



The bridge to possible

Auditd for the recently threatened

Tim (Wadhwa-)Brown

Security Research Lead, CX Technology & Transformation Group

May 2022

Background

- Why listen to me?
 - Designed and built detection capability for banks and telcomms
 - Assessed and provided input and training for OT developers, engineers and operators building, implementing and operating PLCs, HMIs etc
 - Responded to breaches on all manner of weird and wacky
- Recap from previous ATT&CK Community events
 - [All of the threats - Intelligence, modelling, simulation and hunting through an ATT&CKers lens](#)
 - [The UNIX malware landscape - Reviewing the goods at MALWAREbazaar](#)

Reacting to threats, both new and old

- New
 - T1036: Masquerading
 - T1070: Indicator Removal on Host
 - T1205: Traffic Signaling
- Old
 - T1005: Data from Local System
 - T1083: File and Directory Discovery
 - T1003: OS Credential Dumping
 - T1558: Steal or Forge Kerberos Tickets

Malware example

- BPFDoor
 - Writes a 0 byte file to /var/run
 - `// /var/run/haldrund.pid`
 - `close(open(pid_path, O_CREAT|O_WRONLY, 0644));`
 - Writes, executes and deletes from /dev/shm
 - `// /bin/rm -f /dev/shm/%s;/bin/cp %s /dev/shm/%s && /bin/chmod 755 /dev/shm/%s && /dev/shm/%s --init && /bin/rm -f /dev/shm/%s`
 - `snprintf(cmd, sizeof(cmd), fmt, tmp, name, tmp, tmp, tmp, tmp);`
 - `system(cmd);`
 - Time stomps /dev/shm/kdmtmpflush
 - `utimes(file, tv);`
 - Uses raw sockets
 - `sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP))`
 - Sets a BPF filter
 - `setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &filter, sizeof(filter))`
 - Executes commands
- Building detections
 - `strace -f -o bpfdoor.out ./bpfdoor`

File access

- `egrep "/var|/dev" bpfdoor.out | egrep "access|open|unlink"`
 - `access("/var/run/haldrund.pid", R_OK) = -1 ENOENT (No such file or directory)`
 - `-w /run/haldrund.pid -p rwx -k tb_run_haldrund_pid_bpfdoor`
 - `unlinkat(AT_FDCWD, "/dev/shm/kdmtmpflush", 0) = -1 ENOENT (No such file or directory)`
 - `openat(AT_FDCWD, "/dev/shm/kdmtmpflush", O_WRONLY|O_CREAT|O_EXCL, 0755) = 4`
 - `fchmodat(AT_FDCWD, "/dev/shm/kdmtmpflush", 0755) = 0`
 - `-w /dev/shm/kdmtmpflush -p rwx -k tb_dev_shm_kdmtmpflush_bpfdoor`

Command execution

- `egrep "/var|/dev" bpfdoor.out | egrep "exec"`
 - `execve("/bin/rm", ["/bin/rm", "-f", "/dev/shm/kdmtmpflush"], 0x563147b63af8 /* 24 vars */ <unfinished ...>`
 - `execve("/bin/cp", ["/bin/cp", "./bpfdoor", "/dev/shm/kdmtmpflush"], 0x563147b63b00 /* 24 vars */ <unfinished ...>`
 - `execve("/bin/chmod", ["/bin/chmod", "755", "/dev/shm/kdmtmpflush"], 0x563147b63b00 /* 24 vars */ <unfinished ...>`
 - `execve("/dev/shm/kdmtmpflush", ["/dev/shm/kdmtmpflush", "--init"], 0x563147b63ad8 /* 24 vars */ <unfinished ...>`
 - These are all a bit generic in this case, but in theory we could tap into the `execve` syscall
 - `-a exit,always -F arch=b64 -S execve -k tb_exit_b64_execve_syscall_bpfdoor`
 - Sadly we can't filter on `a0` etc as strings, but we could in the SIEM

Other file operations

- `egrep "/var|/dev" bpfdoor.out | egrep -v "exec|access|open|unlink"`
 - `utimensat(AT_FDCWD, "/dev/shm/kdmtmpflush", [{tv_sec=1225394236, tv_nsec=0} /* 2008-10-30T19:17:16+0000 */, {tv_sec=1225394236, tv_nsec=0} /* 2008-10-30T19:17:16+0000 */], 0) = 0`
 - `-a exit,always -F arch=b64 -S utimensat -F a0=AT_FDCWD -k tb_exit_b64_utimensat_syscall_bpfdoor`
- But... `-F` can only take numbers for `a0`, `a1`, `a2`, `a3`
 - `egrep -r "AT_FDCWD" /usr/include`

