accenture

Strategy | Consulting | Digital | Technology | Operations

New. Applied. Now.

Container Attack Surface Reduction Beyond Name Space Isolation

High performance. Delivered.

Accenture Labs, Security R&D
Stony Brook University

Azzedine Benameur
Jay Chen
Lei Ding
Michalis Polychronakis

Introduction

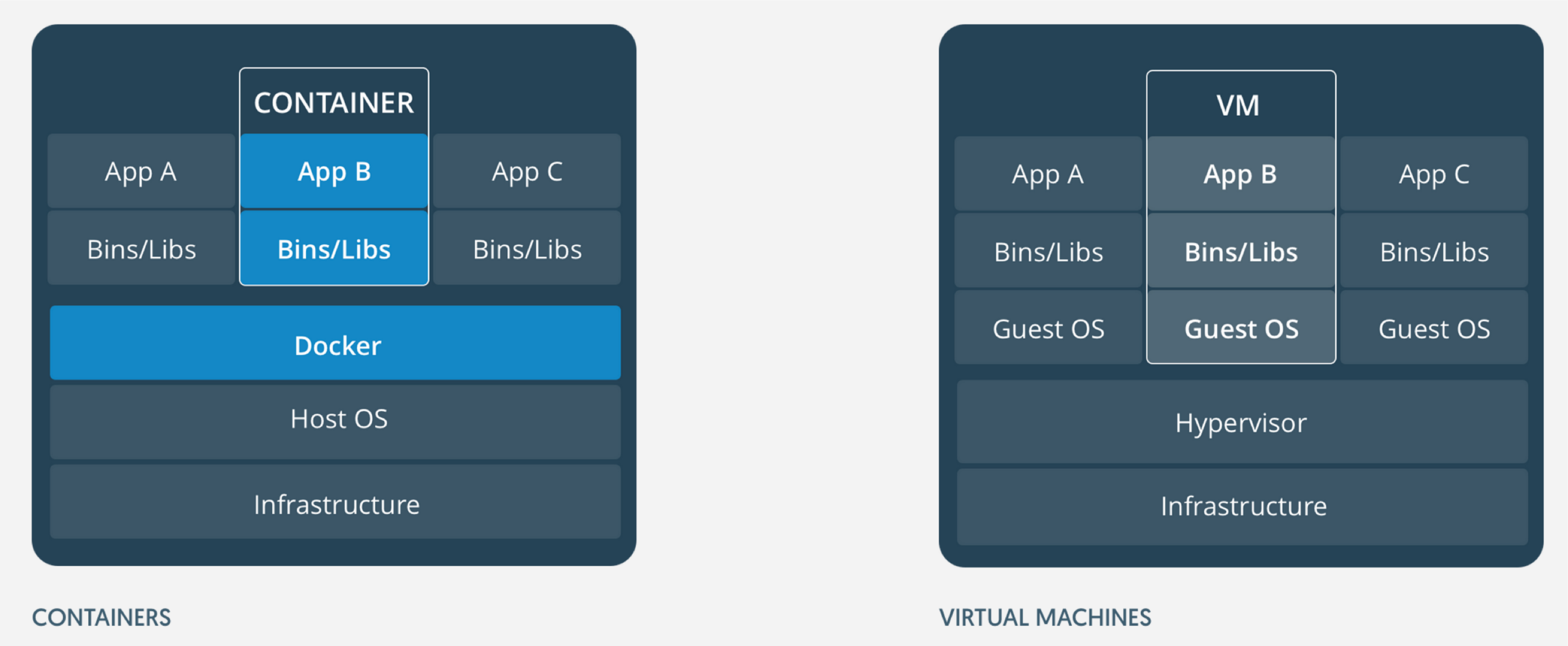
Who are we

- Azzedine Benameur, Jay Chen, Lei Ding:
  - Currently: Accenture Cyberlabs leading Attack Surface Reduction research
  - Past work: Mobile/Car/Cloud/Binary Security
    - Jay Chen:
      - Past work: ICS/Network/Blockchain security
    - Lei Ding:
      - Past work: Document Classification, Machine Learning

- Michalis Polychronakis:
  - Currently: Assistant professor in the Computer Science Department at Stony Brook University working on system security
  - Past work: Network/System Security

