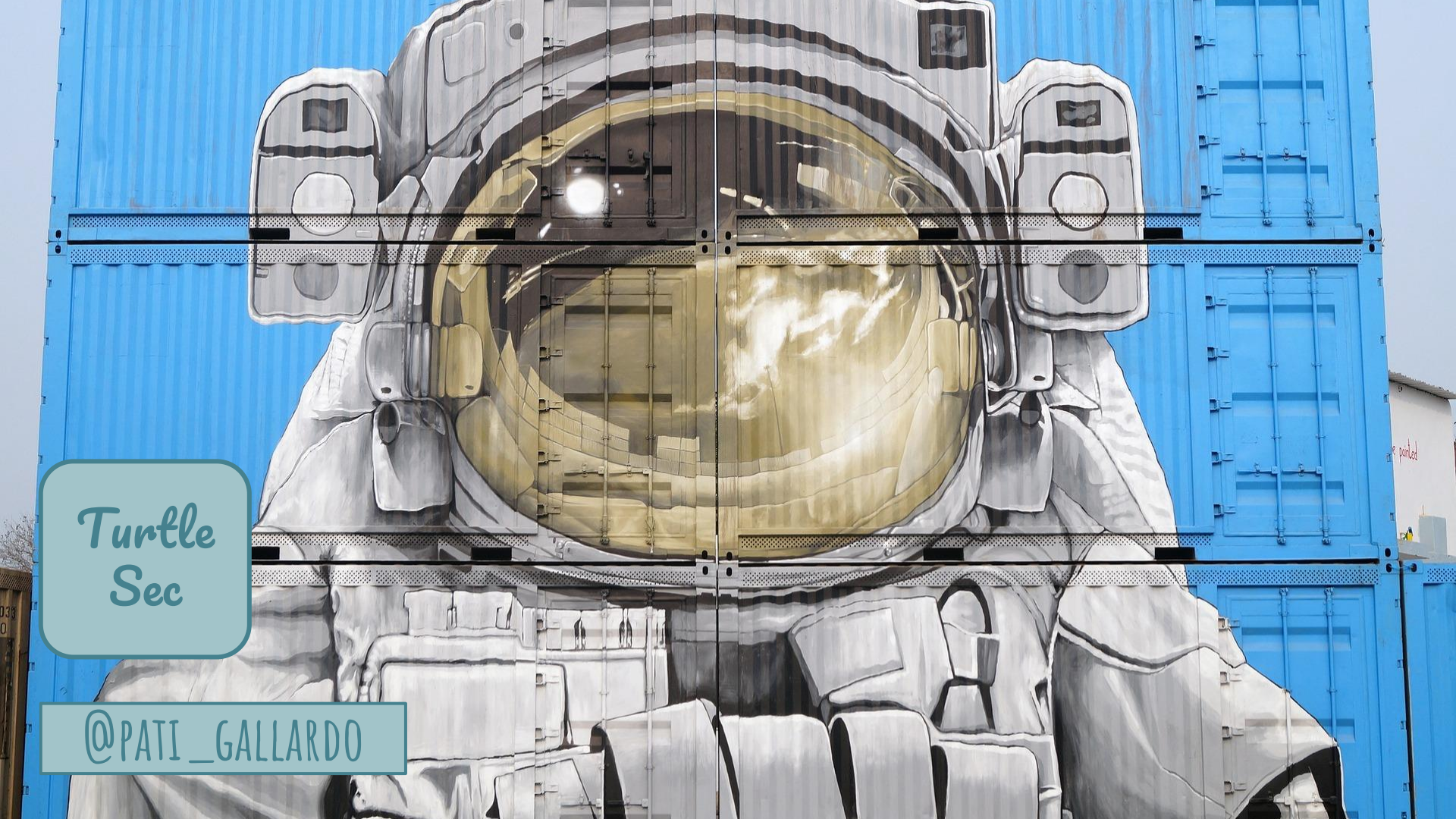

Turtle
Sec
@PATI_GALLARDO

@pati_gallardo

Linux Security

and the Chromium Sandbox

Patricia Aas, TurtleSec

BlackHoodie 2018

Patricia Aas - Consultant

@pati_gallardo

C++ Programmer, Application Security
Currently : **TurtleSec**

Previously : Vivaldi, Cisco Systems, Knowit, Opera Software
Master in Computer Science - main language Java

Pronouns: she/her

Remote Code Execution
is what browsers do.

Sandboxing

- **System External Threat** : Protect the system from the vulnerabilities in the browser
- **System Internal Threat** : Protect the browser from malware on the system

Outline

- Process Architecture
- Linux Security APIs
- The Initial Sandbox
- Shrinking the Sandbox

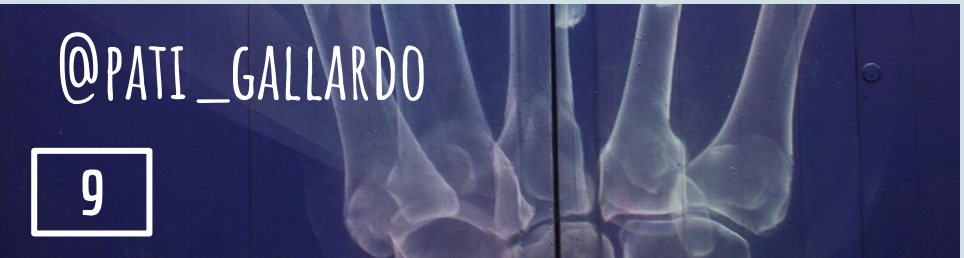
Process Architecture

The Executable Files

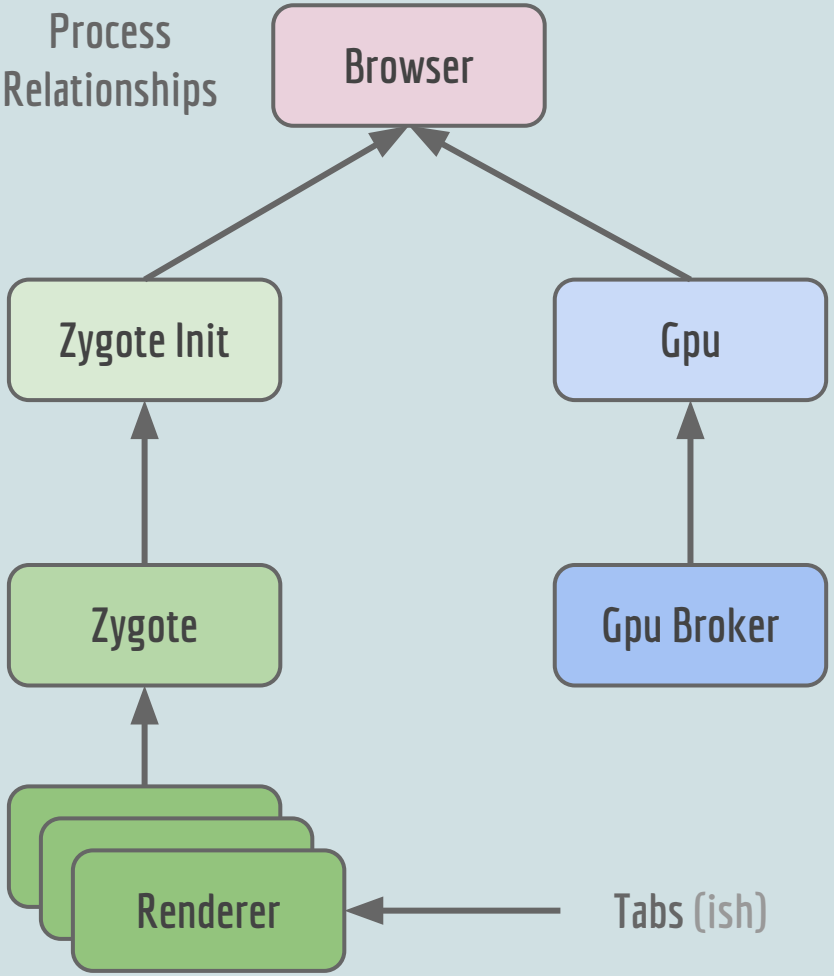
- **vivaldi :** The bash script wrapper, launches vivaldi-bin. Sets up environment.
- **vivaldi-bin :** The browser binary
- **vivaldi-sandbox :** A setuid binary, not in much use today