Raw sockets

- `egrep "socket|setsockopt" bpfdoor.out`
 - `socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP) <unfinished ...>`
 - `-a exit,always -F arch=b64 -S socket -F a0=AF_PACKET -F a1=SOCK_RAW -k tb_exit_b64_socket_syscall_bpfdoor`
 - `setsockopt(3, SOL_SOCKET, SO_ATTACH_FILTER, {len=30, filter=0x7fff628b97f0}, 16) = 0`
 - `-a exit,always -F arch=b64 -S setsockopt -F a1=SOL_SOCKET -F a2=SO_ATTACH_FILTER -k tb_exit_b64_setsockopt_syscall_bpfdoor`
- Remember, `-F` value should be number for `a0` etc

Finalised rules to detect BPFDoor

```
-D  
-w /run/haldrund.pid -p rwx -k tb_run_haldrund_pid_bpfdoor  
-w /dev/shm/kdmtmpflush -p rwx -k tb_dev_shm_kdmtmpflush_bpfdoor  
-a exit,always -F arch=b64 -S utimensat -F a0=-100 -k  
tb_exit_b64_utimensat_syscall_bpfdoor  
-a exit,always -F arch=b64 -S socket -F a0=17 -F a1=3 -k  
tb_exit_b64_socket_syscall_bpfdoor  
-a exit,always -F arch=b64 -S setsockopt -F a1=1 -F a2=26 -k  
tb_exit_b64_setsockopt_syscall_bpfdoor
```

Evaluation and tuning

```
# auditctl -R audit.rules && ./bpfdoor && systemctl stop auditd
```

```
# grep bpfdoor /var/log/audit/audit.log | grep -v add | grep shm | wc -l
```

9

```
# grep bpfdoor /var/log/audit/audit.log | grep -v add | grep run | wc -l
```

1

```
# grep bpfdoor /var/log/audit/audit.log | grep -v add | grep setsockopt | wc -l
```

1

```
# grep bpfdoor /var/log/audit/audit.log | grep -v add | grep socket | wc -l
```

1

- `tb_exit_b64_utimensat_syscall_bpfdoor`
 - `-F exe="/dev/shm/kdmtmpflush"`
 - `-F comm="kdmtmpflush"`
 - You could also potentially exclude known trustworthy processes
- `tb_exit_b64_socket_syscall_bpfdoor`
 - `-F exe="/dev/shm/kdmtmpflush"`
 - `-F comm="kdmtmpflush"`
 - You could also potentially exclude known trustworthy processes
 - `-F a0=3`

Detecting forwards

Ideas for detecting forwards

- General

- /dev
- /tmp, /var/tmp, /dev/shm
- /etc, /var writes from non-root
- /proc, /sys writes from non-root
- `grep -r __SYSCALL /usr/include | cut -f 2 -d "(" | cut -f 1 -d , | sort | uniq | grep NR`
 - `ptrace()`
 - `set[ug]id()`
 - `*chown()`
 - `*chmod()`
 - `mmap()`
 - `mprotect()`
 - `memfd_*`
 - `unshare()`
 - Lots more...