Introduction

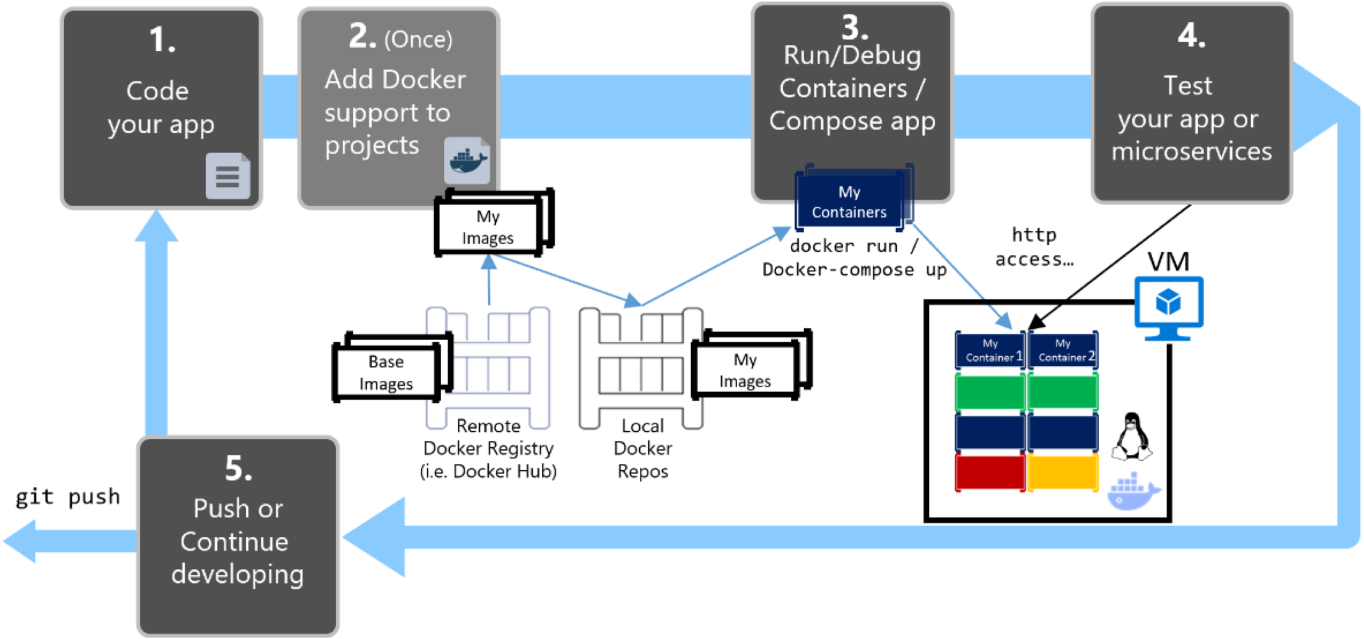
Container 101



Introduction

Container 101

VS development workflow for Docker apps



Introduction

Attack Surface Reduction

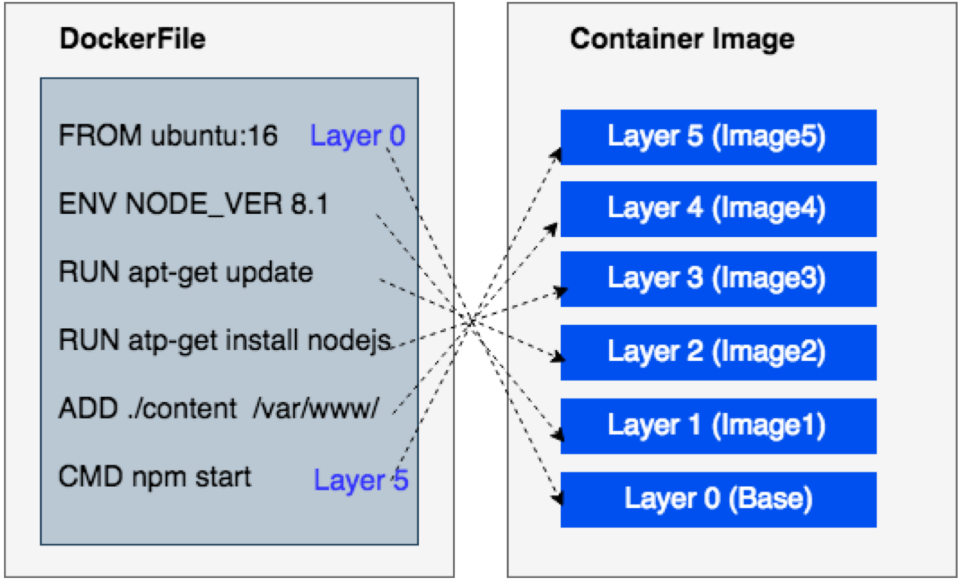
DeBloat

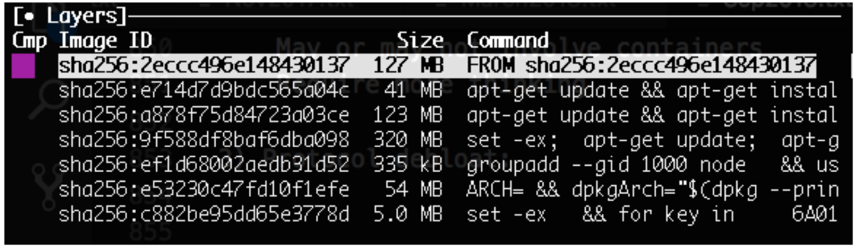
Containers
Applications

Smaller Containers
Fewer Vulnerabilities
Reduced Attack Surface

Attack Surface Reduction

The need





A container image consists of a stack of multiple layers and each layer contains the delta change from the previous layer

Attack Surface Reduction

The need

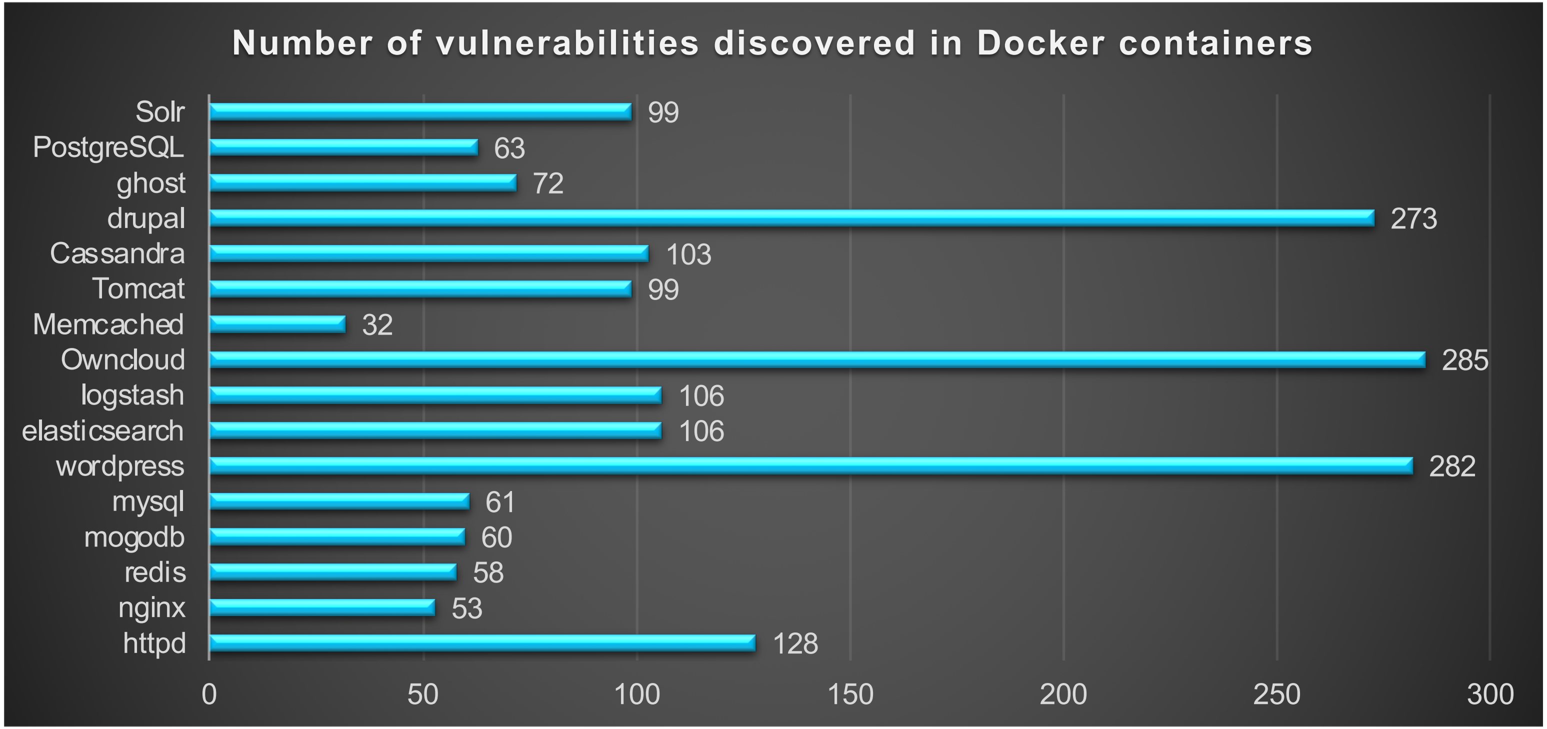
Image Name	# of vulnerabilities
rails-4.2.1	1820
perl-5.12	1770
iojs-3.0	1708
rocket.chat-0.30	1433
elixir-1.2.5	1408
redmine-3.0.4	1406
gcc-5.2.0	1361
pypy-2-5.4.1	1202
r-base	1068