Process Architecture

@pati_gallardo



Process Relationships



The Running Processes

1. **Browser :**
   vivaldi-bin
2. **Zygote (Parent) :**
   vivaldi-bin --type=**zygote**
3. **Zygote (Child) :**
   vivaldi-bin --type=**zygote**
4. **Renderer (MANY) :**
   vivaldi-bin --type=**renderer**
5. **GPU (Parent) :**
   vivaldi-bin --type=**gpu-process**
6. **GPU Broker (Child) :**
   vivaldi-bin --type=**gpu-broker**

Outline

- Process Architecture
- Linux Security APIs
- The Initial Sandbox
- Shrinking the Sandbox

Linux Security APIs

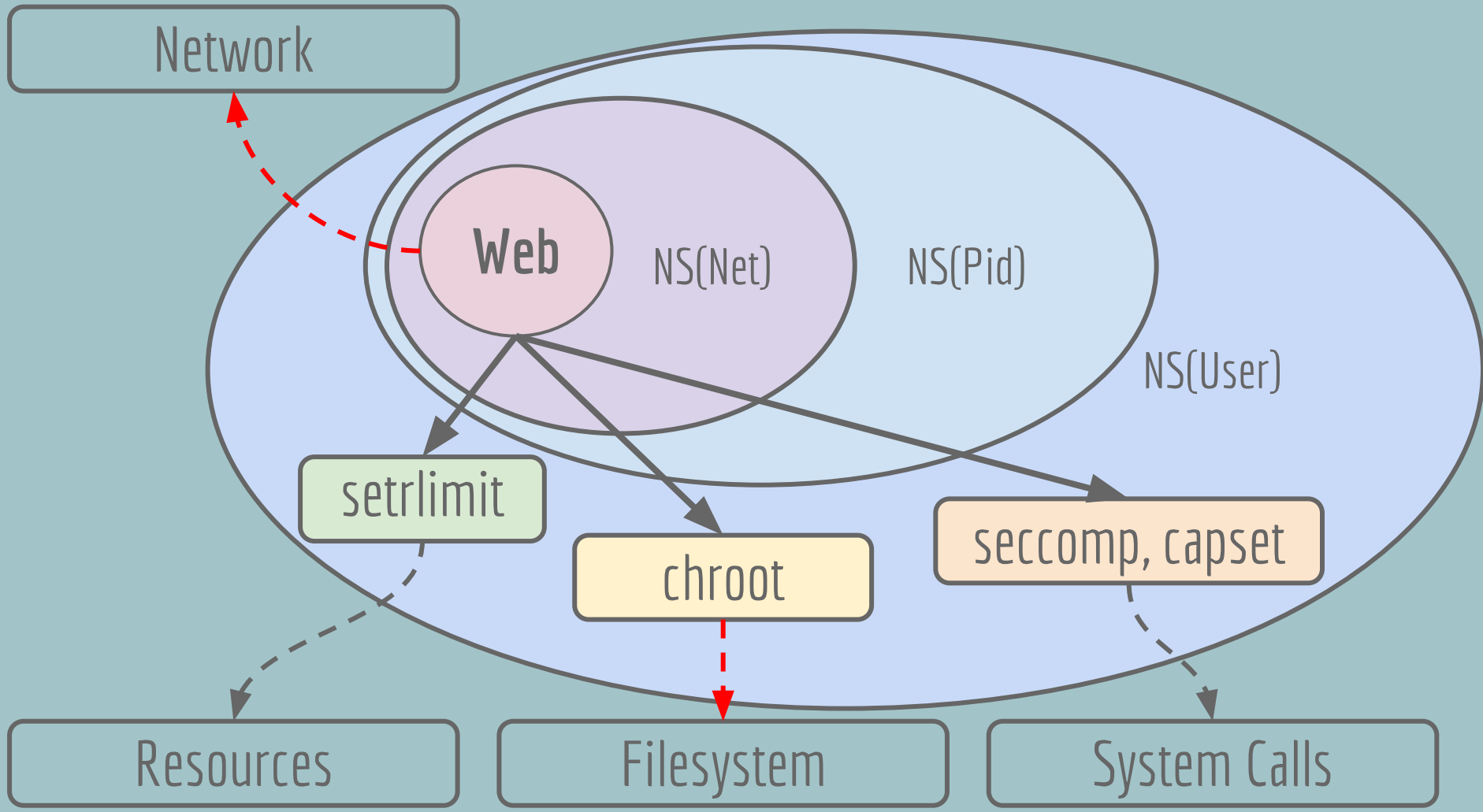
CHROME://SANDBOX

Sandbox Status

SUID Sandbox	No
Namespace Sandbox	Yes
PID namespaces	Yes
Network namespaces	Yes
Seccomp-BPF sandbox	Yes
Seccomp-BPF sandbox supports TSYNC	Yes
Yama LSM Enforcing	Yes

You are adequately sandboxed.