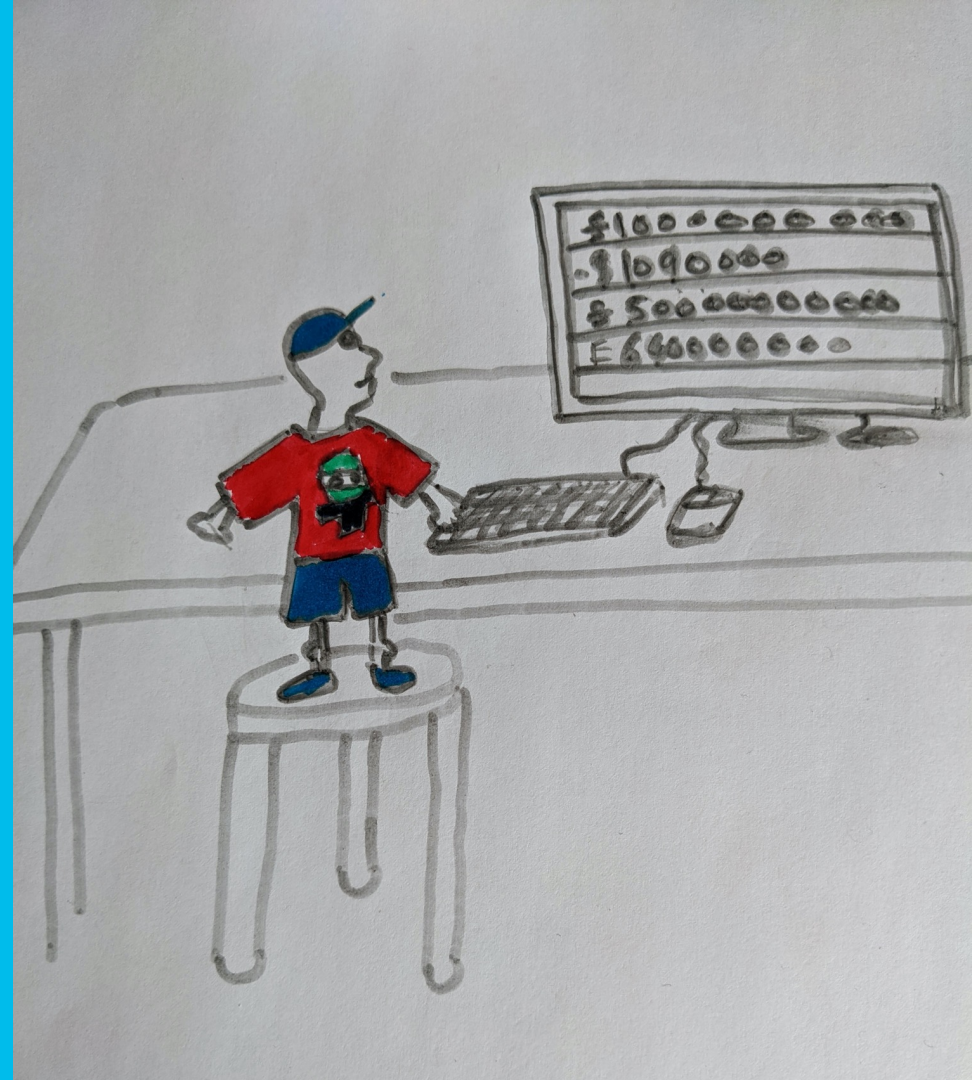
- Daemons

- `write()`
- `bind()`
- `connect()`
- `execve()`

- Users

- /etc/passwd
- /etc/shadow
- /etc/groups
- /home/*/.ssh
- /etc/sudoers
- /etc/sudoers.d
- `execve()` on GTFObins