Top 10 vulnerable images in Docker Hub

Package Name	# of vulnerabilities
imagemagick	142
binutils	129
mariadb	56
mysql	49
Jasper	48
openjdk	40
Libav	39
Ruby	36
Tomcat	36

Top 10 vulnerable package used

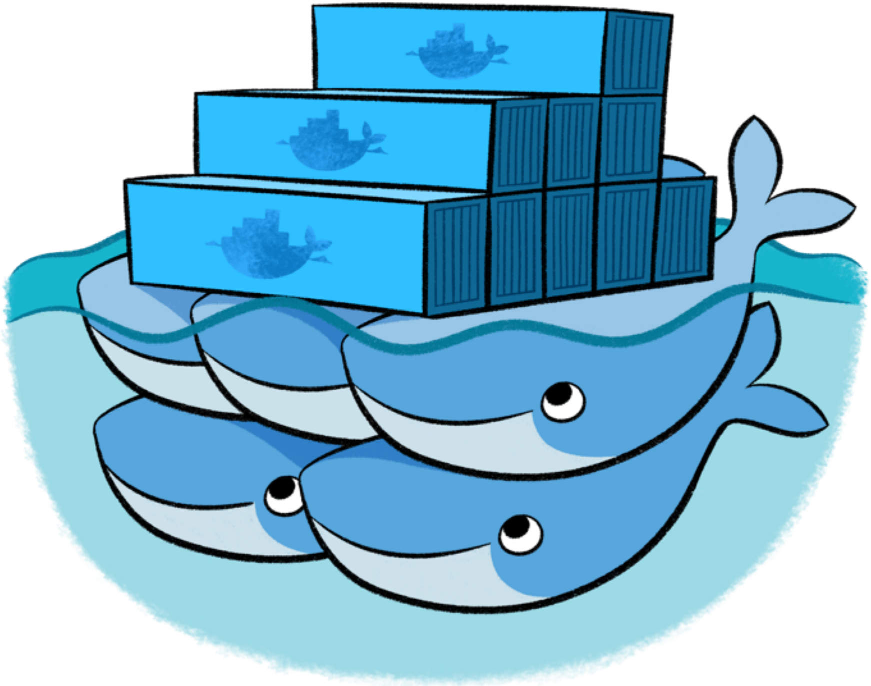
Attack Surface Reduction: The need



Evaluated using CoreOS/Clair, 2018

Attack Surface Reduction

Container Layer



Attack Surface Reduction: Approach

Container Layer

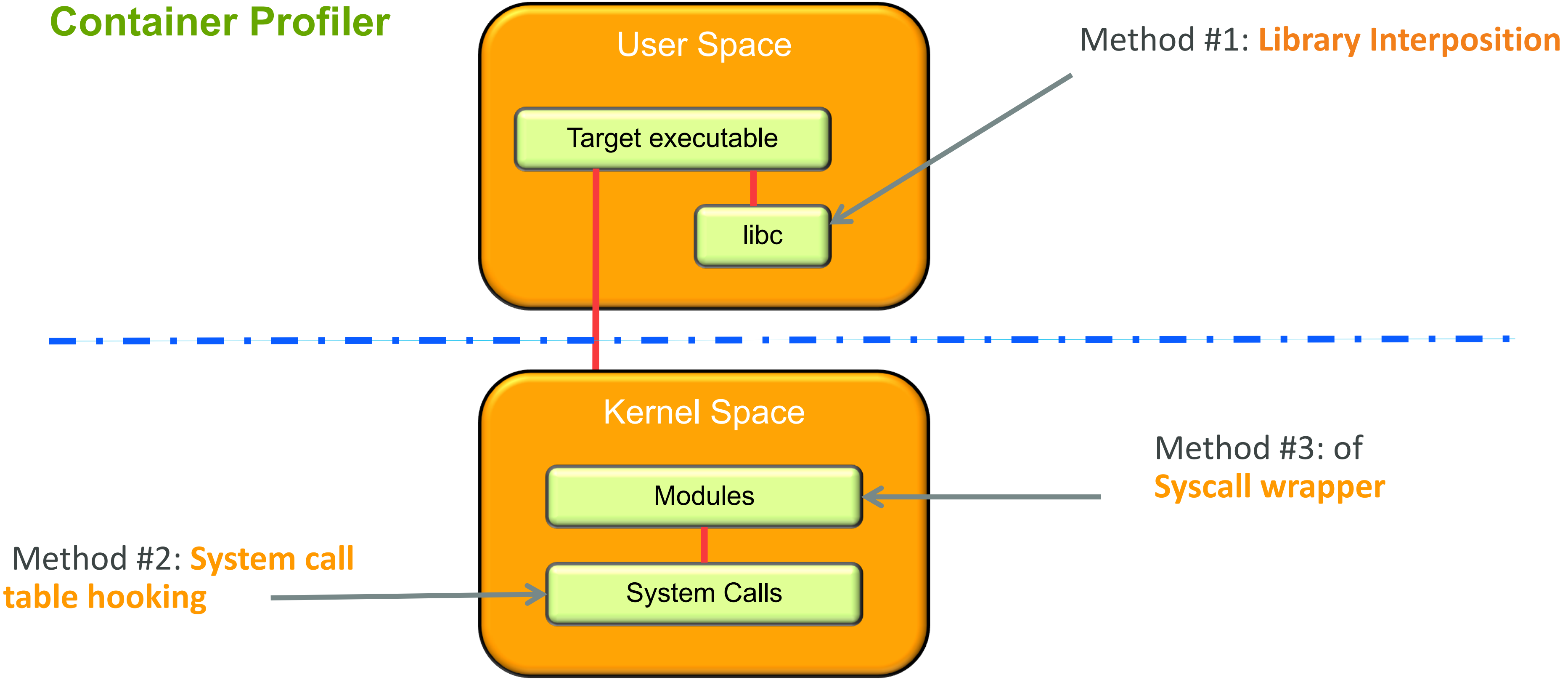
- Containers should be designed for a single purpose/application

- Shipped with many default packages/binaries that are not necessary for the operation of the aforementioned purpose

- We propose an Advanced Secure Lightweight Container
  - Unix Philosophy: Each container is atomic in nature and fulfills only one task: web server, database, file system etc.
  - Two Phase Approach:
    - 1- Profiling: Monitor and identify the required components during an application's execution
    - 2- Image Generation: Produce a **BNB** (**B**are **M**inimum **B**inaries) container image
  - The new image will be smaller in size and contains less vulnerabilities

Attack Surface Reduction: Approach

Container Profiler



Attack Surface Reduction: Approach

Container Profiler: File System

NAME

fanotify - monitoring filesystem events

DESCRIPTION

The fanotify API provides notification and interception of filesystem events. Use cases include virus scanning and hierarchical storage management. Currently, only a limited set of events is supported. In particular, there is no support for create, delete, and move events. (See **inotify**(7) for details of an API that does notify those events.)