@PATI_GALLARDO


Network
Web
NS(Net)
NS(Pid)
NS(User)
setrlimit
chroot
seccomp, capset
Resources
Filesystem
System Calls

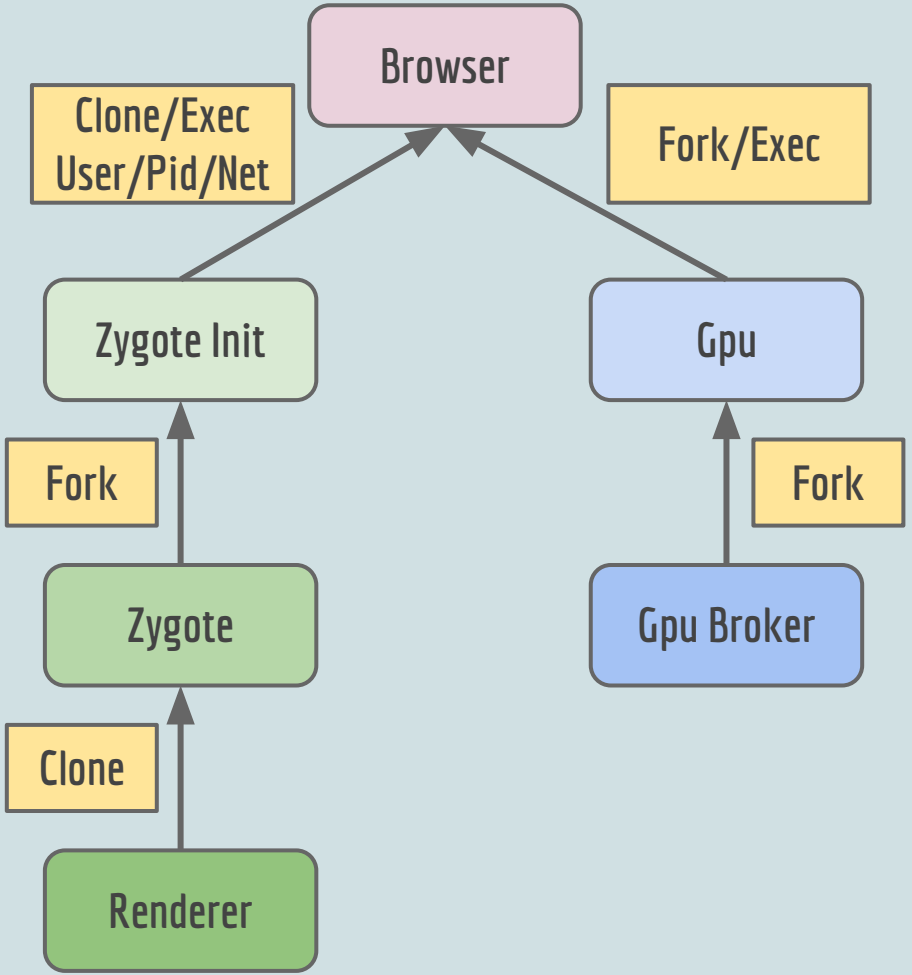
Outline

- Process Architecture
- Linux Security APIs
- The Initial Sandbox
- Shrinking the Sandbox

