Big Game Hunting_

Simple techniques for bug hunting on big iron UNIX

```
adversary@your.domain.tld:~$ ln -s /important /tmp/backup.log
adversary@your.domain.tld:~$ sudo ./backup.sh
adversary@your.domain.tld:~$ ls -la /important
-rw-rw-rw- 1 root root 1798 Aug  2 10:39 /important
```

# whoami_

- Tim Brown
- @timb_machine
- Head Of Research at Portcullis Computer Security Ltd
- http://www.nth-dimension.org.uk/

# last_

- >15 years of UNIX experience
- Background in telcos and finance
- 9 years at Portcullis
- More at http://44con.com/speaker/tim-brown/

# cat .plan_

- Auditing
  - Problems
  - Solutions
- Going further
  - Why?
  - The attack surface
  - In the real world
- In the lab

# Auditing_

- Problems
- Solutions

# Problems_

- Limited access
- Varying OS capabilities
- Multiple solutions
- Differences in requirements

# Limited access_

- Client doesn't own the system
- Client doesn't want to give (root) access
- System is physically unavailable
- System is a black box

# Varying OS capabilites

- Standards leave elements undefined
- OS tool chain not sufficient
- * GNU/Linux moves much faster than commercial OS
- Solaris 10 (much) > Solaris 8

# Multiple solutions_

- How do you lock an account?
  - passwd -l?
  - Change the shell?
  - Etc...
- If you don't run sendmail, should the configuration still be hardened?

# Differences in requirements_

- Which audit methodology do you use?
  - Vendors?
  - US DoD?
  - CIS?
  - Etc...
- What if they differ significantly?
- Would you know?

# Solutions_

- Better scripts
- Gap analysis
- C(ommon) C(onfiguration) E(numeration)
- Smarter humans

# Gap analysis_

- We probably need to know what different methodologies check for
- I wish someone else had done it

# C(ommon) C(onfiguration) E(numeration)_

- They have (kinda):
  - http://cce.mitre.org/
- Incomplete
  - Missing various OS
  - Not sure I agree with their methodology
    - No mention of gap analysis (AIX guy may not know Solaris and vice versa)
    - They consider outcome, not technique

# Smarter humans_

- I don't scale well!
- We *all* need training when it comes to stuff we don't see every day
- Maybe talks like this will help DevOps get their shit together?

# Going further_

- Why?
- The attack surface
- In the real world

# Why?_

- Bug hunting
- More importantly, auditing fails to answer the hard question – did you want segregation of roles with that?

# The attack surface_

OS	Kernel	Services	
Enterprise apps	Services	Batch jobs	User roles
DevOps	Batch jobs	User roles	
Users	Misfortune	Malice	

- If “everything is a file”, we need to get better at analysing the files...

# In the real world_

- The OS should protect us from ourselves
- Enterprise applications continue accumulate features
- DevOps will replace us all with shell scripts

# OS flaws_

- Bad standards
- Forks
- Poor defaults
- Incorrectly implemented separation of privileges
- Poorly implemented administrative functionality
- Incomplete anti-exploitation mitigations

# Examples_

- Shared code such as CDE
- Binaries owned by “bin” user
- Binaries such as telnet and ftp being SetUID
- WPAR isolation
- Patching may be the problem, not the solution

# Anti-exploit mitigations_

Mitigation	* GNU/Linux	AIX
Mandatory access control	Y	N (Y in Trusted AIX)
Non-executable stack	Y	N (select mode by default)
ASLR	Y	N
Hardened malloc()	Y	N (Y with Watson malloc())
Stack cookies and other compile time mitigations	Y (glibc)	N
mmap() NULL	N	N

# Non-executable stack?_

```
# sedmgr
Stack Execution Disable (SED) mode: select
SED configured in kernel: select
# find / -perm -u+s -exec sedmgr -d {} \; | grep -v system
/opt/IBMinvscout/sbin/invscout_lsvpd : Not a recognized executable format.
#
```

# ASLR?_

```
# ./aslr
REMOVE
system() = f1ab5d70
bos.rte.libc                 6.1.3.11        ROOT        REJECT       SUCCESS
bos.rte.libc                 6.1.3.11        USR         REJECT       SUCCESS
ADD
system() = f1c05490
bos.rte.libc                 6.1.3.11        USR         APPLY        SUCCESS
bos.rte.libc                 6.1.3.11        ROOT        APPLY        SUCCESS
REMOVE
system() = f1d4bd70
bos.rte.libc                 6.1.3.11        ROOT        REJECT       SUCCESS
bos.rte.libc                 6.1.3.11        USR         REJECT       SUCCESS
ADD
system() = f1e9b490
bos.rte.libc                 6.1.3.11        USR         APPLY        SUCCESS
bos.rte.libc                 6.1.3.11        ROOT        APPLY        SUCCESS
```

