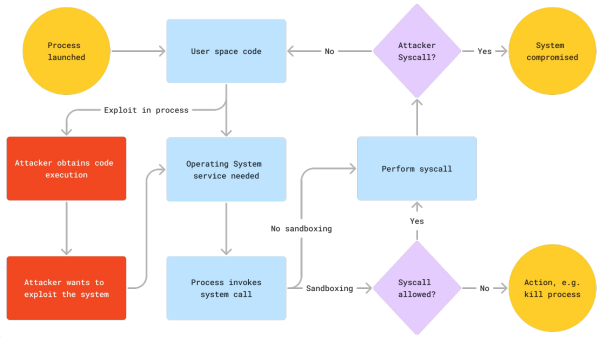
An analysis of system call set extraction tools on configurable Linux binaries

Comparing the performance of various system call set extraction tools on various configurations of the busybox application

Bryan van der Mark
b.b.vandermark@student.tudelft.nl

Responsible professor: Alexios Voulimeneas

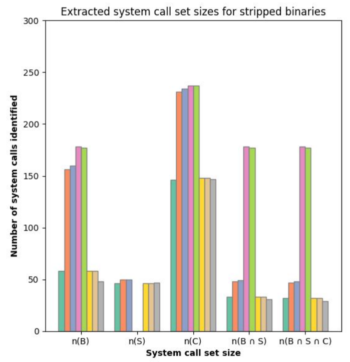
Background

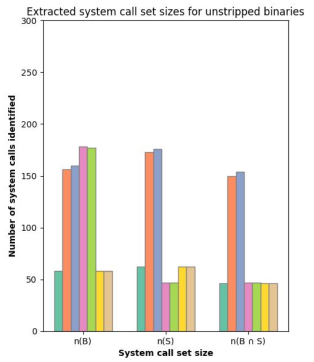

Process launched
User space code
No
Attacker Syscall?
Yes
System compromised
Exploit in process
Attacker obtains code execution
Operating System service needed
Perform syscall
No sandboxing
Yes
Attacker wants to exploit the system
Process invokes system call
Sandboxing
Syscall allowed?
No
Action, e.g. kill process

System calls are a primary way in which applications communicate with the kernel. By default, Linux allows a process to invoke every system call. By sandboxing the process and allowing only the system calls that are required for functionality, we can reduce the attack surface of an application. Getting this set of system calls is difficult to do by hand, and therefore automated tools have been developed to automate this process. In this research, we focus on Chestnut's Binalyzer[1], Sysfilter[2] and Confine[3]. Chestnut and Sysfilter perform static analysis on binaries, while Confine analyses a Docker container and analyzes all binaries ran during the container's lifetime.

Busybox is a binary which contains lightweight versions of several other binaries, intended for use in embedded applications. Busybox is configurable, in the sense that a developer is able to choose which applications (called applets) get included in the compilation process.

Results





B - Set extracted by Binalyzer
S - Set extracted by Sysfilter
C - Set extracted by Confine
B ∩ S - The intersection of B and S in cases where n(S)≠0
The legend to the right is used for both figures

- 🟩 busybox_minimal
- 🟧 busybox_default
- 🟦 busybox_all_applets_default_settings
- 🟪 busybox_maximal_excluding_debugging_options
- 🟩 busybox_maximal_including_debugging_options
- 🟨 busybox_cat_minimal
- 🟫 busybox_cat
- ⬜ cat

Research question

How do Chestnut's Binalyzer, Sysfilter and Confine compare when performing system call set extraction on various configurations of the busybox application?

Conclusion

Sysfilter performs significantly better than Binalyzer on stripped binaries, however, it was not able to complete the analysis of the configuration where all settings were enabled. On unstripped binaries, Sysfilter was able to complete analysis of all binaries, but was outperformed by Binalyzer on all binaries except the ones it was not able to analyze without debug symbols. Sets extracted with Binalyzer did not change between stripped and unstripped binaries. Binalyzer scaled significantly worse than Sysfilter as more complexity was added to the binary. Confine performed worse than both Binalyzer and Sysfilter, and both scripts resulted in the same system call sets.

Methodology

Chosen applications

In this research, we analyze two applications: busybox and cat. Cat is used as a minimal example, and used to compare against busybox in the specific case where busybox is compiled to only include the functionality of cat. We also make a distinction between stripped and unstripped binaries.

Busybox configuration

We compile busybox in the following configurations:
Default: The default configuration of busybox
Minimal: A configuration containing a minimal amount of features
All applets: A configuration containing all possible applets while keeping default settings
Maximal: A configuration containing all applets, all settings are enabled.
Cat: Similar to Minimal, but with the cat applet included in compilation.
For the maximal configuration, we create a configuration with debugging options disabled and another with debugging options enabled. For Cat, we keep the default settings related to the applet and create another (Cat minimal) with all of these settings disabled.

Confine containers

In order to run confine, each binary needs to be run in a Docker container. To do this, we copy the binary to a Docker image and use a script to ensure the binary is running during the container's lifetime. We create multiple scripts and analyze Confine's performance on each one. One script continually invokes the binary, while another script attempts to mimic regular usage of the binary.

References

[1] Claudio Canella, Mario Werner, Daniel Gruss, and Michael Schwarz. Automating seccomp filter generation for linux applications, 2020.
[2] Nicholas DeMarinis, Kent Williams-King, Di Jin, Rodrigo Fonseca, and Vasileios P. Kemerlis. sysfilter: Automated system call filtering for commodity software. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), pages 459–474, San Sebastian, October 2020. USENIX Association.
[3] Seyedhamed Ghavamnia, Tapti Palit, Azzedine Benameur, and Michalis Polychronakis. Confine: Automated system call policy generation for container attack surface reduction. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), pages 443–458, San Sebastian, October 2020. USENIX Association.