SysXCHG

Refining Privilege with Adaptive System Call Filters

Alexander J. Gaidis   Vaggelis Atlidakis   Vasileios P. Kemerlis

Secure Systems Laboratory (SSL)
Department of Computer Science
Brown University

Motivation

- **Syscall filtering** attempts to **limit over-privilege** w.r.t. the syscall API

- **Syscall filtering** attempts to **limit over-privilege** w.r.t. the syscall API
- `Seccomp-BPF` is the de facto filtering mechanism in Linux

- **Syscall filtering** attempts to **limit over-privilege** w.r.t. the syscall API
- `Seccomp-BPF` is the de facto filtering mechanism in Linux
- Users install **BPF programs** at a hook-point in the kernel to filter syscalls

- **Syscall filtering** attempts to **limit over-privilege** w.r.t. the syscall API
- `Seccomp-BPF` is the de facto filtering mechanism in Linux
- Users install **BPF programs** at a hook-point in the kernel to filter syscalls
- **All filter programs are run** and the most restrictive action is taken
  - Filters **can block** allowed syscalls, but **cannot allow** blocked syscalls

Seccomp-BPF has a **hierarchical design** that leads to **over-privilege**

BROWN

Seccomp-BPF Problem #1: Effectiveness

Seccomp-BPF has a **hierarchical design** that leads to **over-privilege**

- **Filters are inherited** across process creation and program execution

BROWN

`Seccomp-BPF` Problem #1: Effectiveness

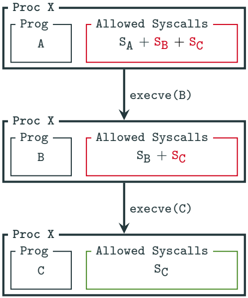
`Seccomp-BPF` has a **hierarchical design** that leads to **over-privilege**

- **Filters are inherited** across process creation and program execution
- A process' privileges can **never grow**

Seccomp-BPF Problem #1: Effectiveness

Seccomp-BPF has a hierarchical design that leads to over-privilege

- **Filters are inherited** across process creation and program execution
- A process' privileges can **never grow**
- A program must allow the syscalls needed by **any sub-programs** ⇝ over-privilege
  - Some binaries must run with twice as many syscalls as necessary



Over-privilege of A: $S_B + S_C$

Over-privilege of B: S_C

Over-privilege of C: $\varnothing$

BROWN

Seccomp-BPF Problem #1: Effectiveness

`Seccomp-BPF` has a **hierarchical design** that leads to **over-privilege**

Exchange filters at runtime to adapt a process' privileges to the current program

Seccomp-BPF Problem #2: Performance

Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

- Most recent work filters syscalls based **solely on number**

Seccomp-BPF Problem #2: Performance

Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

- Most recent work filters syscalls based **solely on number**
- A **bitmap cache** was recently added to Seccomp-BPF to speedup enforcement

BROWN

Seccomp-BPF Problem #2: Performance

Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

- Most recent work filters syscalls based **solely on number**
- A **bitmap cache** was recently added to Seccomp-BPF to speedup enforcement
- Cache is created from a filter by **repeatedly emulating** the filter for individual syscalls

Seccomp-BPF Problem #2: Performance

Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

- Most recent work filters syscalls based **solely on number**
- A **bitmap cache** was recently added to Seccomp-BPF to speedup enforcement
- Cache is created from a filter by **repeatedly emulating** the filter for individual syscalls
  - The cache **slows installation** up to 3.5**x**

BROWN

Seccomp-BPF Problem #2: Performance

Seccomp-BPF improves runtime filtering performance by sacrificing install performance

- Most recent work filters syscalls based **solely on number**
- A **bitmap cache** was recently added to Seccomp-BPF to speedup enforcement
- Cache is created from a filter by **repeatedly emulating** the filter for individual syscalls
  - The cache **slows installation** up to 3.5**x**
- The syscall hot path still requires **multiple function calls** to test the cache

BROWN

Seccomp-BPF Problem #2: Performance

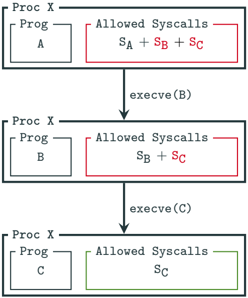
Seccomp-BPF improves runtime filtering performance by **sacrificing install performance**

Pre-compute filters and give each process its own **view of the syscall table** for filtering

`SysXCHG` Design

Inheritance Model:

- "Legacy" model of Seccomp-BPF
- Filters cannot be uninstalled
- Filters are inherited across execve
- Privileges can never grow
- Programs can be over-privileged



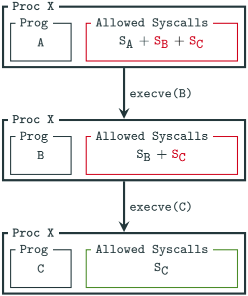
Filtering Models

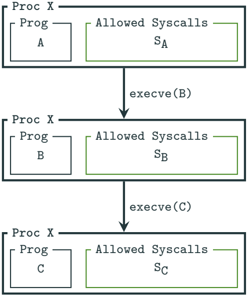
Inheritance Model:

- ✳ "Legacy" model of `Seccomp-BPF`
- ✳ Filters cannot be uninstalled
- ✳ Filters are inherited across `execve`
- ✳ Privileges can never grow
- ✳ Programs can be over-privileged

Exchange Model:

- ✳ Novel model we propose
- ✳ Filters can be replaced
- ✳ Filters aren't inherited across `execve`
- ✳ Privileges adapt to current program
- ✳ Programs are not over-privileged





BROWN

exec Filters

exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

exec Filters

exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

- exec filters are **embedded in binaries** and automatically **installed on execution**

exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

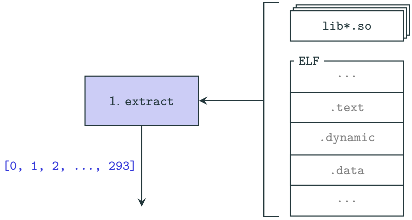
- `exec` filters are **embedded in binaries** and automatically **installed on execution**
- Prepared (offline) by:

BROWN

exec Filters

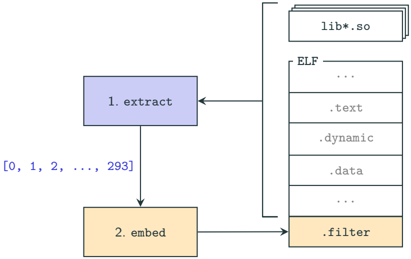
exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

- exec filters are **embedded in binaries** and automatically **installed on execution**
- Prepared (offline) by:
  1. **Extracting** the set of syscalls needed for a binary's benign execution