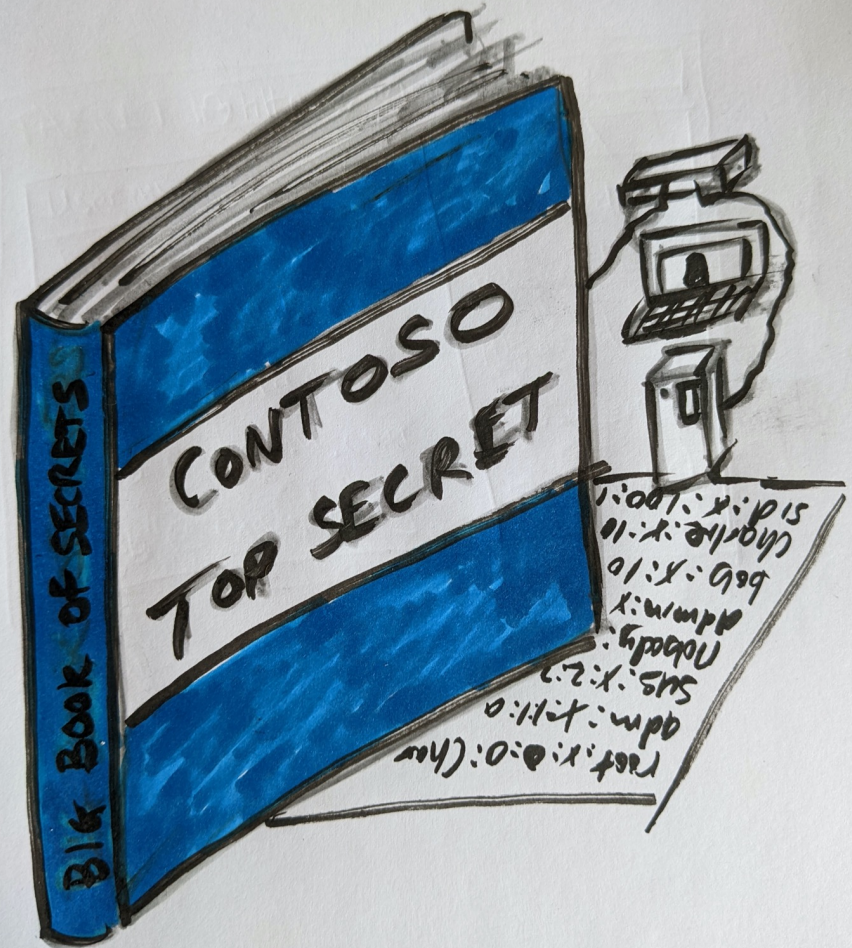
Managing (technical) debt



How did I defend against it?

- ACLs and auditing
- Scripting the generation of an auditing policy and bespoke ACLs based on the output of `find`
- Detection
 - DS0017: Command
 - DS0009: Process

Preparing for Black Hat



How should you defend against it?

- Check the syscalls
- Check file access
 - `-a always,exit -F dir=/var/lib/sss/db -F perm=rwxa -k linikatz-sss`
- Look for static numeric values to match on
 - Constants
 - Size parameters
 - `-a always,exit -F arch=b64 -S connect -F a2=0x2f -k linikatz-vas`
- Detection
 - DS0017: Command
 - DS0022: File
 - DS0009: Process

Conclusions

Auditd crib sheet

- A subset of events will be generated without configuration
 - Don't mistake this for a useful configuration
- Protect the daemon
 - -e 2
- What happens in kernel stays in kernel
 - *entry* && *exit*
- There may be event subsets you don't care about
 - always || never && exclude
- Pick the real path for file system operations
 - Operations on files beneath a symlink won't be logged
- Fine tuning (-F) can help
 - Consider architecture
 - -F arch=b32 vs -F arch=b64
 - Watch by user, process
 - -F auid=<uid>
 - -F pid=<pid>
 - -F ppid=<ppid>
 - etc
 - Sadly you can't match on strings
 - Filenames, syscall specific constants and length arguments may still be useful
- Combine rules where you can
 - The filesystem rules for BPFDoor for example
- Tag your rules (-k) to help your analysts

Questions?

twadhwab@cisco.com



The bridge to possible

Bonus material

Why look at Auditd?

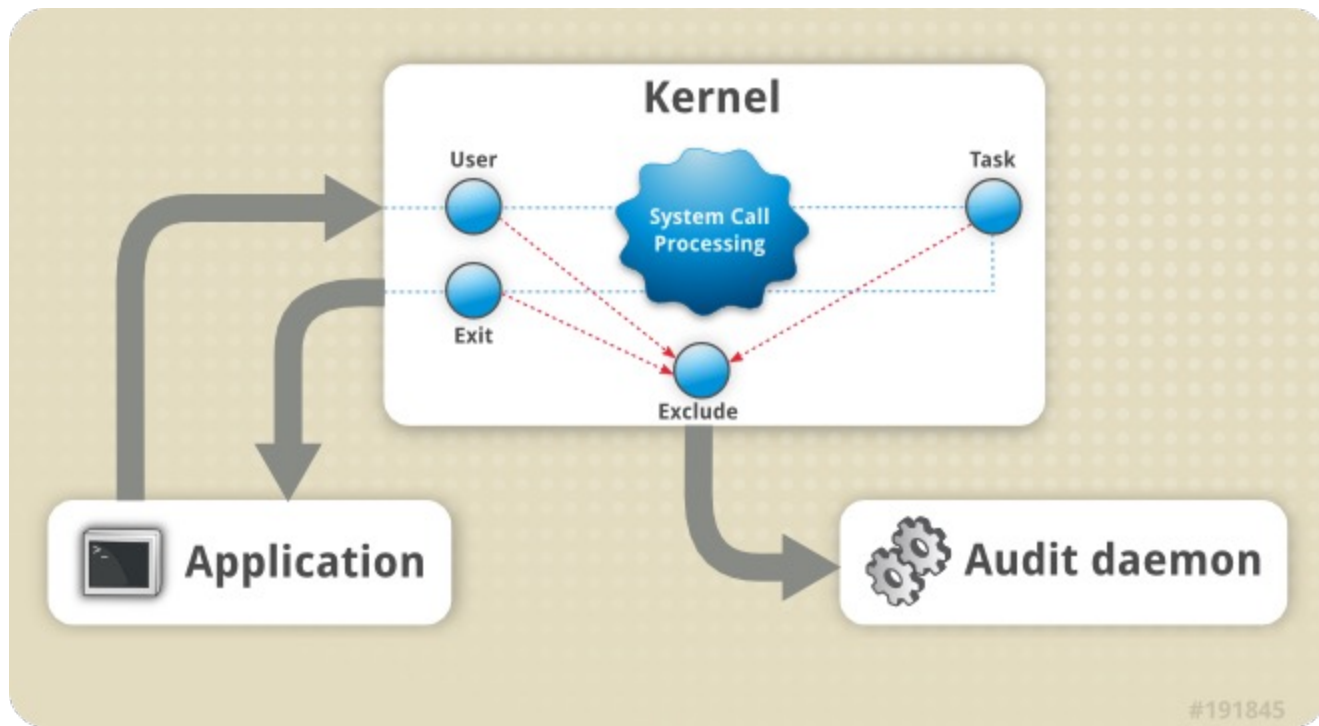
- Reticence to deploy EDR to more interesting systems
- But also...
 - EDR platforms are moving to eBPF but...
 - <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=eBPF>
 - 14 vulnerabilities
 - Broadly, eBPF doesn't really work for end-users
 - Auditd is mainline
 - May already be there
 - Could do a lot more than many will realise