Additional capabilities compared to the **inotify**(7) API include the ability to monitor all of the objects in a mounted filesystem, the ability to make access permission decisions, and the possibility to read or modify files before access by other applications.

The following system calls are used with this API: **fanotify_init**(2), **fanotify_mark**(2), **read**(2), **write**(2), and **close**(2).

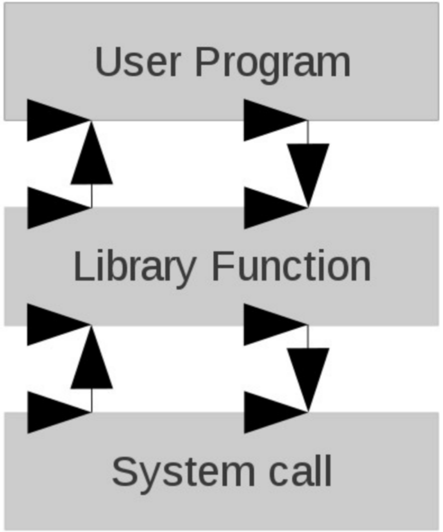
```
int mark = fanotify_mark(
    fan,
    FAN_MARK_ADD | FAN_MARK_MOUNT,
    FAN_ACCESS | FAN_MODIFY | FAN_CLOSE | FAN_OPEN | FAN_ONDIR | FAN_EVENT_ON_CHILD,
    AT_FDCWD, "/");
```

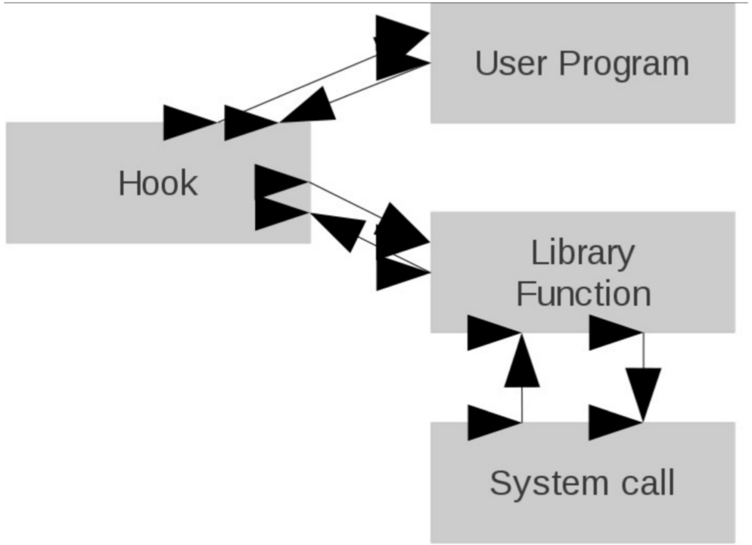
Attack Surface Reduction: Approach

Container Profiler: Library Calls

LD_PRELOAD

```
A list of additional, user-specified, ELF shared objects to be
loaded before all others.  The items of the list can be sepa-
rated by spaces or colons, and there is no support for escap-
ing either separator.  This can be used to selectively over-
ride functions in other shared objects.  The objects are
searched for using the rules given under DESCRIPTION.
```



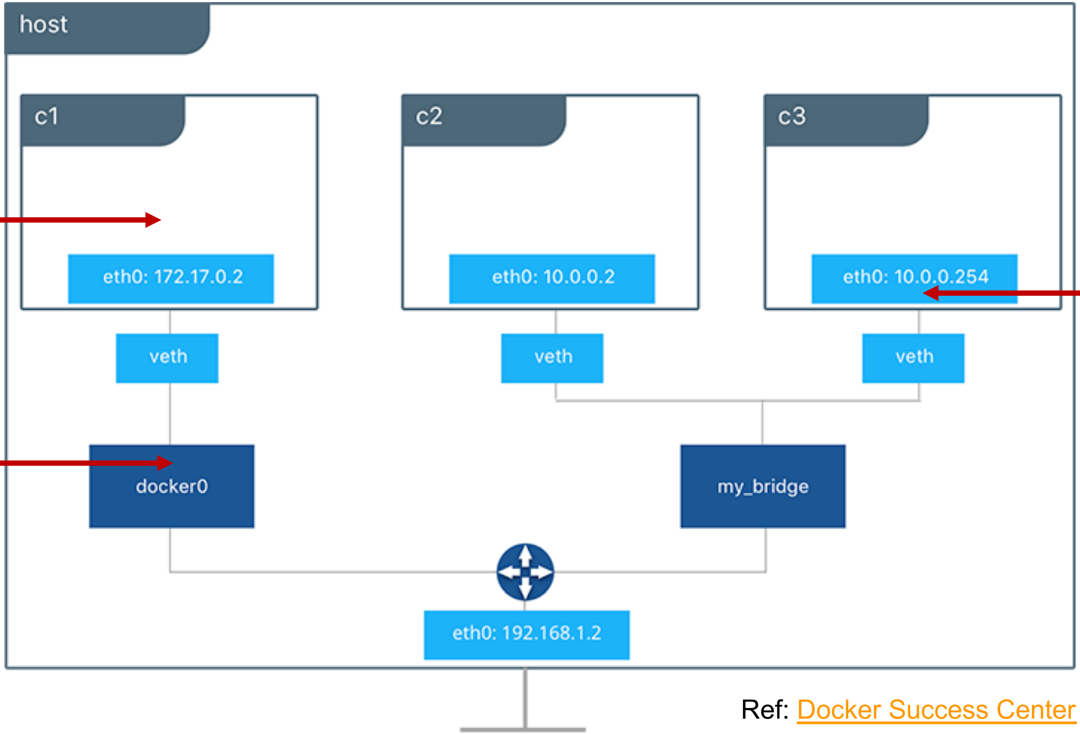


Attack Surface Reduction: Approach

Container Profiler: Network

Library interposition: tap into connect(), accept() in glibc

Tap into the bridge between the container and host network namespaces



Tap into the interface in the container network namespace

Ref: [Docker Success Center]