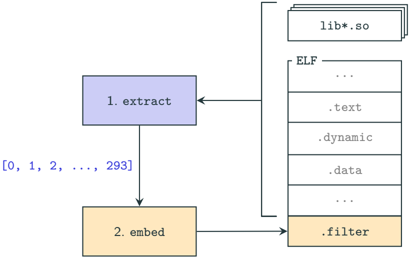
exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

- exec filters are **embedded in binaries** and automatically **installed on execution**
- Prepared (offline) by:
    1. **Extracting** the set of syscalls needed for a binary's benign execution
    2. **Embedding** a filter program in a new ELF section



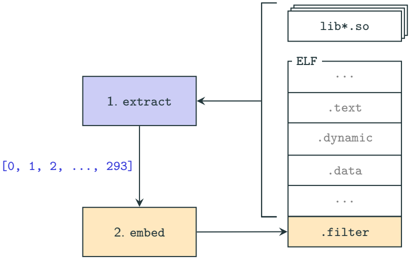
exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

- exec filters are **embedded in binaries** and automatically **installed on execution**
- Prepared (offline) by:
  1. **Extracting** the set of syscalls needed for a binary's benign execution
  2. **Embedding** a filter program in a new ELF section
- We do not depend on specific tools for extraction and embedding



exec filters are a mechanism to **link filters with binaries** to enable secure filter exchanging

- exec filters are **embedded in binaries** and automatically **installed on execution**
- Prepared (offline) by:
  1. **Extracting** the set of syscalls needed for a binary's benign execution
  2. **Embedding** a filter program in a new ELF section
- We do not depend on specific tools for extraction and embedding
- **Agnostic** to the underlying **filter type**



Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

- Replace installed exec filters upon program execution

Filter Exchanging

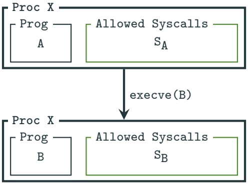
Adapt privileges to the running program by dynamically exchanging exec filters

- Replace installed exec filters upon program execution

Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

- Replace installed exec filters upon program execution

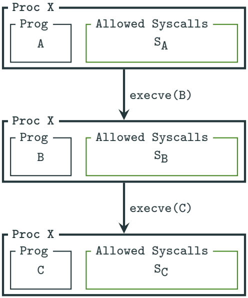


BROWN

Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

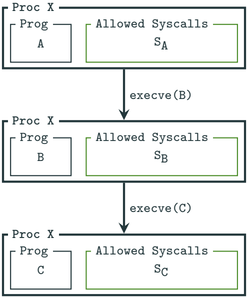
- Replace installed exec filters upon program execution



Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

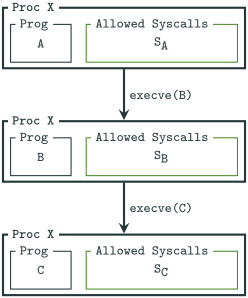
- Replace installed exec filters upon program execution
  - exec filters work on a **program level**



Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

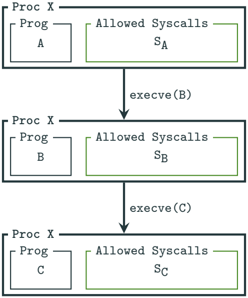
- Replace installed exec filters upon program execution
  - exec filters work on a **program level**
- Manually installed filters adhere to the inheritance model



Filter Exchanging

Adapt privileges to the running program by dynamically exchanging exec filters

- Replace installed exec filters upon program execution
  - exec filters work on a **program level**
- Manually installed filters adhere to the inheritance model
  - Manual filters work on a **process level**



Cryptographic signatures bind exec filters to binaries to prevent malicious privilege increase

Cryptographic signatures bind exec filters to binaries to prevent malicious privilege increase

- Exchanging filters could allow an **adversary** to **increase their privileges** by:

Cryptographic signatures bind exec filters to binaries to prevent malicious privilege increase

- Exchanging filters could allow an **adversary** to **increase their privileges** by:
  i. **Creating** a new program with **no exec filter** and executing it

Security Considerations

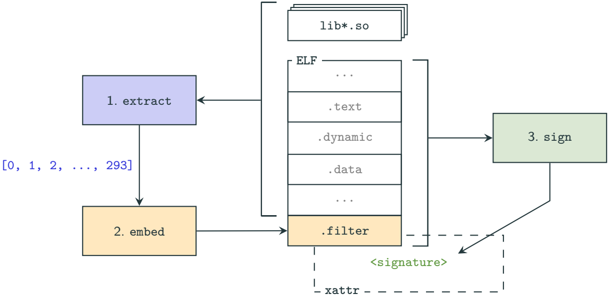
Cryptographic signatures bind `exec` filters to binaries to prevent malicious privilege increase

- Exchanging filters could allow an **adversary** to **increase their privileges** by:
  i. **Creating** a new program with **no** `exec` **filter** and executing it
  ii. **Modifying** a binary's **existing** `exec` **filter** and executing it

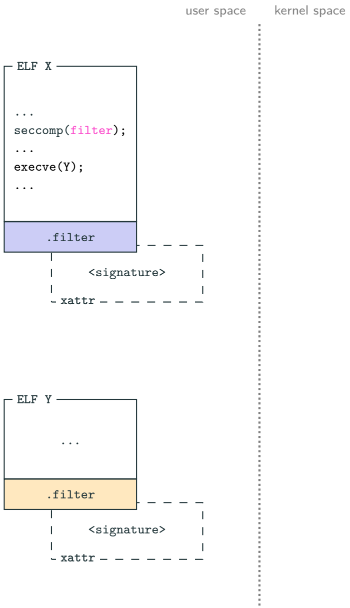
Security Considerations

Cryptographic signatures bind exec filters to binaries to prevent malicious privilege increase

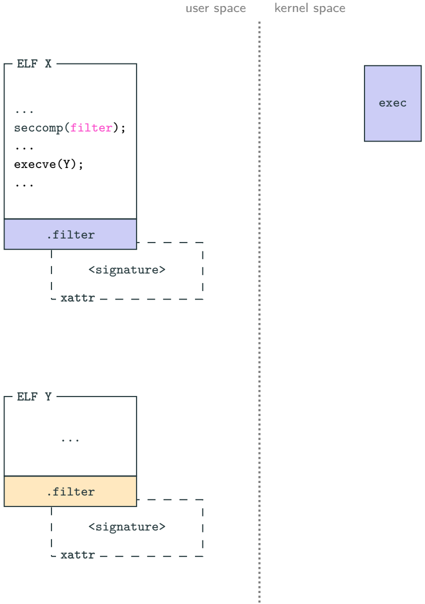
- Exchanging filters could allow an **adversary** to **increase their privileges** by:
  i. **Creating** a new program with **no exec filter** and executing it
  ii. **Modifying** a binary's **existing exec filter** and executing it
- A final (offline) phase:
  3. **Signing** with Linux's IMA security module