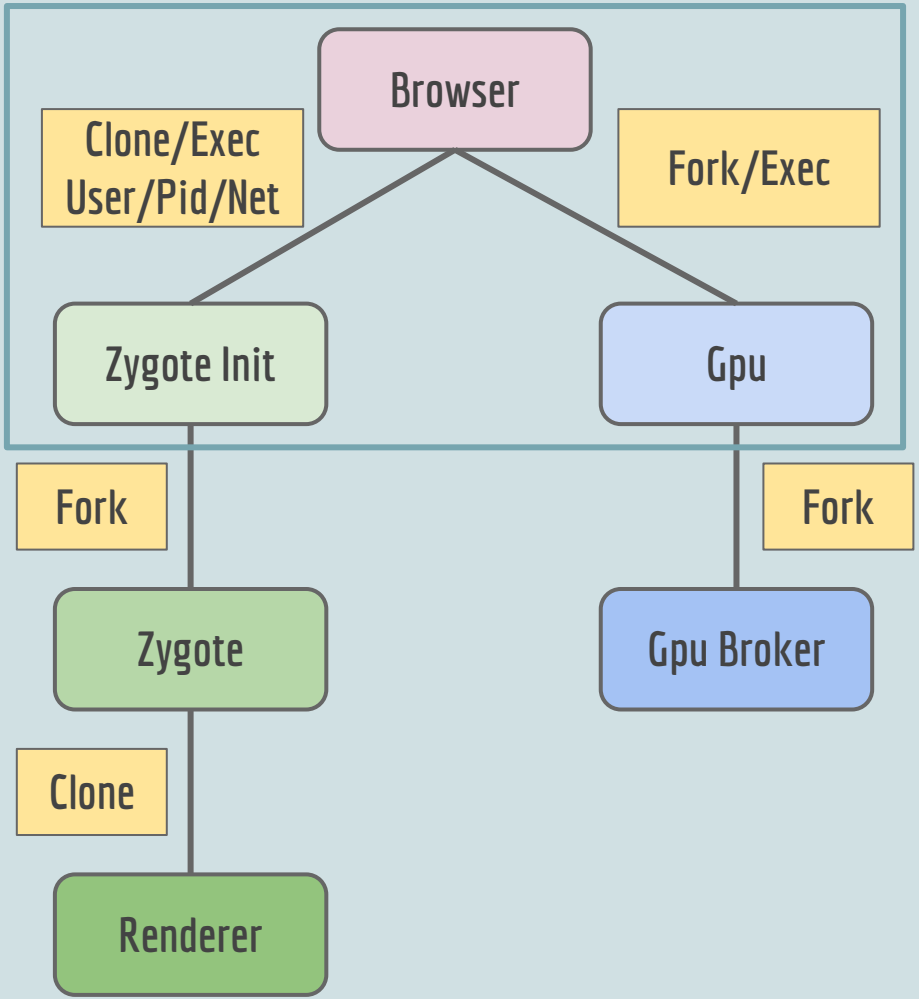
The Initial Sandbox

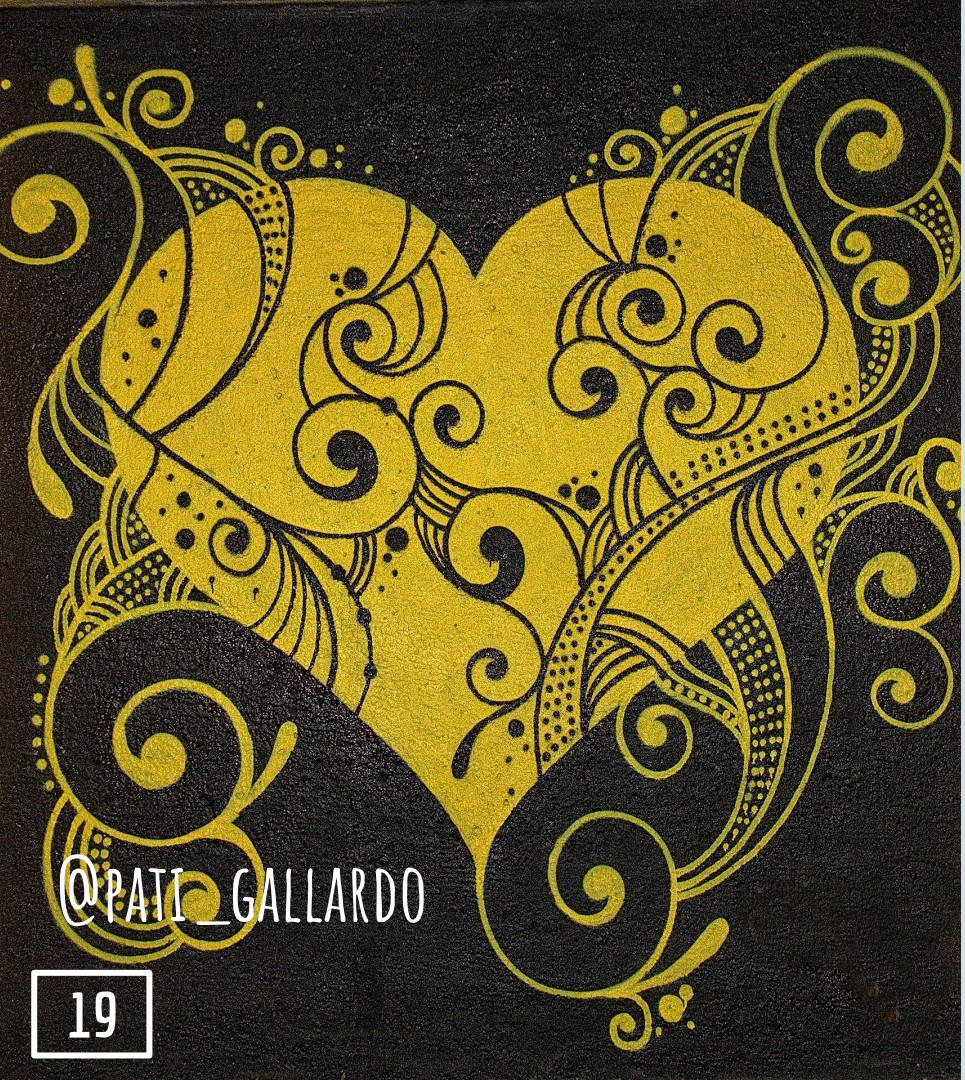
One Binary

@pati_gallardo



Initial Sandbox





Fork / Exec

- A typical start of a new process on Linux is a "**fork/exec**"
- "**Forking**" is creating a new process
- "**Exec**" is executing a binary in a process
- "**Clone**" is a type of fork which can restrict a process at creation

Sandboxing Opportunities: Fork

1. Before Fork
2. After Fork

Sandboxing Opportunities: "Fork"/exec

1. Before clone/fork
2. (At clone)
3. Before Exec
4. At startup (input : argv, env)

Namespaces

@PATI_GALLARDO

Namespaces
(Zygotes/Renderers)

Limits what a process can see

Api : clone/unshare

Namespaces in use

CLONE_NEWUSER

Inside a USER NS we can create a PID NS.

CLONE_NEWPID

Same PID number can represent different processes in different PID namespaces. One init (PID 1) process per PID NS

CLONE_NEWNET

Isolate a process from network

At Clone : Create NAmespaces

Clone flags define the process* created and will create namespaces (NS) for it

1. Test which NS are available
2. Fail if not sufficient
3. Construct the biggest supported and applicable set

Emulates fork with longjmp

* Also used to create threads

```
int flags = CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET;
jmp_buf env;
if (setjmp(env) == 0) {
  return CloneAndLongjmpInChild(flags, ptid, ctid, &env);
}
```

chromium/base/process/launch_posix.cc

Code Simplified To Illustrate

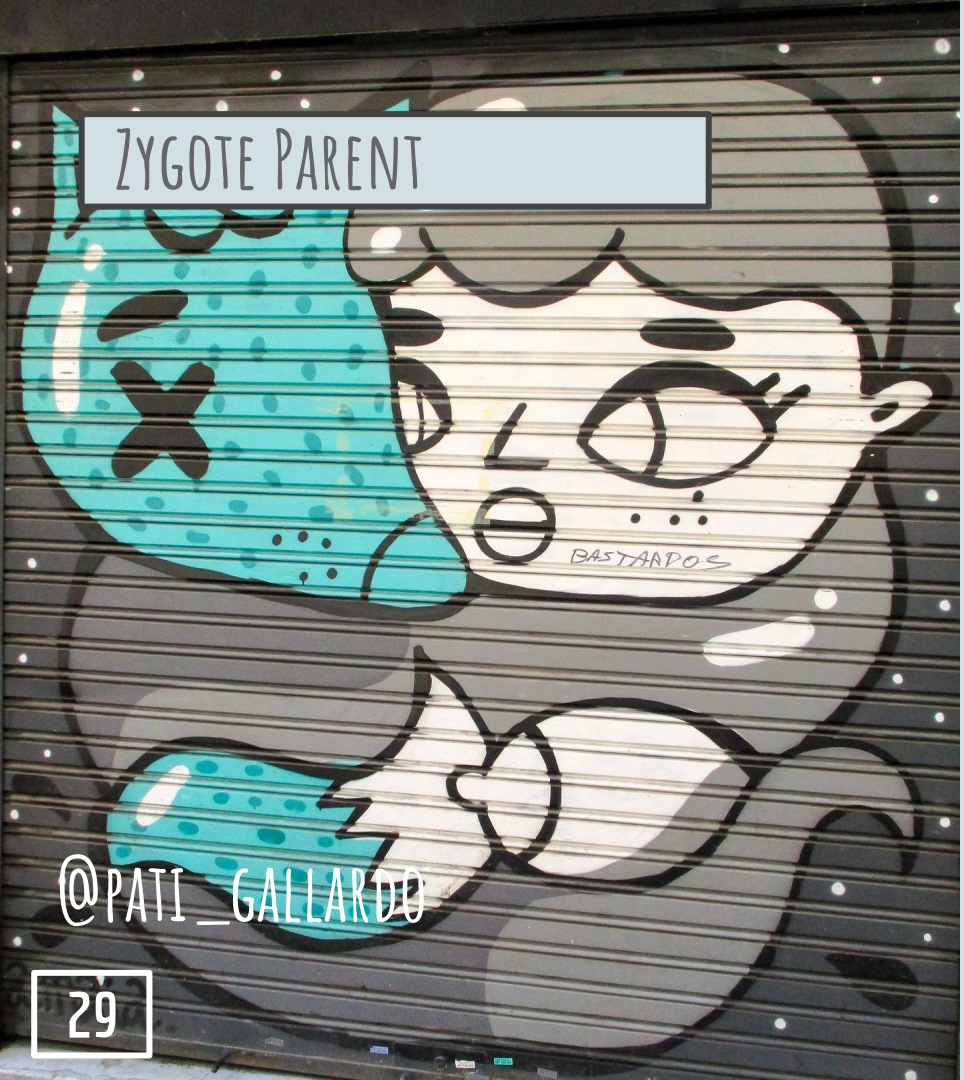
```
pid_t CloneAndLongjmpInChild(unsigned long flags,
                             pid_t* ptid,
                             pid_t* ctid,
                             jmp_buf* env) {
  char stack_buf[PTHREAD_STACK_MIN];
  void* stack = stack_buf + sizeof(stack_buf);
  return clone(&CloneHelper,
               stack, flags, env, ptid, nullptr, ctid);
}
```

chromium/base/process/launch_posix.cc

Code Simplified To Illustrate

```
int CloneHelper(void* arg) {
  jmp_buf* env_ptr = reinterpret_cast<jmp_buf*>(arg);
  longjmp(*env_ptr, 1);
  // Should not be reached
  assert(false);
  return 1;
}
```

chromium/base/process/launch_posix.cc



Unshare- User Namespace

`unshare(CLONE_NEWUSER)`

Alternative to clone(CLONE_NEWUSER), will **not create new process**, but move caller.

