Software Compartmentalization and the Challenge of Interfaces

Pierre Olivier
pierre.olivier@manchester.ac.uk
www.pierreolivier.eu

Memory Safety

- Most systems software are today written in **memory-unsafe languages**
  - C/C++
  - This is for **performance** reasons

Memory Safety

- Most systems software are today written in **memory-unsafe languages**
  - C/C++
  - This is for **performance** reasons
- Programming mistakes introduce bugs leading to **memory corruption/undefined behavior**

Memory Safety

- Most systems software are today written in **memory-unsafe languages**
  - C/C++
  - This is for **performance** reasons
- Programming mistakes introduce bugs leading to **memory corruption/undefined behavior**
- Impact of such bugs in production goes much further than crashes: **security issues**
  - Bugs can be exploited by attackers to take over a system's execution flow, leak/tamper with critical data, escalate privileges, etc.

Memory Safety

Still the Main Security Issue in Systems Software

Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

By Catalin Cimpanu for Zero Day | February 11, 2019 | Topic: Security

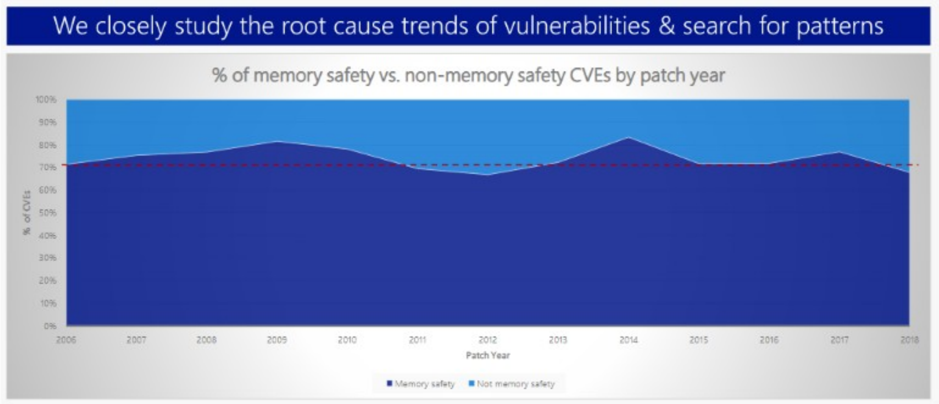


Image: Matt Miller

RELATED

Apple: Side-loading on iOS would open the malware floodgates

Security

Work to earn several highly respected CompTIA certifications with these self-paced courses