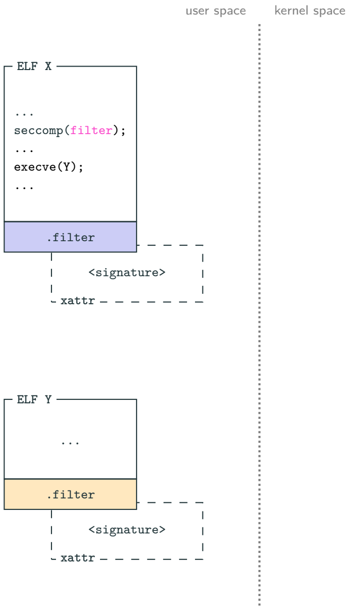
Exchange Example



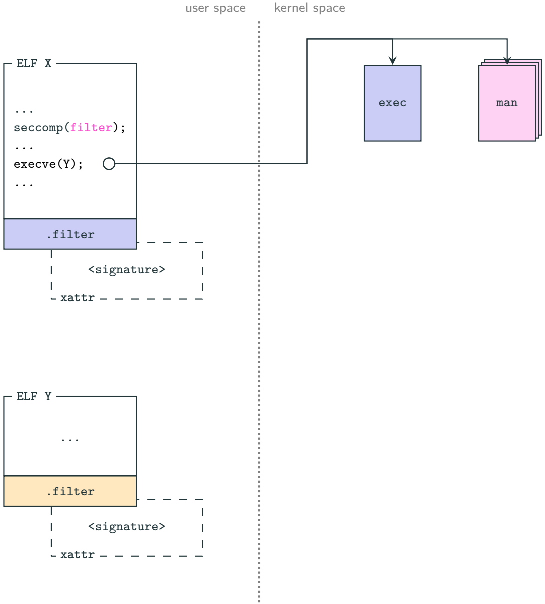
Exchange Example



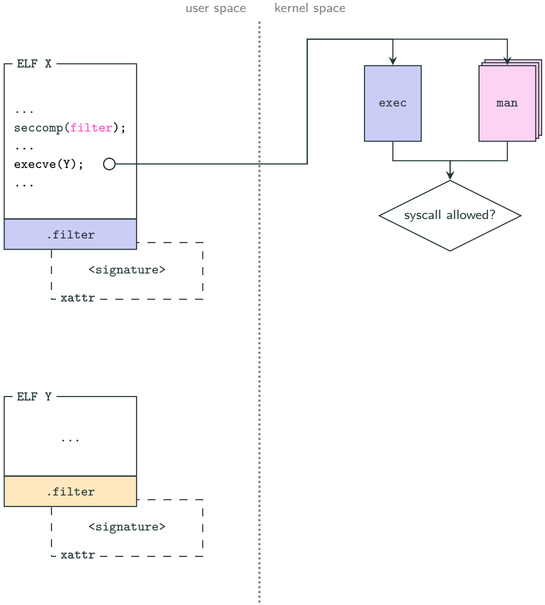
Exchange Example



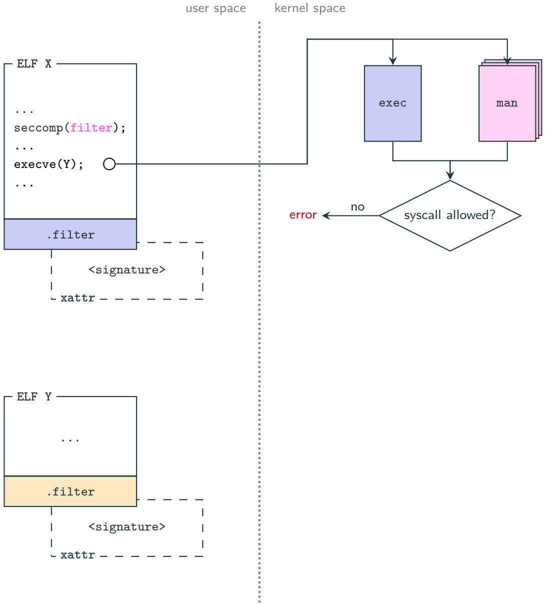
Exchange Example



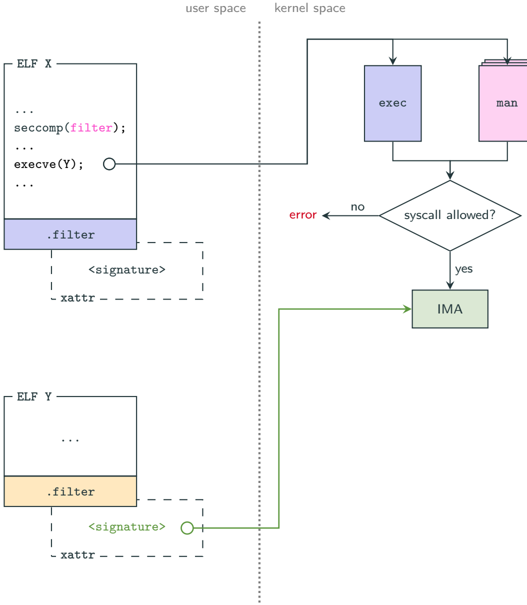
Exchange Example

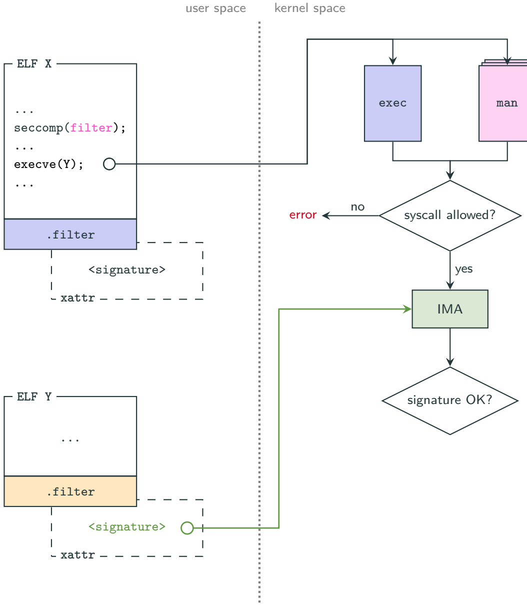


Exchange Example



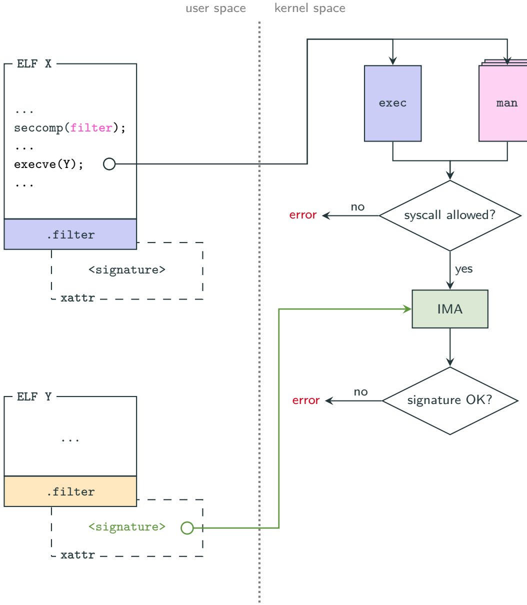
Exchange Example





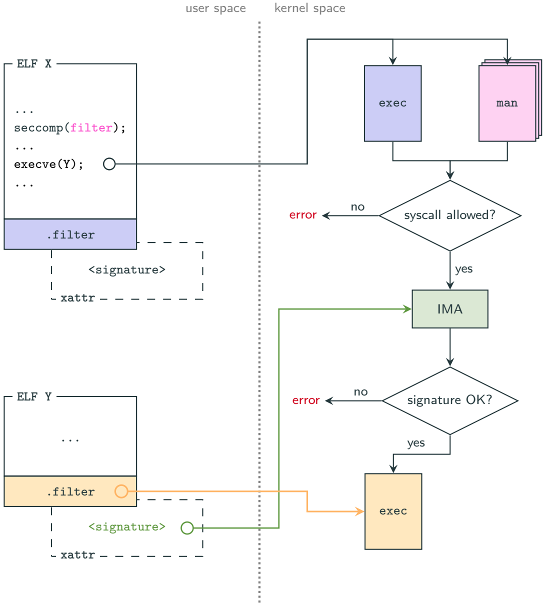
user space
kernel space
ELF X
...
seccomp(filter);
...
execve(Y);
...
.filter
<signature>
xattr
ELF Y
...
.filter
<signature>
xattr
exec
man
error
no
syscall allowed?
yes
IMA
signature OK?

Exchange Example

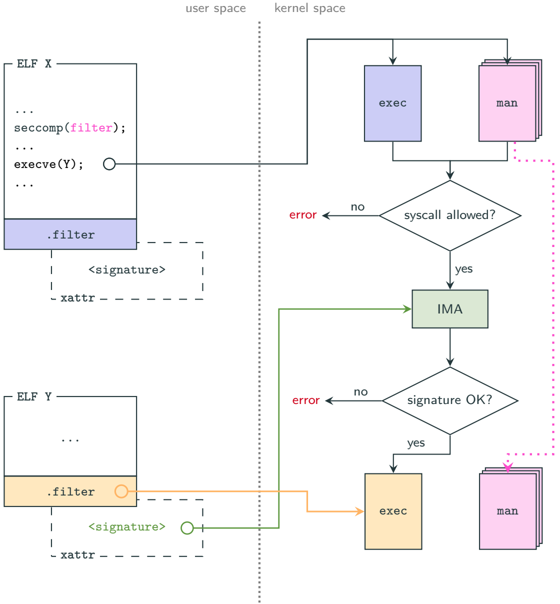


BROWN

Exchange Example



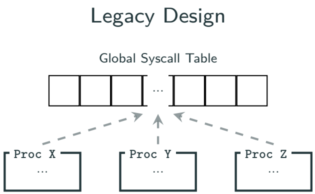
Exchange Example

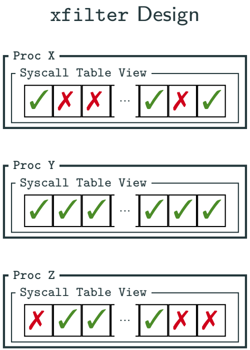


Performance Considerations

`xfilter`: optimize filter installation and runtime performance xfilter: optimize filter installation and runtime performance

- Processes keep **filtered "views"** of the syscall table



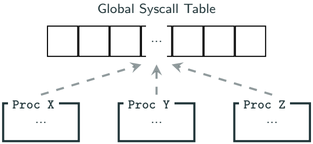


Performance Considerations

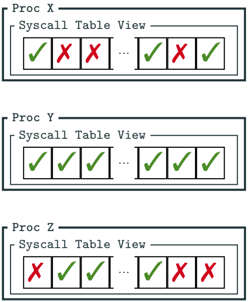
xfilter: optimize filter installation and runtime performance

- Processes keep **filtered "views"** of the syscall table
- Works with both the inheritance and exchange models

Legacy Design



xfilter Design

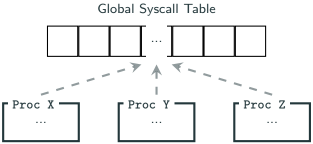


Performance Considerations

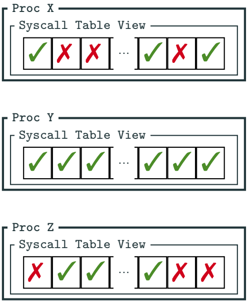
xfilter: optimize filter installation and runtime performance

- Processes keep **filtered "views"** of the syscall table
- Works with both the inheritance and exchange models
- Can be used in tandem with `Seccomp-BPF`

Legacy Design



xfilter Design



Implementation & Evaluation

Implementation