# Hardened malloc()_

- Check out David Litchfield's paper “Heap overflows on AIX 5”
- Also, “Enhancements in AIX 5L Version 5.3 for application development” mentions a number of enhancements / possible areas of concern

# Hardened malloc() ++_

```
$ ls -la malloc
-rwsr-xr-x    1 root     system         53648 Sep 04 22:41 malloc
$ MALLOCTYPE=watson
$ export MALLOCTYPE
$ ./malloc
blah
$ MALLOCDEBUG=catch_overflow ./malloc
Segmentation fault
```

# Enterprise "features"_

- Data
  - The real value of your system
- "Interesting" code
  - More code is always bad, but OS code at least benefits more from the "many eyes" principal – assuming the "many eyes" are actually looking – your enterprise app may not

# “Interesting” code_

- Backdoors
- Proprietary protocols
- Embedded library copies
- Changes to user environment
- Insecure API usage
- Missing anti-exploitation techniques
- Key material and entropy
- Java :-)

# Practising unsafe DevOps_

- Build infrastructure
- Cron, cron, cron
- .rhosts
- Sudo :-)
- Init and inetd
- User provisioning and access management
- Key material
- NFS

# Cron, cron, cron_

- Your shell script just ran over my shadow

```
# grep victim /var/spool/cron/crontabs/*
/var/spool/cron/crontabs/root:0 01 * * * /opt/victim/start.sh
# cat /opt/victim/start.sh
...
umask 000
OUTDIR=/tmp
...
service=/opt/victim/service
...
OUTFILE="${OUTDIR}/${DATE}_${TIME}.log"
...
$service -o ${OUTFILE}
```

# In the lab_

- Systems
- Books
- Code
- Tools
- Techniques

# Systems_

- Buy or emulate the systems you see in the wild
- Better still, buy or emulate those you don't – they're still there!

# Books_

- If you understand how one OS works, the next OS you look at might just work in a similar way (with similar bugs / different edge cases):
  - Vendor web sites
  - Man pages
  - Solaris Systems Programming and Solaris Internals are great books

# Code_

- Next time code leaks, take a look, your adversaries will
- Identify lists like oss-security, fewer size contests mean more signal and less noise
- .jar files are human readable

# Tools_

- # strings and grep
- # truss and strace
- # DTrace and SystemTap
- # objdump, GDB and IDA
- # jad, JD-GUI and friends
- # Compilers
- # checksec.sh (for * GNU/Linux)
- # unix-privesc-check

# Techniques_

- Sometimes the same crash on another OS yields greater joy – the Solaris stack for a certain RPC service isn't munged
- SetUID binaries can often be exploited via obscure enviroment variables – ++ local roots for IBM products :)
- Old techniques can be reapplied – glob() style bugs still afflict AIX

# Techniques ++_

- Auditing (the other type) will catch stuff you might miss
- Decompile .jar files
- Check what libraries $enterpriseapp ships with (don't forget to check for embedded JVMs)

# Techniques ++_

- Check against Microsoft's banned API list
- Check for anti-exploitation mitigations
- DT_RPATH AKA Import File Strings

# DT_RPATH AKA Import File Strings_

```
# dump -Hv kbbacf1

kbbacf1:

                        ***Loader Section***
                      Loader Header Information
VERSION#          #SYMtableENT      #RELOCent         LENidSTR
0x00000001        0x0000000f        0x0000001c        0x000000b5

#IMPfilID         OFFidSTR          LENstrTBL         OFFstrTBL
0x00000007        0x000002d8        0x00000063        0x0000038d


                        ***Import File Strings***
INDEX  PATH                              BASE                   MEMBER
0      /usr/lib:/lib::/opt/IBM/ITM/tmaitm6/links/aix51/lib:.:./lib:../lib::
```

# unix-privesc-check_

- Originally conceived by @pentestmonkey
- I'm working on 2.x
  - Code will be made real soon now!

# Conclusions_

- Ask yourself "who analysed the OS?"; "do I care about segregation of roles?"; "do I know what my applications are doing?"; "do I care what my DevOps teams are bringing to the party?"
- If these questions matter, don't audit, whitebox

# Questions_

< /dev/audience