Credentials::CanCreateProcessInNewUserNS

Sandboxing Opportunities: "Fork"/exec

1. Before clone/fork
2. At clone
3. Before Exec
4. At startup (input : argv, env)

Before Exec : Launch Options

Prepare for Exec

1. Fix the environment
2. Fix file descriptors
3. Fix signal handling
4. Set up process group
5. Maximize resource limits
6. Set PR_SET_NO_NEW_PRIVS
7. Change current dir
8. Select executable path
9. Setup command-line

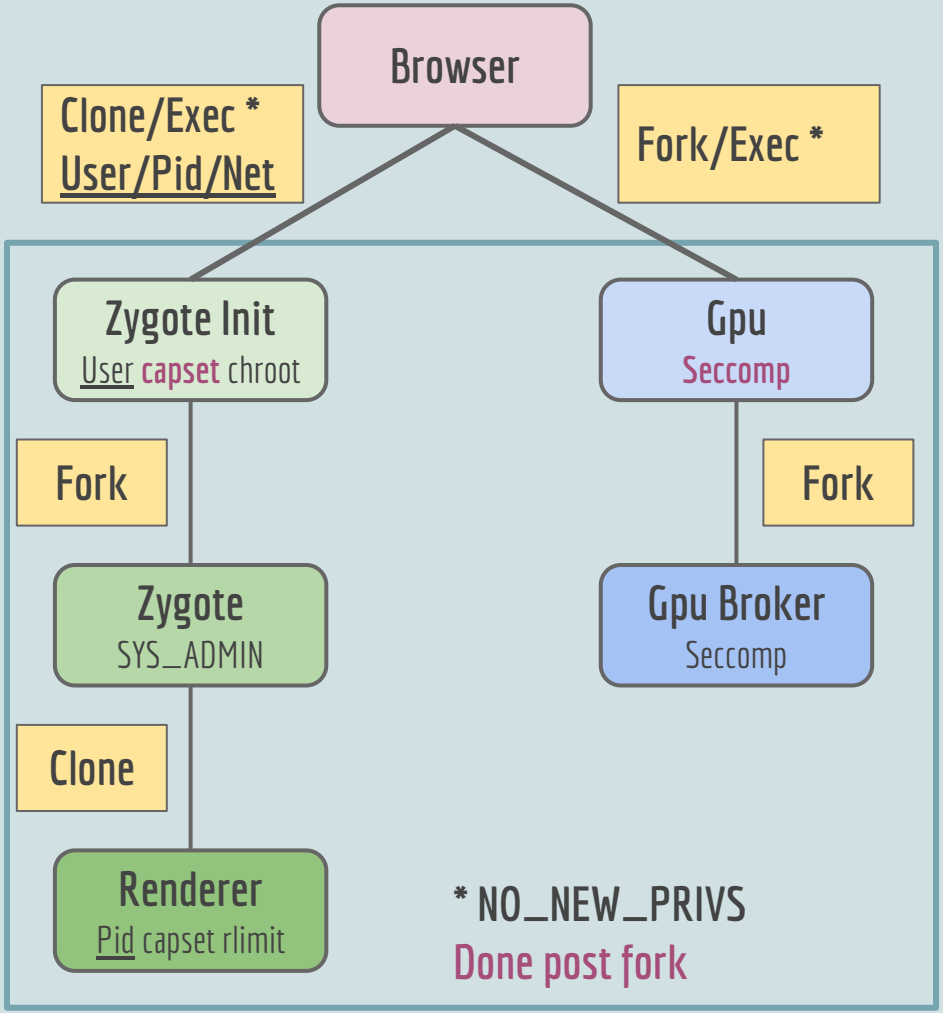
Outline

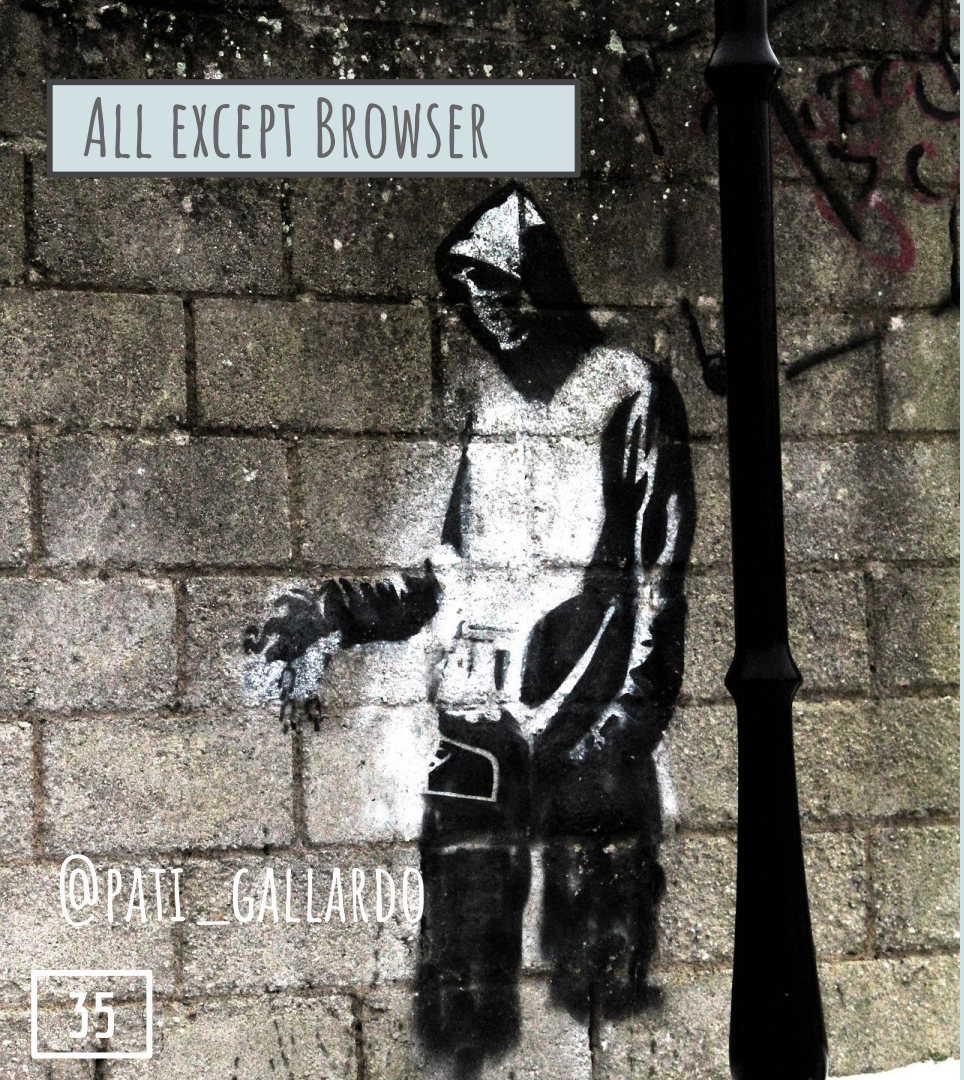
- Process Architecture
- Linux Security APIs
- The Initial Sandbox
- **Shrinking the Sandbox**

Shrinking the Sandbox

Shrinking the Sandbox

@pati_gallardo





PR_SET_NO_NEW_PRIVS

prctl(PR_SET_NO_NEW_PRIVS)

Preserved across fork, clone and execve

"If no_new_privs is set, then **operations that grant new privileges (i.e. execve) will either fail or not grant them**. This affects suid/sgid, file capabilities, and LSMs."