We have implemented our backwards-compatible design with minimal LOC

- Kernel extensions total ≈700 LOC added to:
  - `Seccomp-BPF` infrastructure
  - ELF execution/loading code
  - `prctl` syscall
  - Syscall handling pathway

- Userland enforcement framework totals ≈330 lines of Python, Bash, and C
- Tools used:
  1. Extraction: **`sysfilter`** to derive a syscall set
  2. Embedding: `objcopy` to embed `exec` filters
  3. Signing: **`evmctl`** to sign hardened binaries

BROWN

The exchange model **eliminates the over-privilege** that results from the inheritance model

Benchmark*	Depth	Root	No. Desc.	Total Syscalls		Over-privilege
				Inheritance	Exchange	Percentage
...	...	...	...	...	...	...
Bell Labs Unix50	1	`bash`	12	124	84	47.62%
COVID-19 Transit Analytics	1	`bash`	6	119	84	41.67%
Natural-Language Processing	1	`bash`	14	128	84	52.38%
NOAA Weather Analysis	1	`bash`	13	142	84	69.05%
↪	2	`xargs`	1	109	51	113.73%
↪	2	`sh`	1	77	68	13.24%
Wikipedia Web Indexing	1	`bash`	16	146	84	73.81%
Video Processing	1	`bash`	3	137	84	63.10%
Audio Processing	1	`bash`	3	173	84	105.95%
Program Inference	1	`bash`	2	129	84	53.57%
Traffic Log Analysis	1	`bash`	8	120	84	42.86%
PCAP Log Analysis	1	`bash`	6	135	84	60.71%
↪	2	`sh`	1	83	68	22.06%
Genomics Computation	1	`bash`	8	127	84	51.19%
Encryption	1	`bash`	3	124	84	47.62%
Compression	1	`bash`	3	112	84	33.33%
AUR Package Compilation	1	`bash`	69	176	84	109.52%
...	...	...	...	...	...	...
↪	2	`sh`	2	138	68	102.94%
...	...	...	...	...	...	...
↪	10	`make`	3	123	74	66.22%
↪	11	`collect2`	1	58	46	26.09%

*Kallas, Konstantinos, Tammam Mustafa, Jan Bielak, Dimitris Karnikis, Thurston HY Dang, Michael Greenberg, and Nikos Vasilakis. "Practically Correct, Just-in-Time Shell Script Parallelization." In the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 769–785. 2022.