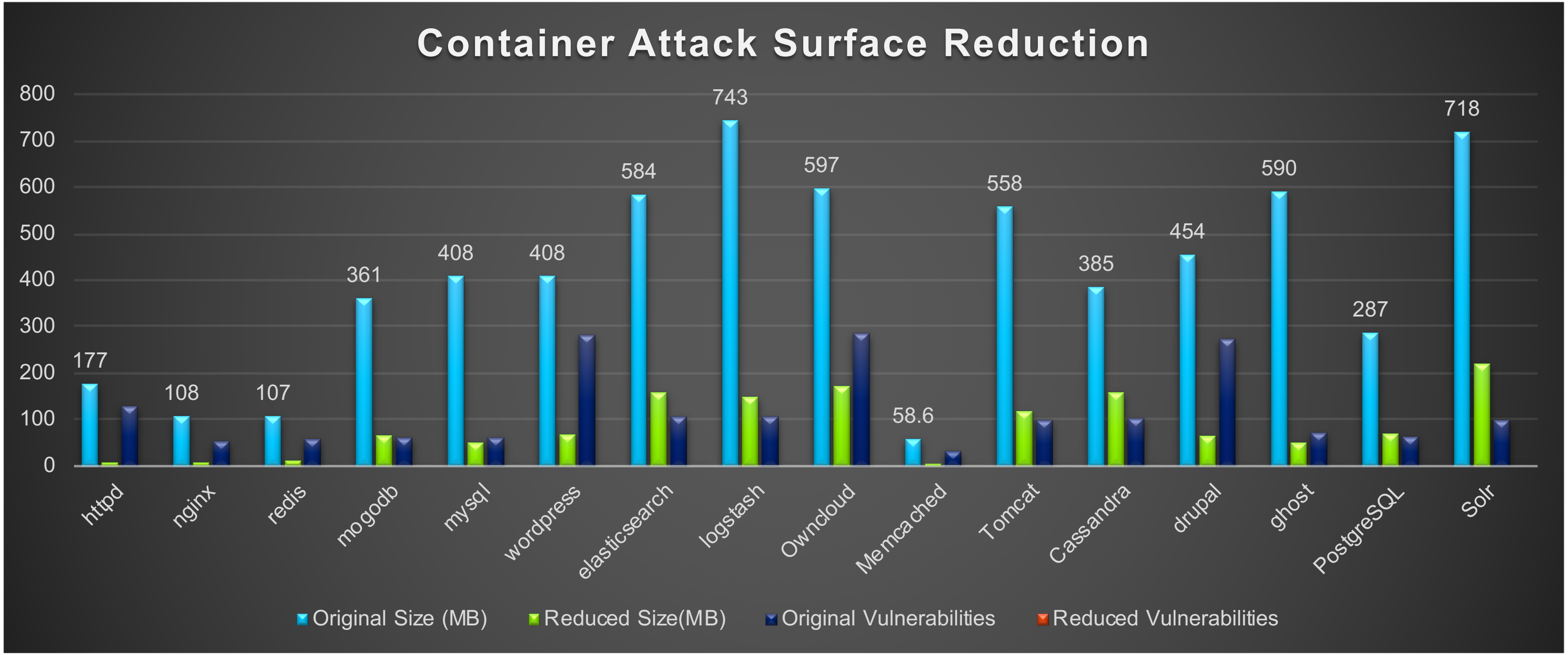
Attack Surface Reduction: Approach

Container Profiler

- User space approach:
  - Using library interposition
  - Leveraging Linux provided API for filesystem event notification/interception
- Phase 1:
  - One time container profiling at pre-production deployment
  - Profile built using a ”normal” workload
- Phase 2:
  - Continuous container profiling after production deployment
  - Enabling continuous refinement and updates
- Limitations:
  - As good as the profiling

Attack Surface Reduction: Demo

Preliminary Results



Evaluated using CoreOS/Clair, 2018

Container Vulnerability Scanning

Vulnerability scanning is a standard feature in most container service providers, but the majority of them perform only “shallow scan” at the package or image level.

- **How the scanner works:**
  - CVE database describes the vulnerable packages in a specific OS.
  - Scanner gathers a list of installed packages from the package manager (dpkg, rpm, pacman, …)
  - Scanner cross check the package list with the vulnerability database

Debian OVAL Definition

```xml
    <dpkginfo_test check="all" check_existence="at_least_one_exists" comment="squid is earlier than 2.5.7-1" id="oval:org.debian.oval:tst:3" version="1"
xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
      <object object_ref="oval:org.debian.oval:obj:3"/>
      <state state_ref="oval:org.debian.oval:ste:2"/>
    </dpkginfo_test>
    <dpkginfo_test check="all" check_existence="at_least_one_exists" comment="gzip is earlier than 1.3.5-6" id="oval:org.debian.oval:tst:4" version="1"
xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
      <object object_ref="oval:org.debian.oval:obj:4"/>
      <state state_ref="oval:org.debian.oval:ste:3"/>
    </dpkginfo_test>
    <dpkginfo_test check="all" check_existence="at_least_one_exists" comment="cpio is earlier than 2.5-1.2" id="oval:org.debian.oval:tst:5" version="1"
xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
      <object object_ref="oval:org.debian.oval:obj:5"/>
      <state state_ref="oval:org.debian.oval:ste:4"/>
    </dpkginfo_test>
```

Ubuntu CVE Tracker

CVE-2017-16832	binutils	needs-triage*	needs-triage*	needs-triage*	needs-triage*
CVE-2017-16845	qemu	DNE	not-affected*	released*	released*
CVE-2017-16879	ncurses	needs-triage*	needs-triage*	needs-triage*	needs-triage*
CVE-2017-16899	fig2dev	DNE	DNE	DNE	needed*
CVE-2017-16911	linux	ignored*	needed*	released*	needed*
CVE-2017-16911	linux-azure	DNE	DNE	needed*	DNE
CVE-2017-16911	linux-hwe	DNE	DNE	needed*	DNE
CVE-2017-16912	linux	ignored*	needed*	released*	needed*
CVE-2017-16912	linux-azure	DNE	DNE	needed*	DNE
CVE-2017-16912	linux-hwe	DNE	DNE	needed*	DNE