/usr/include/linux/prctl.h

Seccomp

Seccomp

(Renderers/GPU/Broker)

Limits which syscalls a process can call and how these calls are handled

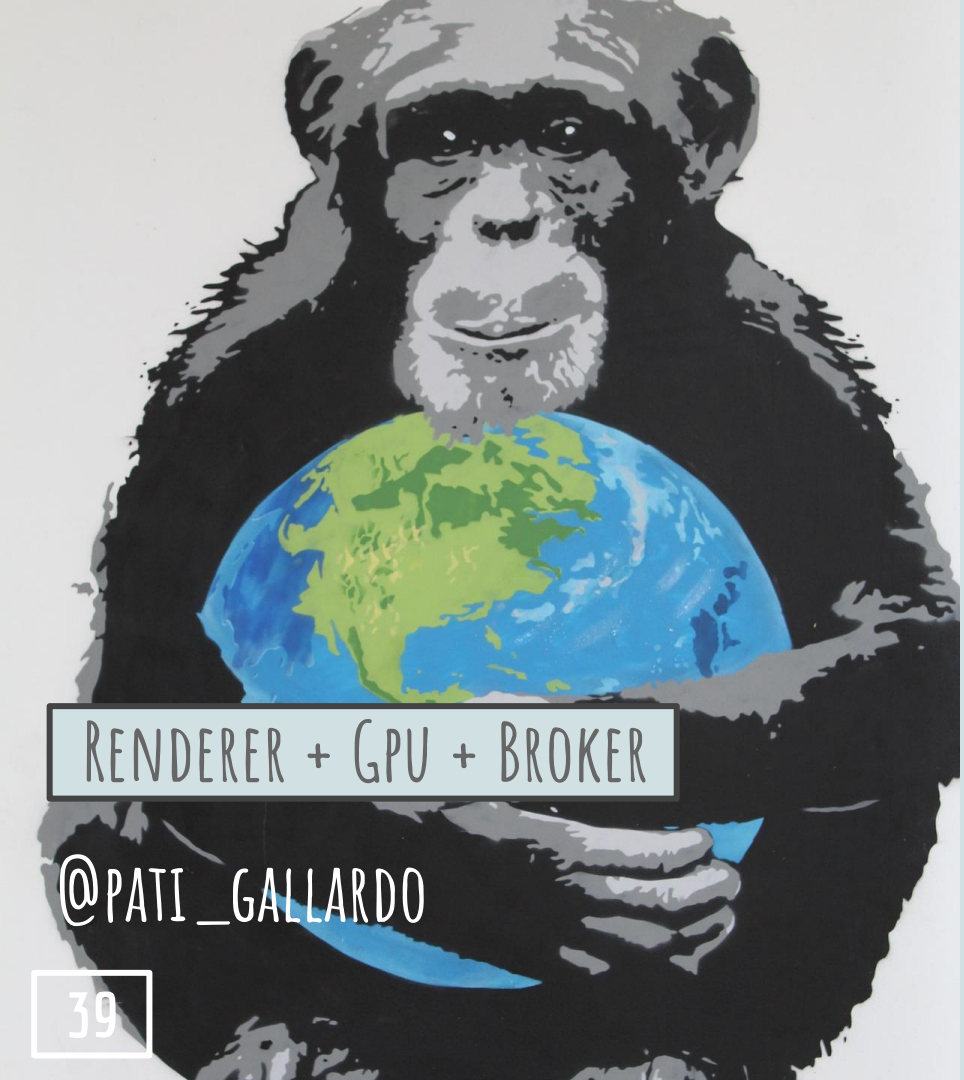
Api : seccomp

Seccomp BPF Program

Program written in an assembly-like language to filter system-calls.

Runs in a simple VM in kernel space. All syscalls will be filtered by this program

TSYNC : Once a Seccomp Program is installed it applies to all threads in a process



Seccomp : BPF Policies

BPF Program defined in a Policy

Fundamentally a **whitelist**, allows a set of syscalls and has custom handling of others.

An extended Policy is generally more permissive

1. **BaselinePolicy**
1.1 **GpuProcessPolicy**
1.1.1 **GpuBrokerProcessPolicy**
1.2 **RendererProcessPolicy**

Code Simplified To Illustrate

```
void StartSandboxWithPolicy(sandbox::bpf_dsl::Policy* policy)
{
  SandboxBPF sandbox(policy);
  assert(sandbox.StartSandbox());
}

bool SandboxBPF::StartSandbox() {
  InstallFilter();
  return true;
}
```

chromium/services/service_manager/sandbox/linux/sandbox_seccomp_bpf_linux.cc
chromium/sandbox/linux/seccomp-bpf/sandbox_bpf.cc

Code Simplified To Illustrate

```
void SandboxBPF::InstallFilter() {
  CodeGen::Program program = AssembleFilter();
  struct sock_filter bpf[program.size()];
  const struct sock_fprog prog =
   { static_cast<unsigned short>(program.size()), bpf };
  memcpy(bpf, &program[0], sizeof(bpf));
  assert(prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0) == 0);
  assert(seccomp(SECCOMP_SET_MODE_FILTER,
                 SECCOMP_FILTER_FLAG_TSYNC, &prog) == 0);
}
```

chromium/sandbox/linux/seccomp-bpf/sandbox_bpf.cc

Chroot

Chroot

(Zygotes/Renderers)

Limits what a process can see of the filesystem

Api : chroot

Zygotes + Renderer

@pati_gallardo

Chroot : Drop Access to FS

- **clone(CLONE_FS)** a child process
- Child **chroot("/proc/self/fdinfo/")**
- Child immediately does a **chdir("/")**
- Child does **_exit(kExitSuccess)**