The exchange model **reduces functionality** useful to an attacker

Benchmark	Depth	Root	No. Desc.	Over-privilege: Critical†	Over-privilege: Functionality
...	...	...	...	...	...
Bell Labs Unix50	1	bash	12	✓	stdio rpath chown fattr id proc vminfo unix inet
COVID-19 Transit Analytics	1	bash	6	✓	stdio chown fattr id proc vminfo unix inet
Natural-Language Processing	1	bash	14	✓	stdio {c,r,w,tmp}path chown fattr id proc vminfo unix inet
NOAA Weather Analysis	1	bash	13	✓	stdio chown {c,r}path fattr flock id proc protexec vminfo unix inet
↪	2	xargs	1	✓	stdio {c,r,tmp}path fattr flock id proc protexec unix inet
↪	2	sh	1	✗	stdio {c,tmp}path chown fattr
Wikipedia Web Indexing	1	bash	16	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec
Video Processing	1	bash	3	✓	stdio {c,r}path chown fattr id proc protexec unix inet
Audio Processing	1	bash	3	✓	stdio {c,r,w}path chown fattr flock id proc protexec unix inet
Program Inference	1	bash	2	✓	stdio {r,w,tmp}path chown fattr id proc protexec
Traffic Log Analysis	1	bash	8	✓	stdio rpath chown fattr id proc vminfo unix inet
PCAP Log Analysis	1	bash	6	✓	stdio rpath chown fattr id proc protexec
↪	2	sh	1	✓	rpath id proc stdio
Genomics Computation	1	bash	8	✓	stdio rpath chown fattr id proc protexec
Encryption	1	bash	3	✓	stdio{c,r}path chown fattr id proc protexec unix inet
Compression	1	bash	3	✓	stdio {c,r}path chown fattr id proc
AUR Package Compilation	1	bash	69	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec vminfo unix inet settime
...	...	...	...	...	...
↪	2	sh	2	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec unix inet
...	...	...	...	...	...
↪	10	make	3	✓	stdio {c,d,r,w}path chown fattr id proc
↪	11	collect2	1	✓	stdio {c,tmp}path fattr

† Ghavamnia, Seyedhamed, Tapti Palit, Shachee Mishra, and Michalis Polychronakis. "Temporal system call specialization for attack surface reduction." In Proceedings of the 29th USENIX Conference on Security Symposium, pp. 1749–1766. 2020.

Both filter exchanging and `xfilter` are **performant**

Both filter exchanging and `xfilter` are **performant**

i. `xfilter` offers a ≈76**%**–97**% reduction in filter install time** over `Seccomp-BPF`
   - SPEC CPU 2017, Nginx, MariaDB, Redis, and SQLite

Performance Results

Both filter exchanging and `xfilter` are **performant**

i. `xfilter` offers a ≈76**%**–97**%** **reduction in filter install time** over `Seccomp-BPF`

- ✳ SPEC CPU 2017, Nginx, MariaDB, Redis, and SQLite

ii. `xfilter` filtering performance is **equal to or better than** `Seccomp-BPF`

- ✳ SPEC CPU 2017

  `xfilter`: $\leq 0.4\%$
  `Seccomp-BPF`: $\leq 0.4\%$

- ✳ Real-world Applications (Nginx, MariaDB, Redis, SQLite)

  `xfilter`: $\leq 1.1\%$
  `Seccomp-BPF`: $\leq 1.1\%$

BROWN

Both filter exchanging and `xfilter` are **performant**

i. `xfilter` offers a ≈76**%**–97**%** **reduction in filter install time** over `Seccomp-BPF`
   - SPEC CPU 2017, Nginx, MariaDB, Redis, and SQLite

ii. `xfilter` filtering performance is **equal to or better than** `Seccomp-BPF`
   - SPEC CPU 2017

     `xfilter`: $\leq$ 0.4%
     `Seccomp-BPF`: $\leq$ 0.4%

   - Real-world Applications (Nginx, MariaDB, Redis, SQLite)

     `xfilter`: $\leq$ 1.1%
     `Seccomp-BPF`: $\leq$ 1.1%

iii. Filter exchanging overhead is **negligible**
   - PaSH

     `xfilter`: $\leq$ 1.7%
     `Seccomp-BPF`: $\leq$ 2.7%

Conclusion

Conclusion

SysXCHG: a syscall filtering enforcement mechanism that prevents over-privilege by securely exchanging filters at runtime.

- Exchange model adapts a process' privileges to current program by swapping filters on `execve`
- `xfilter` provides an optimal number-based enforcement mechanism with syscall table "views"
- Exchange model reduces kernel's attack surface and functionality useful to an attacker
- Filter exchanging (`xfilter`) incurs $\leq$ 1.7% overhead

https://gitlab.com/brown-ssl/sysxchg

BROWN

Backup

Evaluation Overview

We have concluded that `SysXCHG` is **effective** and **performant** through a series of experiments

- Effectiveness:
  i. How much does filter exchanging reduce the **attack surface** of the kernel?
  ii. What **type of unnecessary functionality** does the exchange model reduce?
  iii. Is any functionality **security critical**?
- Performance:
  i. Does `xfilter` speed up **filter installation** over `Seccomp-BPF`?
  ii. Does `xfilter` **filter faster** than `Seccomp-BPF`?
  iii. Is **filter exchanging** performant?