Attack Surface Reduction

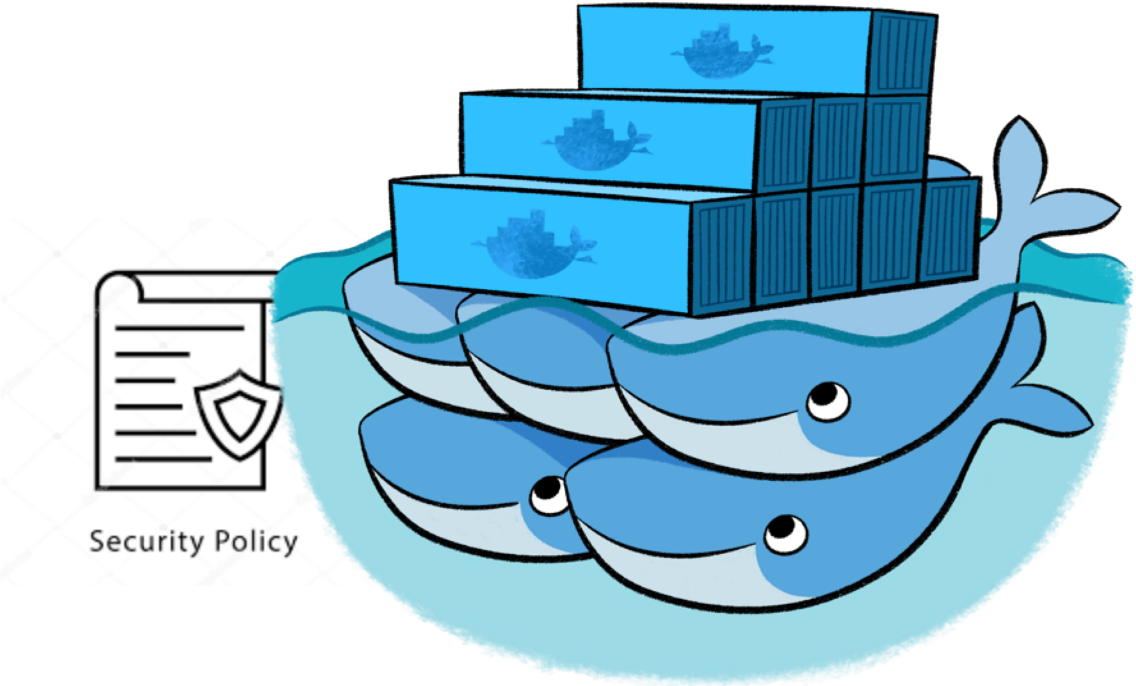
Issues

Through out our research, we were continuously being surprised how unreliable the existing container testing tools are. The bottom line is, you can’t trust what the vulnerability scanners tell you.

Huge reliance on the package managers :
Most scanners fail to identify known vulnerabilities in files such as python, javascript , php, or shared objects.
Most scanners fail to function properly when the package manager is removed.
Most scanners fail to scan images with Fedora or OpenSUSE base OS.
Most scanners fail to identify known malwares in the images.
None of scanners can identify known vulnerable files when the file names are changed.

Attack Surface Reduction

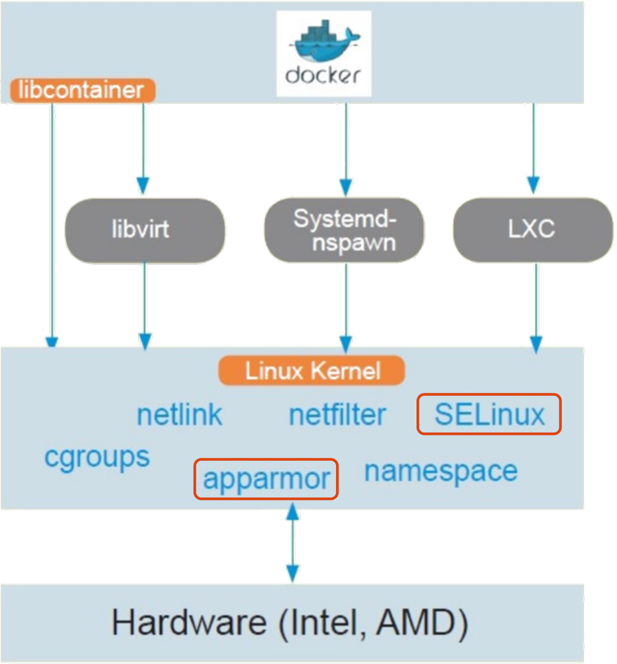
Automated Container Policy Generation



Mandatory Access Control (MAC) policy enforcement

Enforce MAC policies at Linux kernel to restrict file access, capabilities, and network access of a container.

- **Pros:**
  - Integrity of container images is preserved
  - Granular and stricter file access control
  - Can also restrict system calls and network activities
  - Easier to update and maintain (dynamic update)
- **Cons:**
  - Profiler needs to collect granular information.(e.g., read, write, execute, move, attribute change ...)
  - Difficult to create whitelist policies special file systems (e.g, /proc, /dev, /sys)
  - Runtime overhead



Mandatory Access Control (MAC) policy enforcement

Read permission

```
/usr/lib/x86_64-linux-gnu/libcrypto.so.1.0.0 rlk,
/lib/x86_64-linux-gnu/librt-2.19.so rlk,
/lib/x86_64-linux-gnu/libm-2.19.so rlk,
/lib/x86_64-linux-gnu/libgcc_s.so.1 rlk,
/sys/devices/system/cpu/online rlk,
/proc/{[1-9],[1-9][0-9]+}/auxv rlk,
/proc/version_signature rlk,
/proc/cpuinfo rlk,
/usr/lib/os-release rlk,
/proc/sys/kernel/osrelease rlk,
```

Execute permission

```
/data/db/ ix,
/usr/local/bin/gosu ix,
/usr/bin/numactl ix,
/sys/devices/system/node/ ix,
/sys/devices/system/cpu/ ix,
/bin/true ix,
/docker-entrypoint-initdb.d/ ix,
/bin/rm ix,
/usr/bin/mongod ix,
/data/db/journal/*/ ix,
```

Write permission

```
/data/db/journal/WiredTigerLog.0000000001 mwrk,
/data/db/journal/WiredTigerTmplog.0000000002 mwrk,
/data/db/WiredTigerLAS.wt mwrk,
/data/db/journal/WiredTigerTmplog.0000000003 mwrk,
/data/db/sizeStorer.wt mwrk,
/data/db/_mdb_catalog.wt mwrk,
/data/db/storage.bson.tmp mwrk,
```

Granted capabilities

```
capability dac_override,
capability setuid,
capability setgid,
capability net_bind_service,
```

Attack Surface Reduction: Demo

Attack Surface Reduction

Application Layer

Attack Surface Reduction

Application Layer

- Binaries are fat. Many unused functions make the attack surface larger

- Exploitable bugs in popular software still exist
  - Among the leading causes of system compromise

- Finding and fixing software vulnerabilities is not enough
  - Attackers may find them first

- Exploit mitigation technologies aim to make vulnerability exploitation harder
  - Not always the case: under certain conditions bypasses are possible
  - Still, the combined effect of multiple and diverse mitigation technologies makes exploitation harder

Attack Surface Reduction

Application Layer

- Reuse existing code to perform unintended actions
  - Initial instantiation: return-to-libc

- Return Oriented Programming (ROP):
  - Chain gadgets together to achieve arbitrary code execution

- Main mitigation techniques:
  - Make it harder for attackers to locate the code of interest
    - Software diversification (e.g., ASLR, code randomization)
  - Prevent control flow redirection to arbitrary locations
    - Control flow integrity (e.g., shadow stacks, Windows CFG)

Attack Surface Reduction

Application Layer

- **Code Debloating**: remove unused parts of code

- Exploits:
  - Use code/functionality not used by application ➔ **Attack Breaks!**
  - Use code/functionality used by application ➔ **Attack Succeeds!**

  - Granularity of Debloating:
    - Function level ➔ Code Stripping [Mulliner ’15]
    - API level ➔ This work