You can see this by looking at

ls -l /proc/<pid>/root

Of the Zygote or any ancestor

Credentials::DropFileSystemAccess

```
bool ChrootToSafeEmptyDir() {
  pid_t pid = -1;
  char stack_buf[PTHREAD_STACK_MIN];
  void* stack = stack_buf + sizeof(stack_buf);
  int clone_flags = CLONE_FS | LINUX_SIGCHLD;
  pid = clone(ChrootToSelfFdinfo, stack, clone_flags, nullptr);
  int status = -1;
  assert(HANDLE_EINTR(waitpid(pid, &status, 0)) == pid);
  return WIFEXITED(status) && WEXITSTATUS(status) == kExitSuccess;
}
```

chromium/sandbox/linux/services/credentials.cc

Code Simplified To Illustrate

```
int ChrootToSelfFdinfo(void*) {
 assert(chroot("/proc/self/fdinfo/") == 0);
 assert(chdir("/") == 0);
 _exit(kExitSuccess);
}
```

chromium/sandbox/linux/services/credentials.cc


Capabilities
@PATI_GALLARDO
@PATI_GALLARDO

Capabilities
(Zygote/Renderers)

Limits what a process is privileged enough to do

Api : capset

Drop Capabilities

Uses capset() to drop all or some capabilities

"Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled."
Man page for capabilities

Credentials::DropAllCapabilities

Resource Limits

Resource Limits
(Renderers)

Limits the access this process has to platform resources

Api : setrlimit

Resource Limits : setrlimit

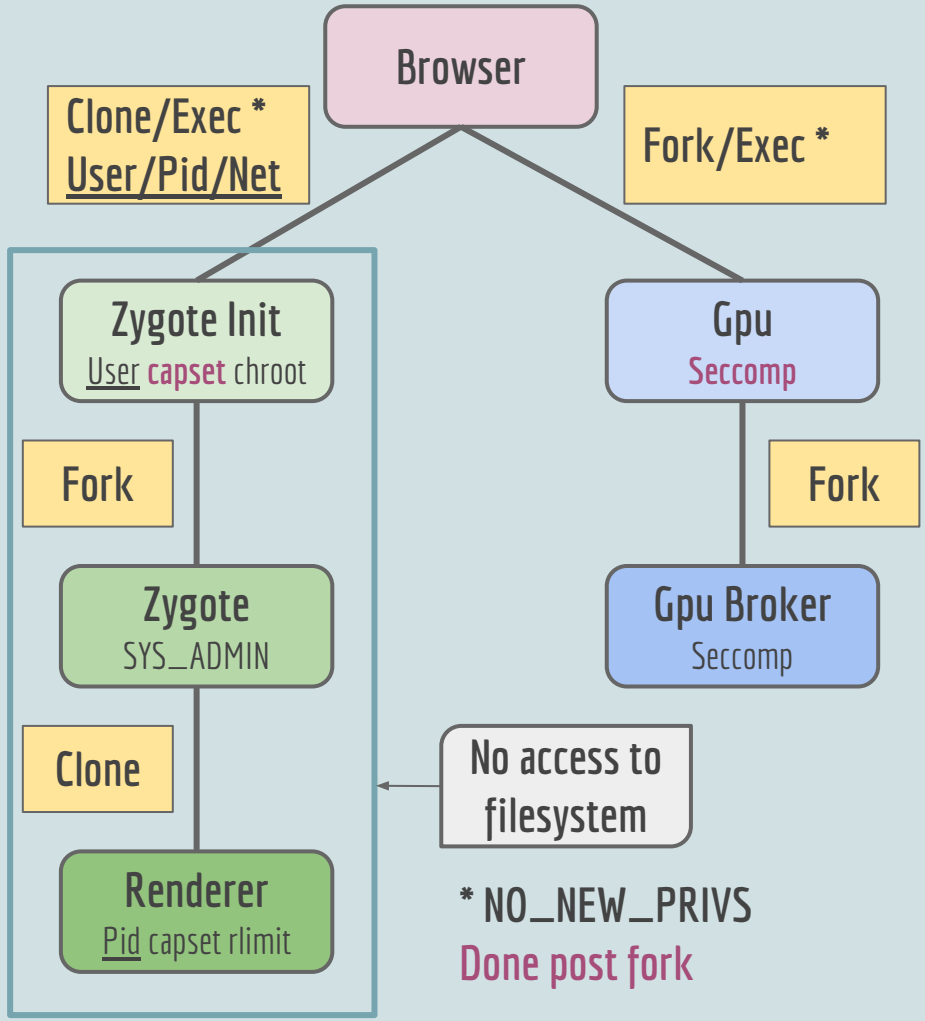
Limits using setrlimit:

1. **RLIMIT_AS** : Maximum size of the process' virtual memory (address space) in bytes
2. **RLIMIT_DATA** : Maximum size of the process's data segment

LinuxSandbox::LimitAddressSpace

Trust is Relative

@pati_gallardo



CHROME://SANDBOX

Sandbox Status

SUID Sandbox	No
Namespace Sandbox	Yes
PID namespaces	Yes
Network namespaces	Yes
Seccomp-BPF sandbox	Yes
Seccomp-BPF sandbox supports TSYNC	Yes
Yama LSM Enforcing	Yes

You are adequately sandboxed.