Security

US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft

Security

NEWSLETTERS

Memory Safety

Still the Main Security Issue in Systems Software

Microsoft: 70 percent … memory safety issues

Percentage of memory safety issues has been hovering

By Catalin Cimpanu for Zero Day |

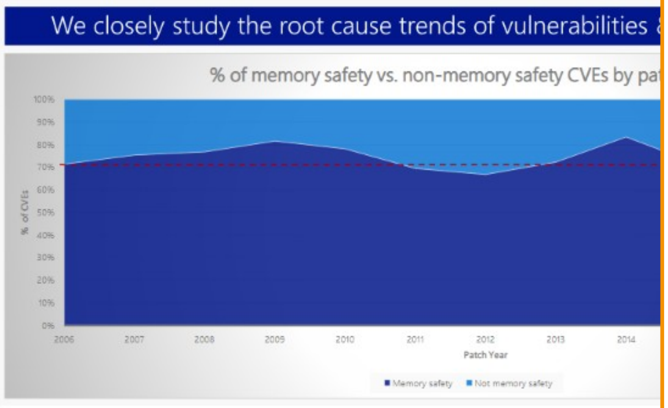


Image: Matt Miller

Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

By Catalin Cimpanu for Zero Day | May 23, 2020 | Topic: Security

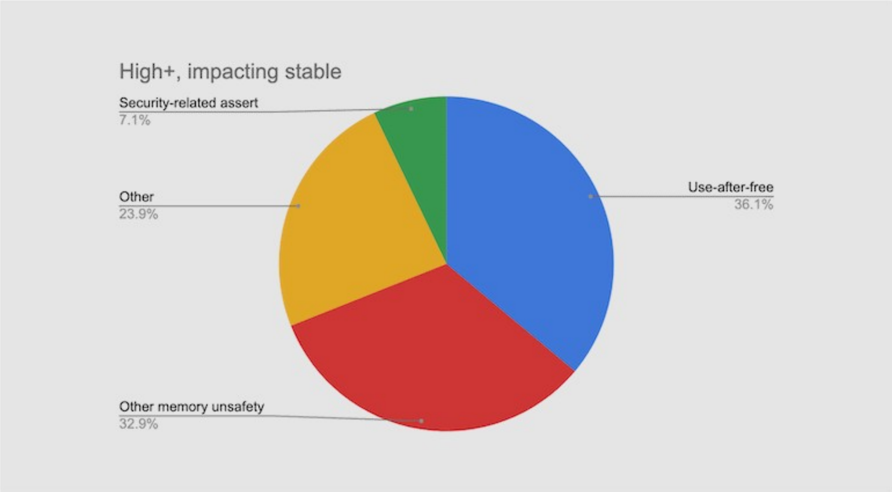


Image: Google

RELATED

Apple: Side-loading on iOS would open the malware floodgates

Security

Work to earn several highly respected CompTIA certifications with these self-paced courses

Security

US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft

Security

NEWSLETTERS

ZDNet Security

Your weekly update on security around the globe, featuring research, threats, and more.

Memory Safety

Still the Main Security Issue in Systems Software

Microsoft: 70 percent
memory safety issues

Percentage of memory safety issues has been hovering

Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

By Catalin Cimpanu for Zero Day | May 23, 2020 | Topic: Security

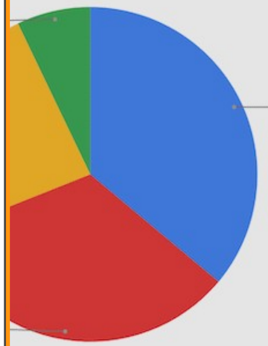


Image: Google

RELATED

Apple: Side-loading on iOS would open the malware floodgates

Security

Work to earn several highly respected CompTIA certifications with these self-paced courses

Security

US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft

Security

NEWSLETTERS

ZDNet Security

Your weekly update on security around the globe, featuring research, threats, and more.

National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also found a similar percentage of memory safety vulnerabilities over several years in Chrome®. [2] Malicious cyber actors can exploit these vulnerabilities for remote code execution or other adverse effects, which can often compromise a device and be the first step in large-scale network intrusions.

Commonly used languages, such as C and C++, provide a lot of freedom and flexibility

Memory Safety

Possible Solutions

- **Existing solutions are not perfect**
  - Using **memory safe languages** is either too slow or too restrictive for the programmer

Memory Safety

Possible Solutions

- **Existing solutions are not perfect**
  - Using **memory safe languages** is either too slow or too restrictive for the programmer
  - Formal **verification techniques** are either too restrictive or do not scale to the large code bases of modern systems software

Memory Safety

Possible Solutions

- **Existing solutions are not perfect**
  - Using **memory safe languages** is either too slow or too restrictive for the programmer
  - Formal **verification techniques** are either too restrictive or do not scale to the large code bases of modern systems software
  - **C/C++ hardening techniques** are not comprehensive/can be bypassed/have an unacceptable performance impact
  - etc.

Memory Safety

Possible Solutions

- **Existing solutions are not perfect**
  - Using **memory safe languages** is either too slow or too restrictive for the programmer
  - Formal **verification techniques** are either too restrictive or do not scale to the large code bases of modern systems software
  - **C/C++ hardening techniques** are not comprehensive/can be bypassed/have an unacceptable performance impact
  - etc.
- **So C/C++ remain popular, and memory corruption vulnerabilities are not going away anytime soon**

Software Compartmentalization

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**

Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

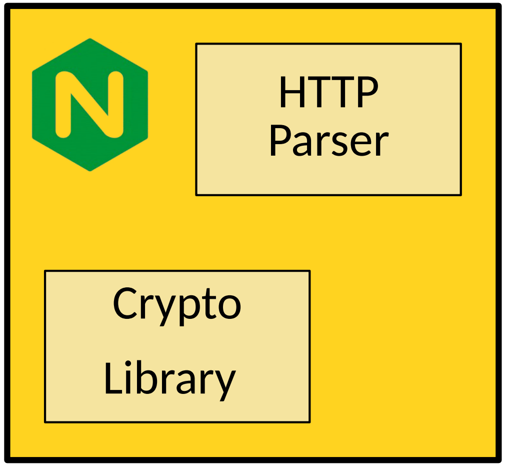
- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact

Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact



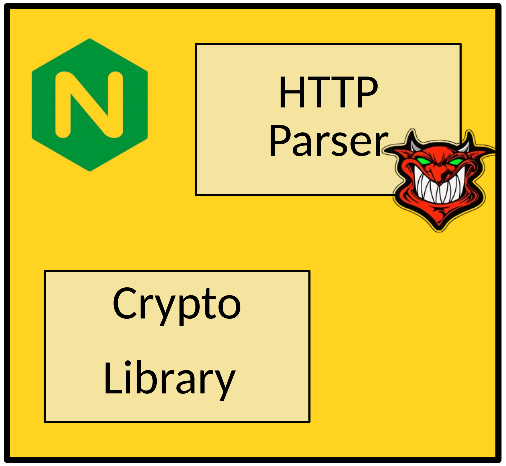
Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.

Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact

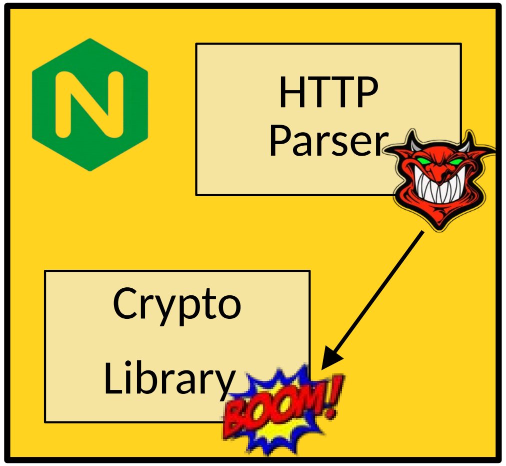


Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact

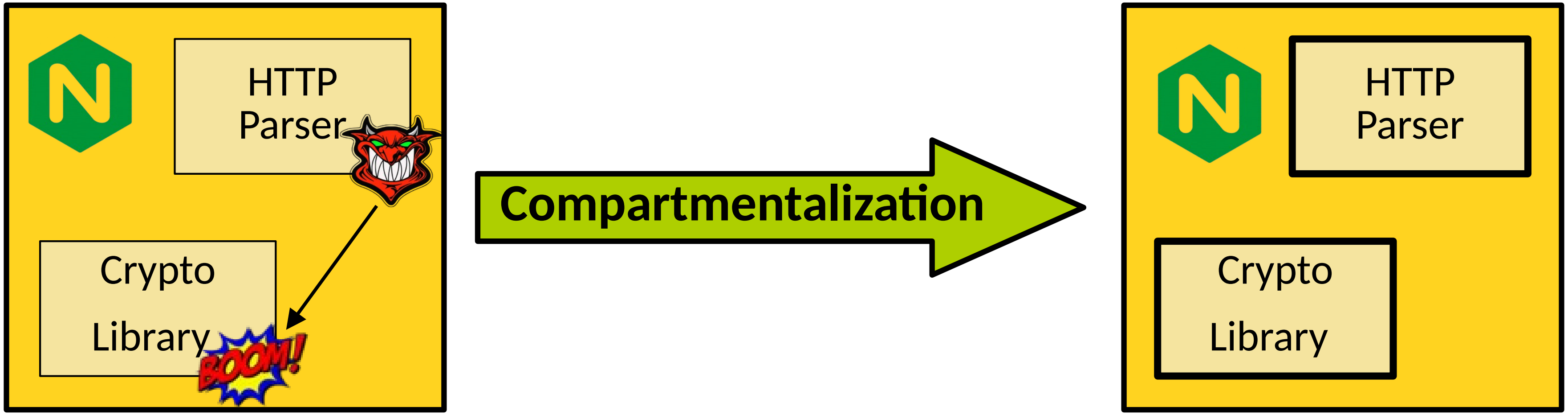


Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact



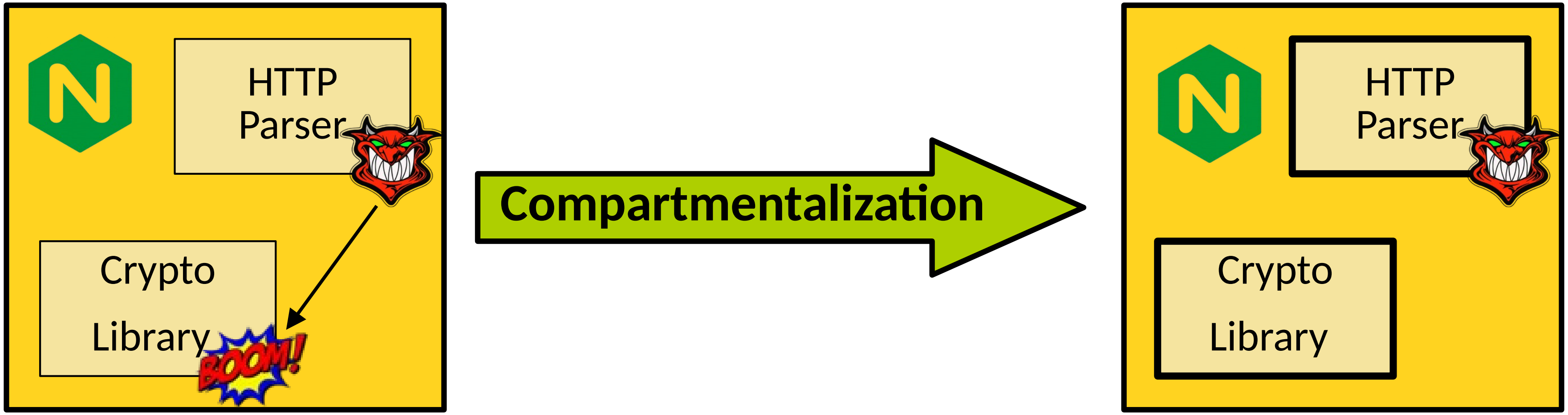
Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.

Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact

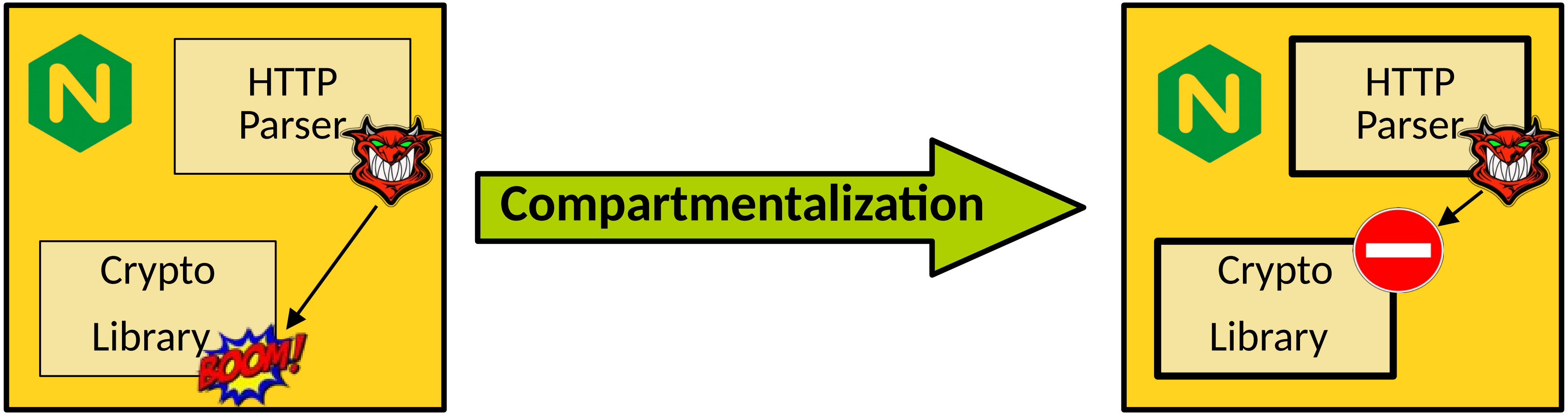


Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Motivation & Presentation

- Software compartmentalization **decompose software into lesser-privileged components that only have access to what they need to do their job**
  - Different from previous approaches: acknowledge there will be bugs and exploits, try to limit their impact



Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track, pp. 273-284. 2003.
Brumley, David, and Dawn Song. "Privtrans: Automatically partitioning programs for privilege separation." In USENIX Security Symposium, vol. 57, no. 72. 2004.

