Firefox, Acrobat, Java…

BackupExec w/ SecureStack (interesting story…)

Canary in a coalmine: Tripwire FTW!

Okay then, why not just check for memory corruption?

Checksums for the stack!

Canary in a coalmine

StackGuard:

Initial approach. Crispin Cowan / Immunix in ’99

“Canary” word in stack for RTA in *function_prologue*

If canary word changed on *function_epilogue*, call *exit()*

Later versions improved layout, different canary approaches

Cheep, cheep...

ProPolice / SSP:

Hiroaki Etoh of IBM is teh awesome+++

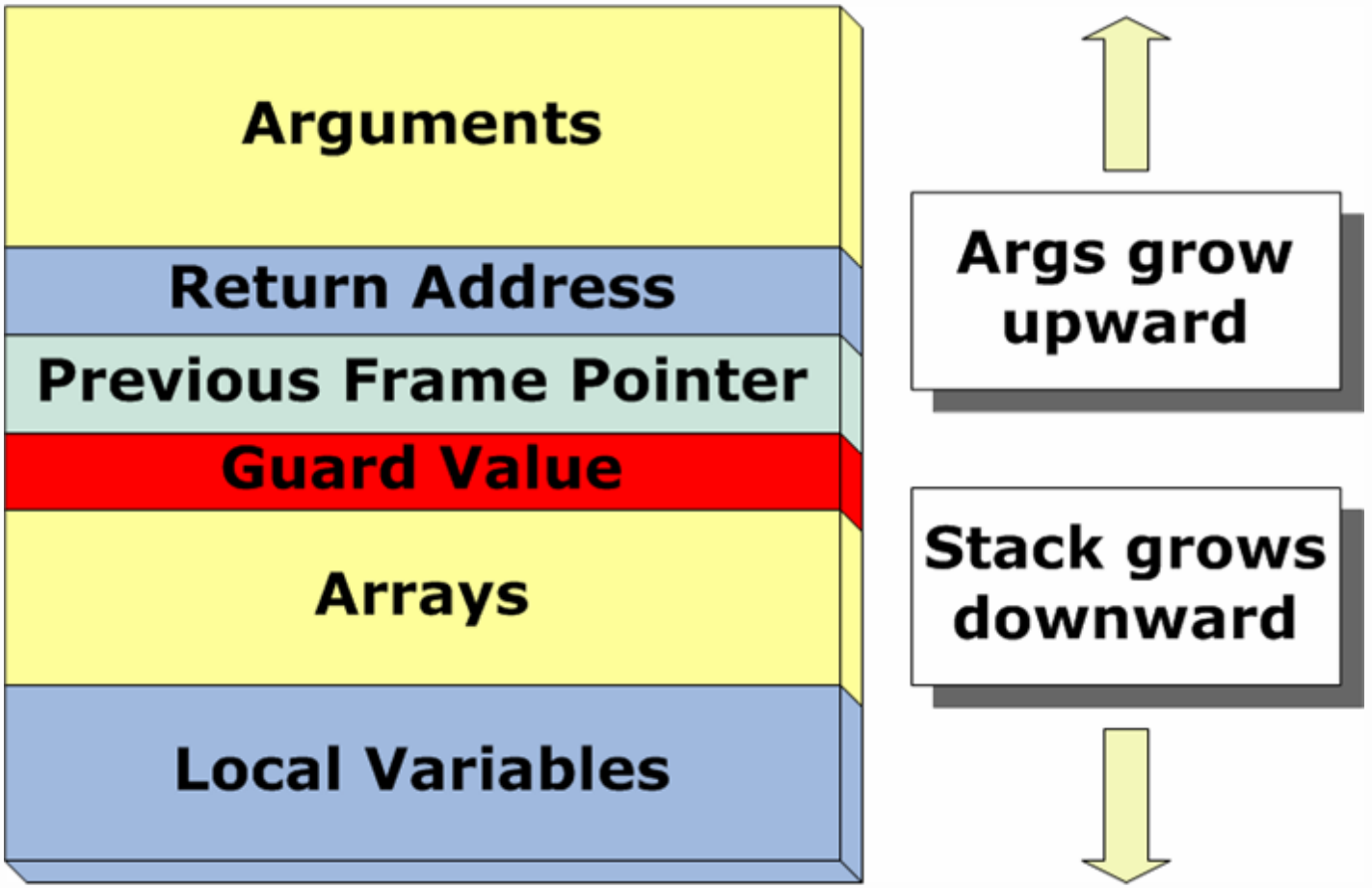
Integrated into gcc > 4.1, backports for 3.x in OBSD
"Embraced and extended" by MS as /GS compiler flags
GS "cookie" is ProPolice guard value

Not exactly a canary approach. Hiroaki prefers "guard value"
XOR of ptr and random value, stored off-stack

Move beyond just return address to all registers, in prologue
Well-ordered stack: variables and arrays at bottom, args at top

Cheep, cheep (Propolis)…

ProPolice Safe Stack



Canary interlude: (un)Smashing the Heap!

Heap Canary implementations:

Same canary principle has been applied to protect the heap

Guard value / canary, damage to guard triggers exception

OpenBSD “G” option to malloc.conf

ContraPolice

http://synflood.at/contrapolice.html

WKR -> dlmalloc

http://www.cs.ucsb.edu/~wkr/projects/

Hey, who ate my canary?

If the canary is the key element in protecting against memory corruption, the defense is only as good as the canary.

Gerardo Richarte / CoreSec:

Global Offset Table writes “after” return address, SFP stuff

Specific to StackShield, /GS, StackGuard

Phrack 56 / Bulba and Kil3r:

Noted StackGuard weaknesses in canary for only return pointer

Write pointer elsewhere in unbounded strcpy / memcpy

Overwrite return pointer without harming canary

Hey, who ate my canary? (cont.)