BROWN

Benchmark Descriptions

- PaSH Benchmark Suite (*Effectiveness*, *Performance*)
  - Real-world applications/workloads
  - Many programs are executed → filter exchanging occurs frequently
- SPEC CPU 2017 (*Performance*)
  - CPU-intensive workloads that perform many syscalls
- Nginx, MariaDB, Redis, SQLite (*Performance*)
  - Real-world applications
  - Demonstrate worst-case overheads

BROWN

Benchmark Testbed

- Intel Xeon W-2145 8-core (16-thread) processor
- 64GB of DDR4 memory
- Debian v11 (“`bullseye`”) Linux with kernel v6.0.8
- CPU fixed at 3.7GHz in C0 C-state
- DVFS (Intel Turbo Boost, Intel SpeedStep) was disabled
- Machine operated in single-user mode with a minimal number of processes
- Simultaneous multithreading (SMT) was enabled
- ASLR was enabled
- All binaries were built position-independent
- Speculative store bypass mitigations were always disabled for `Seccomp-BPF`

(Full) Over-privilege Quantification

Benchmark	Depth	Root	No. Desc.	Total Syscalls		Over-privilege
				Inheritance	Exchange	Percentage
Common Unix One-liners						
`bi-grams`	1	`bash`	11	122	84	45.24%
`diff`	1	`bash`	6	109	84	29.76%
`nfa-regex`	1	`bash`	3	100	84	19.05%
`set-diff`	1	`bash`	7	109	84	29.76%
`shortest-scripts`	1	`bash`	8	108	84	28.57%
↪	2	`xargs`	1	52	51	1.96%
↪	2	`xargs`	1	60	51	17.65%
`sort-sort`	1	`bash`	3	102	84	21.43%
`sort`	1	`bash`	2	102	84	21.43%
`spell`	1	`bash`	7	106	84	26.19%
`top-n`	1	`bash`	5	115	84	36.90%
`wf`	1	`bash`	4	103	84	22.62%
Bell Labs Unix50	1	`bash`	12	124	84	47.62%
COVID-19 Transit Analytics	1	`bash`	6	119	84	41.67%
Natural-Language Processing	1	`bash`	14	128	84	52.38%
NOAA Weather Analysis	1	`bash`	13	142	84	69.05%
↪	2	`xargs`	1	109	51	113.73%
↪	2	`sh`	1	77	68	13.24%
Wikipedia Web Indexing	1	`bash`	16	146	84	73.81%
Video Processing	1	`bash`	3	137	84	63.10%
Audio Processing	1	`bash`	3	173	84	105.95%
Program Inference	1	`bash`	2	129	84	53.57%
Traffic Log Analysis	1	`bash`	8	120	84	42.86%
PCAP Log Analysis	1	`bash`	6	135	84	60.71%
↪	2	`sh`	1	83	68	22.06%
Genomics Computation	1	`bash`	8	127	84	51.19%
Encryption	1	`bash`	3	124	84	47.62%
Compression	1	`bash`	3	112	84	33.33%
AUR Package Compilation	1	`bash`	69	176	84	109.52%
...	...	...	...	...	...	...
↪	2	`sh`	2	138	68	102.94%
...	...	...	...	...	...	...
↪	10	`make`	3	123	74	66.22%
↪	11	`collect2`	1	58	46	26.09%

BROWN

(Full) Over-privilege Characterization

Benchmark	Depth	Root	No. Desc.	Over-privilege		
				Percentage	Critical‡	Functionality
Common Unix One-liners						
bi-grams	1	bash	11	45.24%	✓	stdio {c,d,r}path chown fattr id proc
diff	1	bash	6	29.76%	✓	stdio {c,d,r}path fattr id proc
nfa-regex	1	bash	3	19.05%	✓	stdio rpath id proc
set-diff	1	bash	7	29.76%	✓	stdio {c,d,r}path fattr id proc
shortest-scripts	1	bash	8	28.57%	✓	stdio rpath fattr id proc
↪	2	xargs	1	1.96%	✗	stdio
↪	2	xargs	1	17.65%	✓	stdio {c,tmp}path fattr proc
sort-sort	1	bash	3	21.43%	✓	stdio id proc
sort	1	bash	2	21.43%	✓	stdio id proc
spell	1	bash	7	26.19%	✓	stdio id proc
top-n	1	bash	5	36.90%	✓	stdio chown fattr id proc
wf	1	bash	4	22.62%	✓	stdio id proc
Bell Labs Unix50	1	bash	12	47.62%	✓	stdio rpath chown fattr id proc vminfo unix inet
COVID-19 Transit Analytics	1	bash	6	41.67%	✓	stdio chown fattr id proc vminfo unix inet
Natural-Language Processing	1	bash	14	52.38%	✓	stdio {c,r,w,tmp}path chown fattr id proc vminfo unix inet
NOAA Weather Analysis	1	bash	13	69.05%	✓	stdio chown {c,r}path fattr flock id proc protexec vminfo unix inet
↪	2	xargs	1	113.73%	✓	stdio {c,r,tmp}path fattr flock id proc protexec unix inet
↪	2	sh	1	13.24%	✗	stdio {c,tmp}path chown fattr
Wikipedia Web Indexing	1	bash	16	73.81%	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec
Video Processing	1	bash	3	63.10%	✓	stdio {c,r}path chown fattr id proc protexec unix inet
Audio Processing	1	bash	3	105.95%	✓	stdio {c,r,w}path chown fattr flock id proc protexec unix inet
Program Inference	1	bash	2	53.57%	✓	stdio {r,w,tmp}path chown fattr id proc protexec
Traffic Log Analysis	1	bash	8	42.86%	✓	stdio rpath chown fattr id proc vminfo unix inet
PCAP Log Analysis	1	bash	6	60.71%	✓	stdio rpath chown fattr id proc protexec
↪	2	sh	1	22.06%	✓	rpath id proc stdio
Genomics Computation	1	bash	8	51.19%	✓	stdio rpath chown fattr id proc protexec
Encryption	1	bash	3	47.62%	✓	stdio{c,r}path chown fattr id proc protexec unix inet
Compression	1	bash	3	33.33%	✓	stdio {c,r}path chown fattr id proc
AUR Package Compilation	1	bash	69	109.52%	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec vminfo unix inet settime
...	...	...	...	...	...	...
↪	2	sh	2	102.94%	✓	stdio {c,d,r,w,tmp}path chown fattr id proc protexec unix inet
...	...	...	...	...	...	...
↪	10	make	3	66.22%	✓	stdio {c,d,r,w}path chown fattr id proc
↪	11	collect2	1	26.09%	✓	stdio {c,tmp}path fattr