Software Compartmentalization

Scope

Software Compartmentalization

Scope

- Compartmentalization is not complete isolation: components are still part of a single application/system and **communicate**

Software Compartmentalization

Scope

- Compartmentalization is not complete isolation: components are still part of a single application/system and **communicate**
- Traditional examples: OS kernels, web browser, web servers, SSH software

Software Compartmentalization

Scope

- Compartmentalization is not complete isolation: components are still part of a single application/system and **communicate**
- Traditional examples: OS kernels, web browser, web servers, SSH software
- We'll also focus on compartmentalization **applied to existing software**

Software Compartmentalization

Scope

- Compartmentalization is not complete isolation: components are still part of a single application/system and **communicate**
- Traditional examples: OS kernels, web browser, web servers, SSH software
- We'll also focus on compartmentalization **applied to existing software**
  - Notice the disconnection with the traditional examples that were **built from scratch with (some degree of) compartmentalization in mind**
  - The idea is that we have a gigantic **existing legacy codebase of unsafe systems software that would benefit from that approach**

Software Compartmentalization

Scope

	Compartment 1 (crypto library)	Compartment 2 (HTTP parser)
Crypto keys	Read access	No access
HTTP request data	No access	Read access

Lampson's
access control matrix

Lampson, Butler W. (1971). "Protection". Proceedings of the 5th Princeton Conference on Information Sciences and Systems. p. 437.

Software Compartmentalization

Scope

- Key idea: restrict control and data flow in the application so that each compartment has the permissions it requires to do its job
- It's an application of the **principle of least privilege**

	Compartment 1 (crypto library)	Compartment 2 (HTTP parser)
Crypto keys	Read access	No access
HTTP request data	No access	Read access

Lampson's
access control matrix

Lampson, Butler W. (1971). "Protection". Proceedings of the 5th Princeton Conference on Information Sciences and Systems. p. 437.

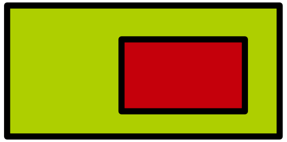
Software Compartmentalization
Trust Models

- **3 trust models:**

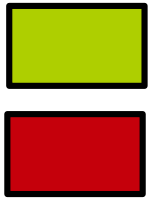
Software Compartmentalization

Trust Models

- **3 trust models:**
  - **Sandbox:** part of the program is untrusted, isolated the (trusted) rest of the program from it



Sandbox



Trusted

Untrusted

Software Compartmentalization

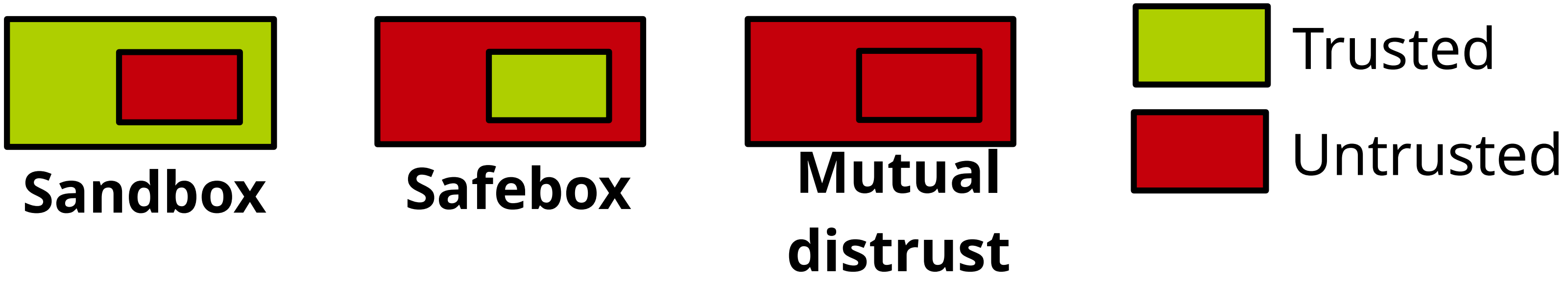
Trust Models

- **3 trust models:**
  - **Sandbox:** part of the program is untrusted, isolated the (trusted) rest of the program from it
  - **Safebox:** part of the program is security critical, isolated it from the (untrusted) rest of the program