@PATI_GALLARDO

Sources

Chromium/Kernel source + Linux Man Pages + lwn.net

Michael Kerrisk

Book: *The Linux Programming Interface*

Course: ***Linux Security and Isolation APIs***

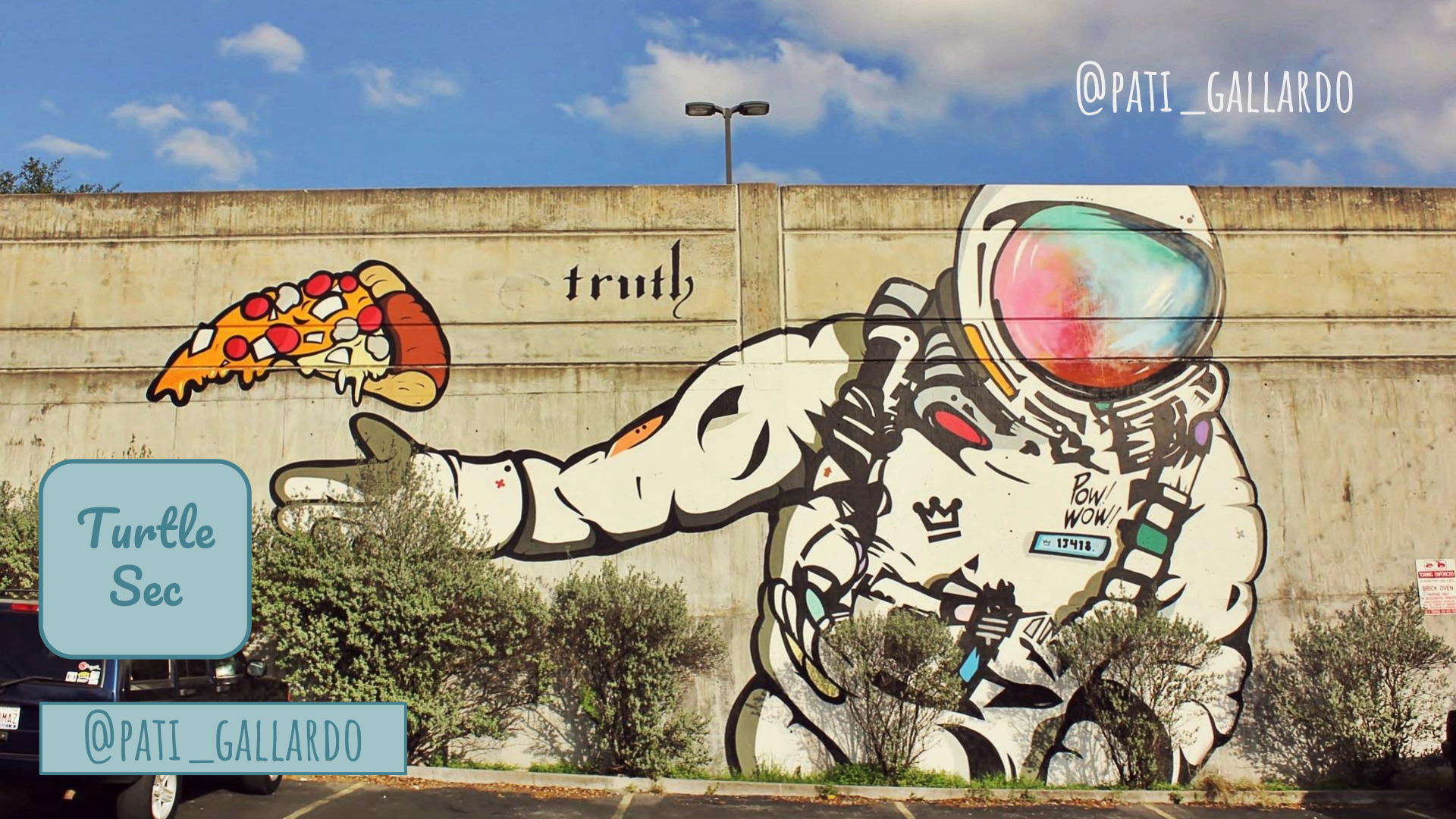
All Errors Are My Own

@pati_gallardo

Patricia Aas, Consultant

TurtleSec

C++ and Application Security


@PATI_GALLARDO
truth
POW! WOW!
17418
Turtle Sec
@PATI_GALLARDO

Appendix / Some Notes

Other APIs in use

Not Used Much in Chromium, but...

YAMA LSM

Limits the access that other process have to this process - especially ptracing

Status is checked by reading :

/proc/sys/kernel/yama/ptrace_scope

Setuid/setgid

(Legacy)

Increases what a process is privileged enough to do, by using the privilege of the executables owner

Api : set*uid / set*gid

CGroups
(ChromeOS)

Limits the resources available to a process. Used in ChromeOS - not covered here.

/proc/‹PID›/cgroup

/proc/cgroups

Process Groups
(Chromedriver)

Can be used to treat a group of processes as one.
Used with the ‘detach’ property of Chromedriver

Api : setpgid

Challenges

Debugging

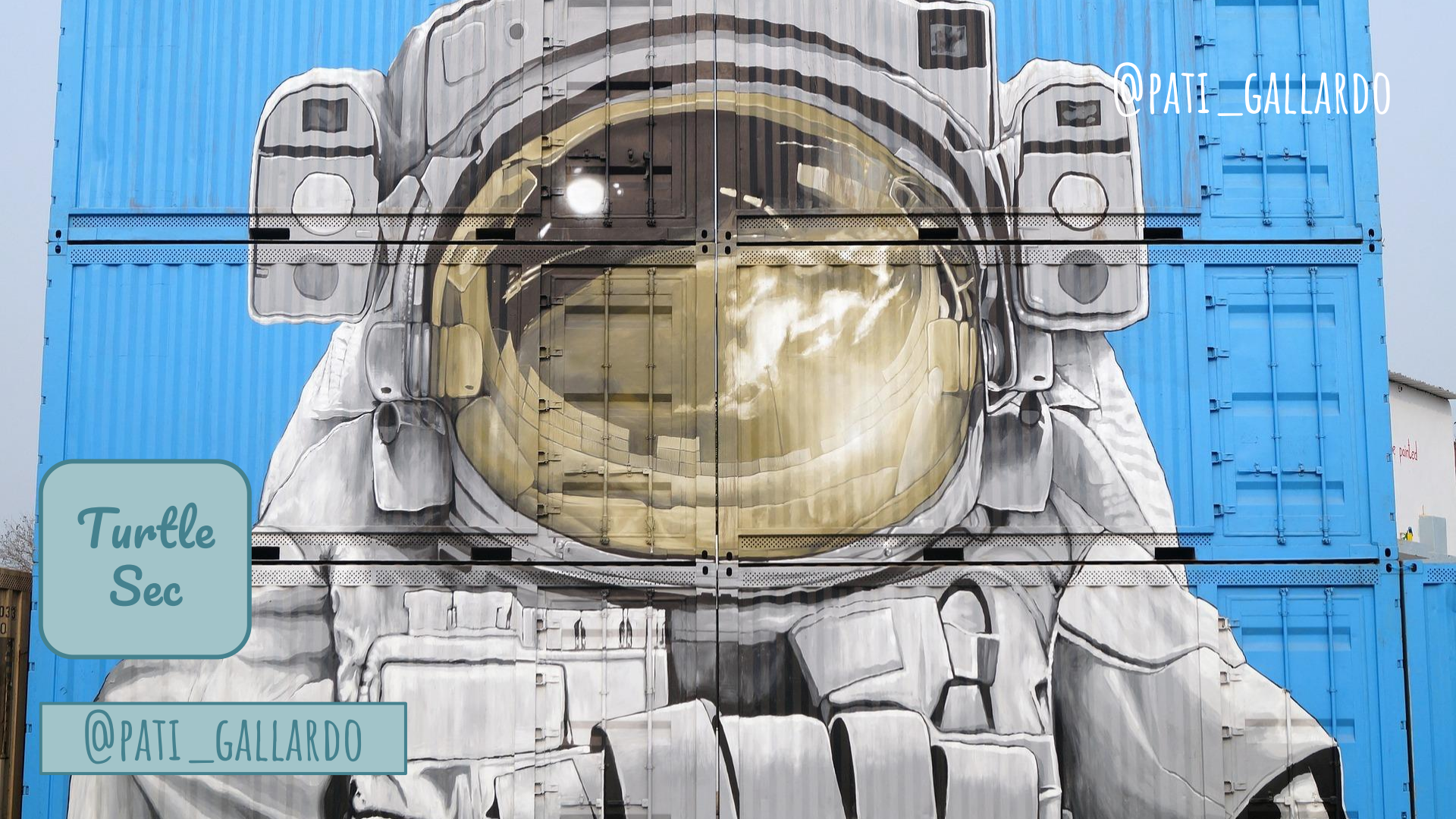
@pati_gallardo

Crash Reporting

@pati_gallardo

Photos from pixabay.com

Patricia Aas, TurtleSec
@pati_gallardo


@PATI_GALLARDO
Turtle
Sec
@PATI_GALLARDO