Lessons learnt

- <https://github.com/timb-machine/linux-malware>
 - Yes, there is interesting malware
 - No, we don't do enough to protect ourselves
 - I actually spotted my first bit of malware derived from our research 😞
- Vertical specific systems often don't have sufficient monitoring
 - Banking systems of record
 - Telecomms OSS/BSS
 - Retail payment platforms
 - Operational technologies
 - Etc

Tracing all the things with Auditd

- Syscalls
 - *entry*, task, **exit**, **user**, exclude, **filesystem**
 - -a exit,always -F arch=b64 -S all -k tb_exit_b64_all_syscall
 - -a exit,always -F arch=b32 -S all -k tb_exit_b32_all_syscall
 - -a user,always -F arch=b64 -S all -k tb_user_b64_all_syscall
 - -a user,always -F arch=b32 -S all -k tb_user_b32_all_syscall
- File systems
 - **read**, **write**, **execute**, **attribute** operations
 - -w / -p r -k tb_read_all_files
 - -w / -p w -k tb_write_all_files
 - -w / -p x -k tb_execute_all_files
 - -w / -p a -k tb_attribute_all_files



A dirty script

```
find /opt/component -name -perm -o+w | while read filename
```

```
do
```

```
    printf -- "-w %s -p r -k flag-%s-r\n" "${filename}" "$(printf "%s" "${filename}" | tr "\"/\n" _)">>/etc/audit/rules.d/honeypot-component-dynamic.rules
```

```
    printf -- "-w %s -p w -k flag-%s-w\n" "${filename}" "$(printf "%s" "${filename}" | tr "\"/\n" _)">>/etc/audit/rules.d/honeypot-component-dynamic.rules
```

```
    printf -- "-w %s -p w -k flag-%s-x\n" "${filename}" "$(printf "%s" "${filename}" | tr "\"/\n" _)">>/etc/audit/rules.d/honeypot-component-dynamic.rules
```

```
    printf -- "-w %s -p a -k flag-%s-a\n" "${filename}" "$(printf "%s" "${filename}" | tr "\"/\n" _)">>/etc/audit/rules.d/honeypot-component-dynamic.rules
```

```
done
```

Useful links

- Auditd documentation
 - https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardenin_g/auditing-the-system_security-hardening
- A decent blog post on Auditd for detection
 - <https://izyknows.medium.com/linux-auditd-for-threat-detection-d06c8b941505>
- Upstream rules
 - <https://github.com/linux-audit/audit-userspace/tree/master/rules>
- ATT&CK aligned rules
 - <https://github.com/bfuzzy/auditd-attack>
- UK HMG rules
 - <https://github.com/alphagov/chef-auditd>
- A decent blend
 - <https://github.com/Neo23x0/auditd>