Attack Surface Reduction

API Specialization

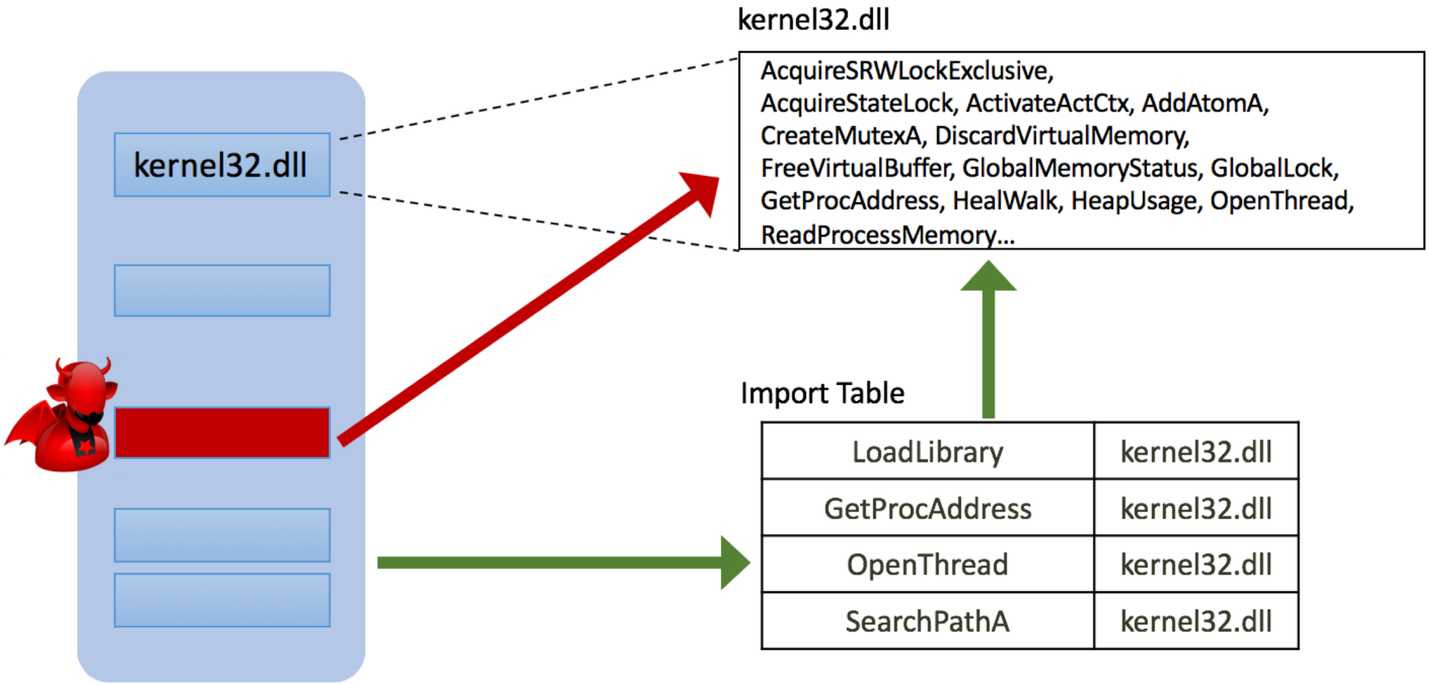
- Assumption: attacker has hijacked control flow
  – Non-randomized gadgets, JIT-ROP, full-function reuse, etc.

- Goal: break the exploit code by restricting its interaction with the OS
  – Restrict what/how system APIs are invoked

- Key insight: not all available API functions are used by most apps
  – From the functions used, only partial functionality is really needed

Attack Surface Reduction

API Specialization



Attackers have access to *All* available functions, although applications use *only few* of them.

Attack Surface Reduction

API Specialization

DLL	kernel32	advapi	shlwapi	user32	ole32
# of funcs	1941	902	931	1152	163
Adobe Reader	203	77	20	145	33
Notepad++	139	13	13	168	2
VLC	38	2	-	1	-
7Zip	93	19	-	84	11
Google Chrome	191	33	-	25	3

Attack Surface Reduction

API Specialization

- Create specialized versions of critical API functions for individual applications
  - Critical == 52 security-critical API functions
  - E.g., VirtualProtect(), VirtualAlloc(), connect()

- Goal: Neutralize dangerous argument values or combinations
  - E.g., changing memory permissions, connecting to servers

- Main intuition:

App's usage of critical API functions $\neq$ **Exploit code's usage of critical API functions**

Attack Surface Reduction

API Specialization

- Protects transparently application binaries
  – Does not require source code

- Best-effort approach!
  – It may not always break exploits
  – It may be easy to bypass

- Can be deployed along with other exploit mitigations

Attack Surface Reduction

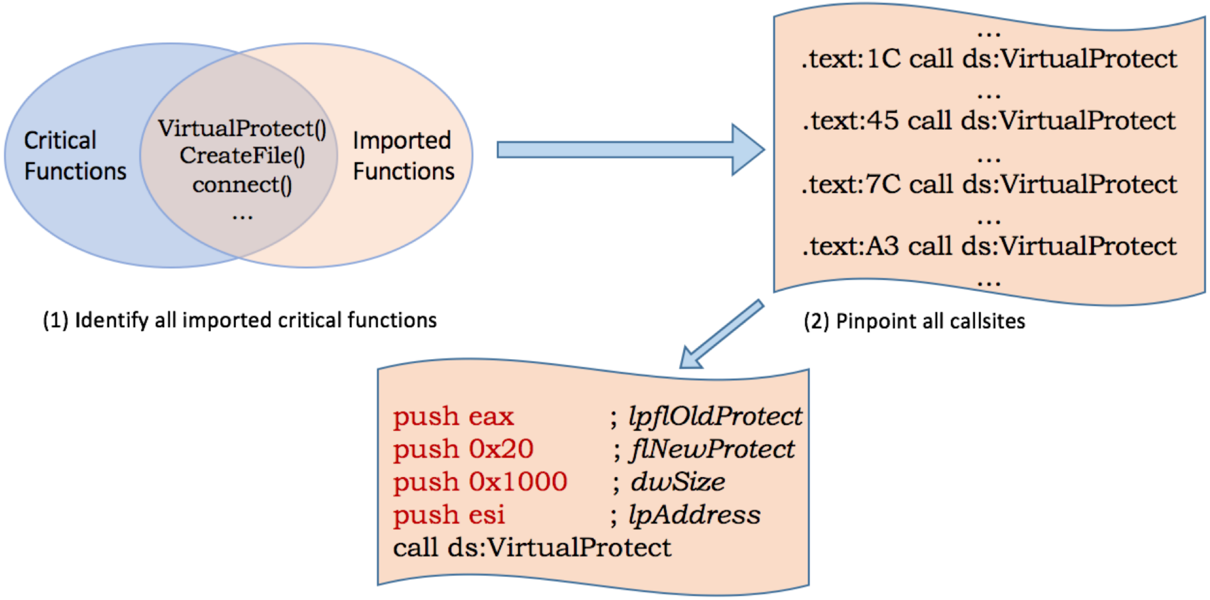
API Specialization: Approach

- Phase 1: **Offline pre-processing**
  - Disassemble binary
  - Extract CFG
  - Identify critical function call sites
  - Extract argument values and patterns *(backwards data-flow analysis)*
  - Generate process-wide per-function policies

- Phase 2: **Runtime enforcement**
  - Use library interposition to enforce extracted policies at runtime