Descendant Listings

Benchmark	Depth	Root	Descendants
Common Unix One-liners			
bi-grams	1	bash	cat, mkfifo, mktemp, paste, rm, sed, sort, tail, tee, tr, uniq
diff	1	bash	cat, diff, mkfifo, rm, sort, tr
nfa-regex	1	bash	cat, grep, tr
set-diff	1	bash	cat, comm, cut, mkfifo, rm, sort, tr
shortest-scripts	1	bash	cat, cut, file, grep, head, sort, wc, xargs
↪	2	xargs	file
↪	2	xargs	wc
sort-sort	1	bash	cat, sort, tr
sort	1	bash	cat, sort
spell	1	bash	cat, col, comm, iconv, sort, tr, uniq
top-n	1	bash	cat, sed, sort, tr, uniq
wf	1	bash	cat, sort, tr, uniq
Bell Labs Unix50	1	bash	awk, cat, cut, fmt, grep, head, sed, sort, tail, tr, uniq, wc
COVID-19 Transit Analytics	1	bash	awk, cat, cut, sed, sort, uniq
Natural-Language Processing	1	bash	awk, cat, grep, head, ls, mkdir, paste, rev, rm, sed, sort, tail, tr, uniq
NOAA Weather Analysis	1	bash	awk, cat, curl, cut, grep, gzip, head, sed, seq, sh, sort, tr, xargs
↪	2	xargs	curl
↪	2	sh	gzip
Wikipedia Web Indexing	1	bash	cat, cut, grep, iconv, mkfifo, mktemp, node, pandoc, paste, rm, sed, sort, tail, tee, tr, uniq
Video Processing	1	bash	basename, convert, mkdir
Audio Processing	1	bash	basename, ffmpeg, mkdir
Program Inference	1	bash	mkdir, node
Traffic Log Analysis	1	bash	awk, basename, cat, cut, head, mkdir, sort, uniq
PCAP Log Analysis	1	bash	grep, mkdir, sh, sort, tcpdump, uniq
↪	2	sh	grep
Genomics Computation	1	bash	cat, cut, mkdir, samtools, sed, sort, tr, uniq
Encryption	1	bash	basename, mkdir, openssl
Compression	1	bash	basename, mkdir, zip
AUR Package Compilation	1	bash	arch, ar, as, awk, basename, bash, bsdtar, cat, cc1, cc, chmod, chown, cmp, collect2, cpp, cp, cut, date, echo, expr, faked-sysv, file, find, gcc, getopt, gettext, git-upload-pack, git, gmake, gpg, grep, gzip, head, hostname, install, jar, ld, ln, logname, ls, make, md5sum, mkdir, mktemp, msgfmt, msgmerge, mv, nawk, nm, patch, pkg-config, python3, readelf, rm, sed, sha1sum, sha256sum, sh, sort, split, strip, test, touch, tr, uname, uniq, wc, xgettext, zstd
…	…	…	…
↪	2	sh	bash, bsdtar
…	…	…	…
↪	10	make	install, make, sh
↪	11	collect2	ld

Syscall Functionality Descriptors

- Syscall functionality descriptors based on OpenBSD `pledge` promises

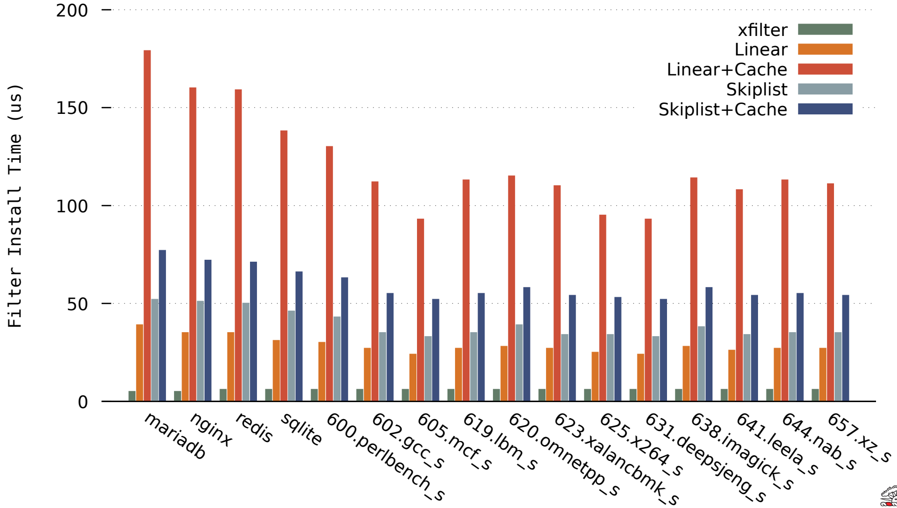
Functionality	Description
`stdio`	standard input/output functionality
`cpath`	create new files or directories
`dpath`	create special files
`rpath`	read-only effects on file system
`wpath`	write-only effects on file system
`tmppath`	create/read/write in `/tmp`
`chown`	change file ownership
`fattr`	modify file attributes
`flock`	file locking functionality
`id`	change rights of a process
`proc`	operations for processes
`protexec`	create executable memory regions
`vminfo`	operations for virtual memory inspection
`unix`	socket programming for `AF_UNIX`
`inet`	socket programming for `AF_INET[6]`
`settime`	set system time

BROWN

- Critical syscalls from Ghavamnia et al.

clone	execveat	execve	fork	ptrace
chmod	fchmodat	mprotect	setgid	setreuid
setuid	accept4	accept	bind	connect
listen	recv	recvfrom	read	socket
send	sendto	write	dup	dup2
dup3	eventfd	eventfd2	open	openat
select	pselect6	epoll_wait	epoll_wait_old	poll
ppoll	epoll_pwait			

Install Timing



BROWN

(Full) SPEC CPU 2017 Performance Results

- Performance results for SPEC CPU 2017 using the inheritance model

Benchmark	Seccomp-BPF		xfilter	
	Man.	Exec.	Man.	Exec.
600.perlbench_s	≈0%	≈0%	≈0%	≈0%
602.gcc_s	≈0%	0.44%	≈0%	0.33%
605.mcf_s	≈0%	0.37%	≈0%	0.29%
620.omnetpp_s	≈0%	0.34%	≈0%	≈0%
623.xalancbmk_s	≈0%	≈0%	0.11%	0.36%
625.x264_s	≈0%	0.12%	≈0%	0.09%
631.deepsjeng_s	0.02%	0.02%	0.38%	0.02%
641.leela_s	≈0%	0.02%	≈0%	0.05%
657.xz_s	≈0%	≈0%	≈0%	≈0%
619.lbm_s	0.14%	≈0%	0.05%	≈0%
638.imagick_s	0.09%	0.04%	0.04%	≈0%
644.nab_s	0.02%	0.07%	≈0%	≈0%

(Full) Real-world Performance Results

- Performance results for real-world applications using the inheritance model (`exec` filters only).

Benchmark	`Seccomp-BPF`	`xfilter`
Nginx (`1KB`)	1.08%	0.07%
Nginx (`100KB`)	0.54%	1.10%
Nginx (`1MB`)	1.07%	1.11%
Redis (`GET`)	0.53%	0.26%
Redis (`SET`)	0.53%	0.26%
MariaDB	0.80%	0.50%
SQLite	≈0%	≈0%

BROWN

(Full) PaSH Performance Results

- Performance results for PaSH using the exchange model

Benchmark	Seccomp-BPF	xfilter
Common Unix One-liners		
bi-grams	6.02%	3.30%
diff	6.57%	0.84%
nfa-regex	7.21%	0.90%
set-diff	7.81%	0.69%
shortest-scripts	1.42%	0.34%
sort-sort	7.11%	0.36%
sort	6.38%	1.08%
spell	1.77%	0.37%
top-n	2.41%	≈0%
wf	2.54%	0.82%
Bell Labs Unix50	0.40%	0.39%
COVID-19 Transit Analytics	0.87%	0.78%
Natural-Language Processing	0.95%	0.51%
NOAA Weather Analysis	0.96%	0.88%
Wikipedia Web Indexing	0.34%	0.16%
Video Processing	0.43%	0.40%
Audio Processing	0.03%	0.05%
Program Inference	0.29%	0.12%
Traffic Log Analysis	1.11%	0.30%
PCAP Log Analysis	0.25%	0.31%
Genomics Computation	≈0%	0.18%
Encryption	0.29%	0.10%
Compression	≈0%	≈0%
AUR Package Compilation	2.74%	1.71%