Other approaches:

Canary is somewhere in (readable) memory, right?

(random XORed canary or off-stack approach helps here)

Arbitrary memory reads could disclose canary value

Format string bugs, /proc/mem/, info leakage

Obfuscation and complexity

Hey…
Let's shuffle memory around!

See: Art of War. Misdirection.

Obfuscation and complexity

PaX:

Randomized userland, kstack, mmap()

Tunable knobs in kernel / binaries via paxctl / chpax

OBSD > 3.3:

Randomized malloc(), mmap(), “stack fencing”, “stack gap”

Similar bits for NetBSD and FBSD are floating around as well…

ExecShield:

RedHat’s own NoExec implementation: RHEL/Fedora

Also does stack and mmap() base randomization

Can someone tell me why they didn’t just use PaX???

Obfuscation and complexity (cont.)

Vista:

Random .exe and .dll loader. Varying degrees of entropy (ahem)

For apps to utilize internally, requires */dynamicbase*, VS 2005 SP1

Weak on the heap, as noted by WhiteHouse (BHDC07)

Obfuscation: Intermission

PIC or PIE: Position-Independent (Executables|Code)

Execute properly and safely, regardless of location
Find the GOT, and go somewhere random via ASLR

John Moser of Ubuntu-Hardened estimates success rate of
1 in 2^(STACK_RANDOM_BITS + MMAP_RANDOM_BITS)

PIC/PIE/dynamicbase is key to full ASLR implementation
If app itself uses predictable addresses, ASLR is moot

Elegant solution, meet brute force...

The bar is raised, but... We're only as good as our randomization.

Hovav Shacham et al:

"Derandomization attack": brute-force the system() location

Especially applicable to forking services that respawn

New stuff: ret2libc with no function calls (chaining sequences)

Homework: Shacham on the heap in Vista? Client-side?

More brute force...

Ben Hawkes / SureSec:

"Code access brute forcing": use unsuccessful reads to infer mapping / position, look for known-size libs (ret2libc)
(apps using prelinked libs can also leak locations)

Whitehouse / BHDC07:

Vista found to be inconsistent in randomization, esp. heap
256 possible values, Shacham et al may be applicable?

Exploit mitigation: Cliff's Notes

Noexec / NX:

If runtime configurable, it's useless

Oodles of other exploitation methods

ASLR:

Bad crypto is not a panacea

Memory leaks bugs will break the model

Canaries:

Bad crypto is not a panacea

All writes to memory space requre protection

Best practice today: Mesh model / Sec in depth.

Aggregate of all of these = a modicum of safety

Other ways to skin a cat

We're just getting started, nineteen years later.

What else can we do to raise the bar?

Other ways to skin a cat

Okay, how about we just fix the $@%# code?

Look for likely exploitable vectors, via static code analysis

GCC's FORTIFY_SOURCE:

Identify commonly misused functions

Replace with better alternatives, strcpy with strncpy, etc

Includes some checks for format string stuff…

Coverity / DHS joint project for OSS bearing fruit

Scads of commercial tools (insert your vendor here)

Rice's Theorem, Rumsfeld's corollary

Skinning a cat and tanning the hide

Okay, so does my web server really need to spawn a reverse shell and cat /etc/shadow?

Access Control Models:

Not exploit mitigation but containment post-exploit
Restrict file, device, inode access based on UID or other criteria

Linux:

RSBAC, GRSecurity, AppArmor, SELinux, others
Varying levels of complexity and maintenance

MS:

Vista's Mandatory Integrity Control (Click "Allow","Allow", etc)

Tanning the hide and sewing on buttons

Big Iron Unix:
TrustedSolaris, HP/UX C2 Trusted mode

Little Iron Unix:
TrustedBSD, integrated into FBSD > 5.0, OpenBSD's systrace

Mac has MAC:
Originally announced as part of Leopard @ WWDC
Check out SEDarwin: **http://www.sedarwin.org**

Rubber, meet road.

OpenBSD
Extensive work with ProPolice, W^X, various ASLR stuff
Mprotect () works, no rand kstack (?), no noexec kstack

FreeBSD
Very basic NX integration, extensions for SSP / ASLR at:
http://tataz.chchile.org/~tataz/FreeBSD/SSP/

NetBSD
Moving SSP, OpenBSD, PaX-inspired stuff into 4.0
http://www.securityfocus.com/infocus/1878

Rubber, meet road.

The Linuces

GCC 4.1 means everyone has *some* SSP / ProPolice

Ubuntu pushing for –fstackprotector-all where possible, PaX integration, other stuff

Hardened Gentoo probably the most thorough and active

Lots of consistency issues… Search any vendor for SSP bug reports.

Rubber, meet road.

Vista:
The least explosive Ford Pinto, ever!
ASLR, PIC/PIE, MIC, NX, other stuff
Consistency is the biggest issue today

2003/XP:
Not much of consequence. DEP/NX, /GS with no ASLR
See **http://www.wehnus.com**

Rubber, meet road.

OSX. **Think obscurity.**
Zip. Squat. Zed. Zilch. Nada. No-thing.
Yeah, okay. Kern-level NX.
MAC in Leopard

Belt? Check. Suspenders? Check.

All defensive measures can be defeated in a vaccuum

Vulnprog.c will always get 0wned.
Sum of the whole is greater than its parts.
We hope.

Check out: PaXTest, VistaProbe

Thanks: DT, DA, Dragonxhero, Mom, Alpha and Zed, and most of all you guys!