Attack Surface Reduction

API Specialization: Approach



(1) Identify all imported critical functions

(2) Pinpoint all callsites

(3) Analyze arguments and patterns of each invocation

Attack Surface Reduction

API Specialization: Implementation

- Current prototype supports Win10 64-bit and Win7 32-bit
- Uses IDAPython scripting in IDA Pro 6.8 to perform inter-procedural backward-slicing
- Runtime enforcement is performed using the Microsoft Detours framework for library interposition

Attack Surface Reduction

API Specialization: Evaluation

- 251 Shellcode and 30 ROP Payloads samples
  – Collected from Metasploit, ExploitDB, and real-world/PoC exploits

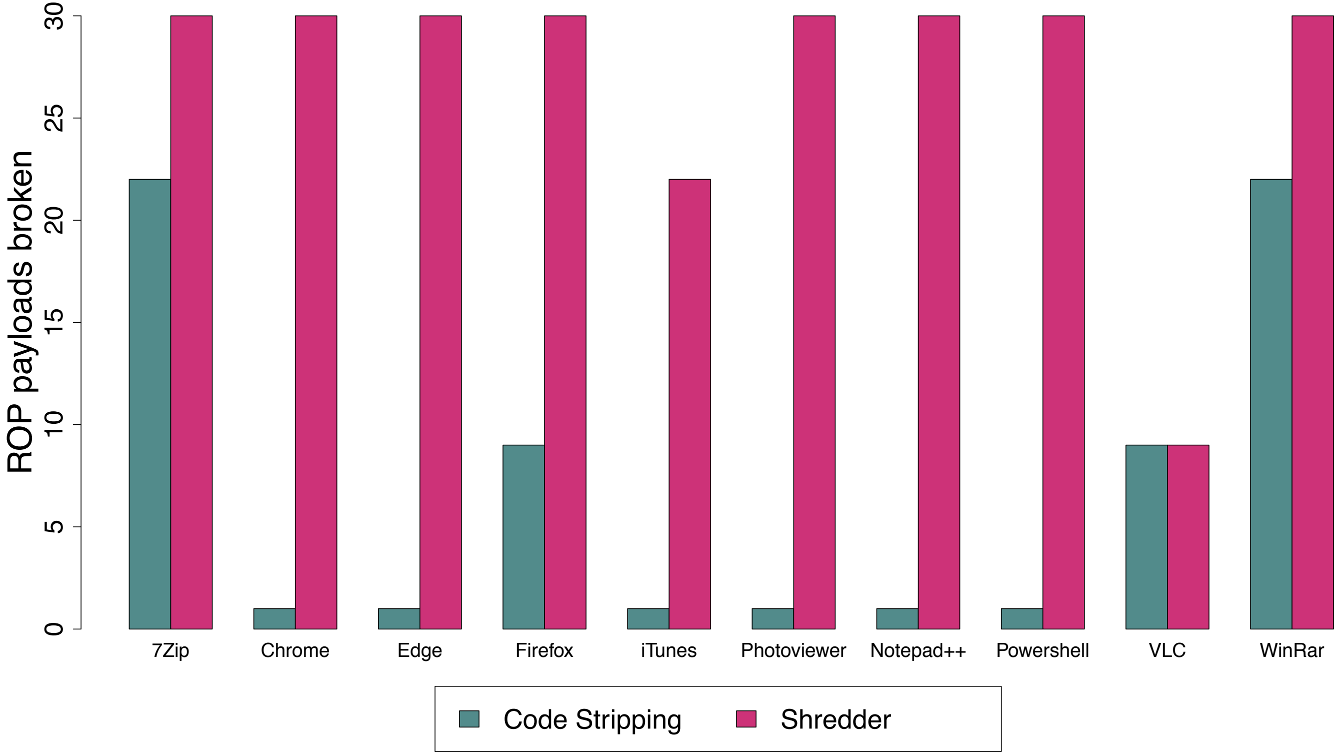
- Applications: 10 popular end-user programs
  – Web browsers, media players, text editors, etc.

- Main Result (compared to Code Stripping)
  – Breaks 18.3% more shellcodes
  – Breaks 298% more ROP payloads

- Negligible runtime overhead

Attack Surface Reduction

API Specialization



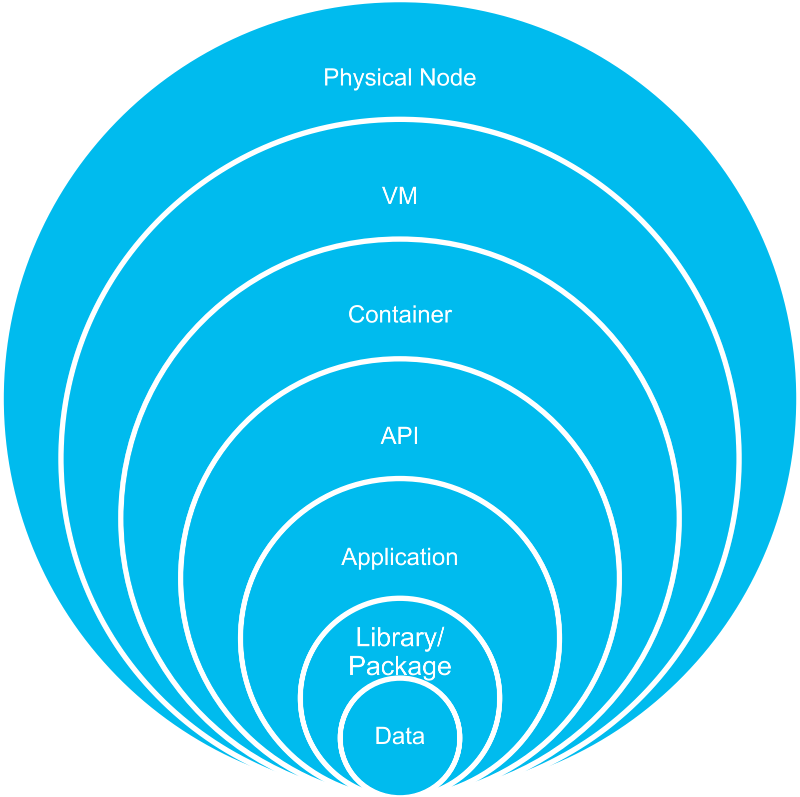
Attack Surface Reduction

API Specialization

- Shredder is a best effort attack surface reduction tool
  – Move beyond code debloating, to *functionality* debloating
- Relies on static analysis over application binaries to create policies which are enforced at runtime
- Policies restrict the application's usage of critical API functions
- Experimental evaluation across 10 popular user-apps
- Main Result (compared to Code Stripping)
  – Breaks 298% more ROP Payloads

Conclusion

- Defense in depth !
- Your Attack surface is too big, reduce it!
- Containers are still cool…ish



Questions

azzedine.benameur@accenture.com

lei.a.ding@accenture.com

jay.chen@accenture.com

mikepo@cs.stonybrook.edu