Software Compartmentalization

Trust Models

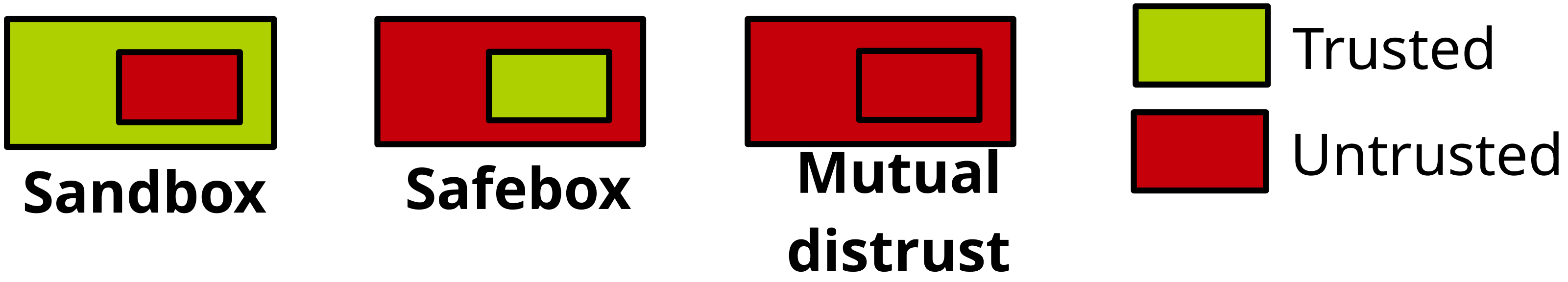
- **3 trust models:**
  - **Sandbox:** part of the program is untrusted, isolated the (trusted) rest of the program from it
  - **Safebox:** part of the program is security critical, isolated it from the (untrusted) rest of the program
  - **Mutual distrust:** compartments distrust each others
    - Stronger generalization of the other 2 models



Software Compartmentalization

Trust Models

- **3 trust models:**
  - **Sandbox:** part of the program is untrusted, isolated the (trusted) rest of the program from it
  - **Safebox:** part of the program is security critical, isolated it from the (untrusted) rest of the program
  - **Mutual distrust:** compartments distrust each others
    - Stronger generalization of the other 2 models
  - All generalise to more than 2 compartments



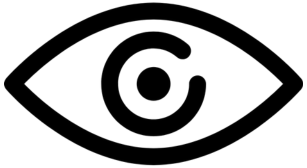
Software Compartmentalization

Security Properties

- **3 security properties considered:**

Software Compartmentalization

Security Properties

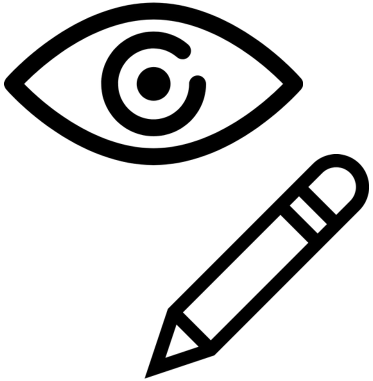


- **3 security properties considered:**
  - **Confidentiality:** an attacker cannot read/leak information from outside of a subverted compartment

Software Compartmentalization

Security Properties

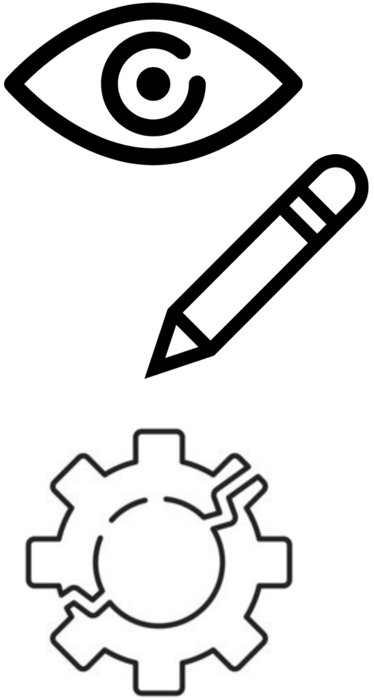
- **3 security properties considered:**
  - **Confidentiality:** an attacker cannot read/leak information from outside of a subverted compartment
  - **Integrity:** an attacker cannot write/tamper with data outside of a subverted compartment



Software Compartmentalization

Security Properties

- **3 security properties considered:**
  - **Confidentiality:** an attacker cannot read/leak information from outside of a subverted compartment
  - **Integrity:** an attacker cannot write/tamper with data outside of a subverted compartment
  - **Availability:** an attacker cannot disrupt (e.g. crash) code running outside of a subverted compartment
    - Very hard to achieve without complete redesing of monolithic application, out of scope for most existing efforts