BROWN

Adversarial Capabilities

- No constraints on the types of vulnerabilities (ab)used by the attacker
- No constraints on the applied exploitation technique
- Ultimately, we assume an attacker that can:
  1. Invoke **any syscall**
  2. Pass **arbitrary arguments**
  3. Repeatedly perform i. and ii. at **arbitrary times**
- On par with state-of-the-{art, practice} regarding syscall filtering

Hardening Assumptions

- Linux kernel with support for `Seccomp-BPF` and IMA (neither can be disabled)
- Target applications contain benign code
- Standard userland hardening schemes are orthogonal to our scheme
- Given this, an attacker can attempt to:
  1. **Elevate their privileges** by finding and exercising vulnerabilities in syscalls
  2. Maliciously **request unintended OS services** (post-exploitation)

Example Seccomp-BPF Filter Program

- struct of syscall information passed to Seccomp-BPF programs:

```c
struct seccomp_data {
  int nr; /* syscall number */
  u32 arch; /* syscall convention */
  u64 instruction_pointer; /* next insn */
  u64 args[6]; /* syscall arguments */
};
```

- Simple Seccomp-BPF program that enforces the syscall set of read (0), write (1), exit (15), and sigreturn (60) via linear search.

```c
#define NRMAX (X32_SYSCALL_BIT - 1)
#define ALLOW SECCOMP_RET_ALLOW
#define DENY SECCOMP_RET_KILL_PROCESS

struct sock_filter filter[] = {
  /* ... check arch ... */
  BPF_JUMP(BPF_JMP | BPF_JGT | BPF_K, NRMAX , 5, 0),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 0, 3, 0),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 1, 2, 0),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 15 , 1, 0),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 60 , 0, 1),
  BPF_STMT(BPF_RET | BPF_K, ALLOW),
  BPF_STMT(BPF_RET | BPF_K, DENY)
};
```

Syscall Filtering Classifications

Syscall filtering schemes can be classified based on where filtering decision-making and enforcement takes place.

i. Both occur in **user mode**
   - ✳ Usually done with process tracing utilities (e.g., `ptrace(2)`)
   - ✳ Easy to test/deploy (no superuser privilege or kernel recompilation required)
   - ✳ Adds no additional code to the kernel
   - ✳ Suffers from poor performance (adds 2–4 context switches per syscall)

ii. Both occur in **kernel mode**
   - ✳ Done by modifying kernel source directly or relying on `Seccomp-BPF`
   - ✳ Provides optimal performance
   - ✳ Provides greater visibility/control over the system

iii. Both occur in a **hypervisor**
   - ✳ OS-transparent and allows filtering in the presence of an untrusted OS
   - ✳ Suffers from unnecessary complexity
   - ✳ Suffers from poor visibility (it is removed from the interface it interposes)

iv. A **hybrid** of i.–iii.
   - ✳ Can minimize code added to sensitive areas
   - ✳ Suffers from poor performance due to additional context switches

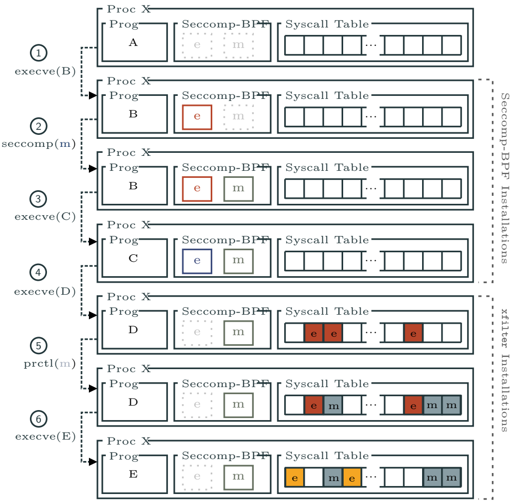
BROWN

- Capability-based sandboxes only allow a program to access a resource if they have an unforgeable reference to it plus the appropriate rights
- While related to syscall filtering, we consider capability systems a separate line of work
- Capability systems never fully block operating system functionality, as a process can confer capabilties to another, granting increased privilege
- Syscall filtering schemes aim to completely block OS functionality, permanently slimming the amount of accessible code paths

BROWN

xfilter Compatibility with Seccomp-BPF

- xfilter can be used in tandem with Seccomp-BPF
  e.g., xfilter can filter based on numbers and Seccomp-BPF on arguments



Examples of Software Bloat

- Windows 95 ($\approx$50MB)[§] $\rightarrow$ Windows 11 ($\approx$64GB)[¶]
- `/bin/true`[‖]: 1979 (0B) $\rightarrow$ 2012 (22KB)

Table 1: Static bloat measurement. LIR: Percentage of instructions in all the dependent libraries that a program *may* execute. OIR: Percentage of instructions in all the libraries + executable that a program *may* execute. LFR: Percentage of functions in all the dependent libraries that a program *may* execute. OFR: Percentage of functions in all the libraries + executable that a program *may* execute.

Program	% Library Instructions Required (LIR)	% Overall Instructions Required (OIR)	% Library Functions Required (LFR)	% Overall Functions Required (OFR)
firefox	67.20%	68.37%	36.42%	38.60%
chrome	69.72%	95.67%	33.57%	36.75%
webbrowser-app	58.86%	59.03%	29.34%	30.22%
vlc	78.22%	78.25%	42.44%	42.79%
rhythmbox	77.92%	77.92%	29.83%	29.83%
evince	70.84%	71.34%	33.61%	36.19%
sublime	68.88%	84.95%	39.13%	41.42%
gnome calculator	68.59%	69.21%	34.02%	36.18%
git	62.70%	78.11%	22.75%	29.11%
clang	53.99%	73.91%	34.32%	56.83%
g++	52.36%	64.37%	23.90%	29.58%
make	52.13%	56.06%	23.11%	27.75%
Average	**65.11%**	**73.01%**	**31.87%**	**36.27%**

[**]

[§] http://support.microsoft.com/kb/138349/

[¶] https://www.theverge.com/microsoft/22544171/microsoft-windows-11-system-requirements-hardware

[‖] https://spinroot.com/gerard/pdf/Code_Inflation.pdf

[**] Quach, Anh, Rukayat Erinfolami, David Demicco, and Aravind Prakash. "A multi-OS cross-layer study of bloating in user programs, kernel and managed execution environments." In Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation, pp. 65-70. 2017.