Software Compartmentalization

Illustrative Example

```
int global;

int library_function(int *parameter) {
      char *cryptokey = “private“;

      int ret = *parameter + global + 42;
      return ret;
}

int main() {
      int param = 100;
      global = 50;
      char *password = “secret“;

      /* … */

      int res = library_function(&param);

      /* … */

      return 0;
}
```

Software Compartmentalization

Illustrative Example

```
int global;

int library_function(int *parameter) {
      char *cryptokey = “private“;

      int ret = *parameter + global + 42;
      return ret;
}

int main() {
      int param = 100;
      global = 50;
      char *password = “secret“;

      /* … */

      int res = library_function(&param);

      /* … */

      return 0;
}
```

Policy: put `library_function` in a compartment, and `main` in another

Software Compartmentalization

Illustrative Example

```
int global;

int library_function(int *parameter) {
      char *cryptokey = “private“;

      int ret = *parameter + global + 42;
      return ret;
}

int main() {
      int param = 100;
      global = 50;
      char *password = “secret“;

      /* … */

      int res = GATE(library_function, &param);

      /* … */

      return 0;
}
```

Policy: put `library_function` in a compartment, and `main` in another

Compartmentalization:

- Add a gate performing security domain switch (e.g. page table, MPK, etc.)
- Identify shared data and allocate it somewhere accessible from both compartments

Software Compartmentalization

Illustrative Example

```
int global;

int library_function(int *parameter) {
      char *cryptokey = "private";

      int ret = *parameter + global + 42;
      return ret;
}

int main() {
      int param = 100;
      global = 50;
      char *password = "secret";

      /* … */

      int res = GATE(library_function, &param);

      /* … */

      return 0;
}
```

Policy: put `library_function` in a compartment, and `main` in another

Compartmentalization:

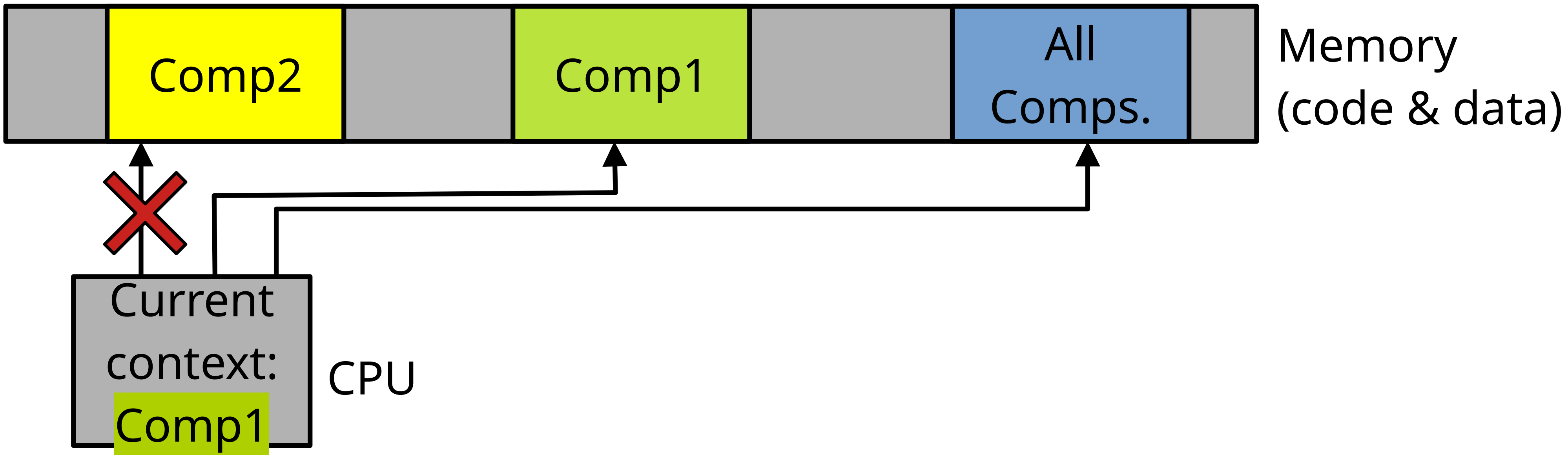
- Add a gate performing security domain switch (e.g. page table, MPK, etc.)
- Identify shared data and allocate it somewhere accessible from both compartments

Many modern frameworks can help:
codejail, ERIM, Hodor, Donky, Ptrsplit, memsentry, libmpk, Cali, CubicleOS, LibHermitMPK, FlexOS, Polytope, etc.

Software Compartmentalization

Illustrative Example

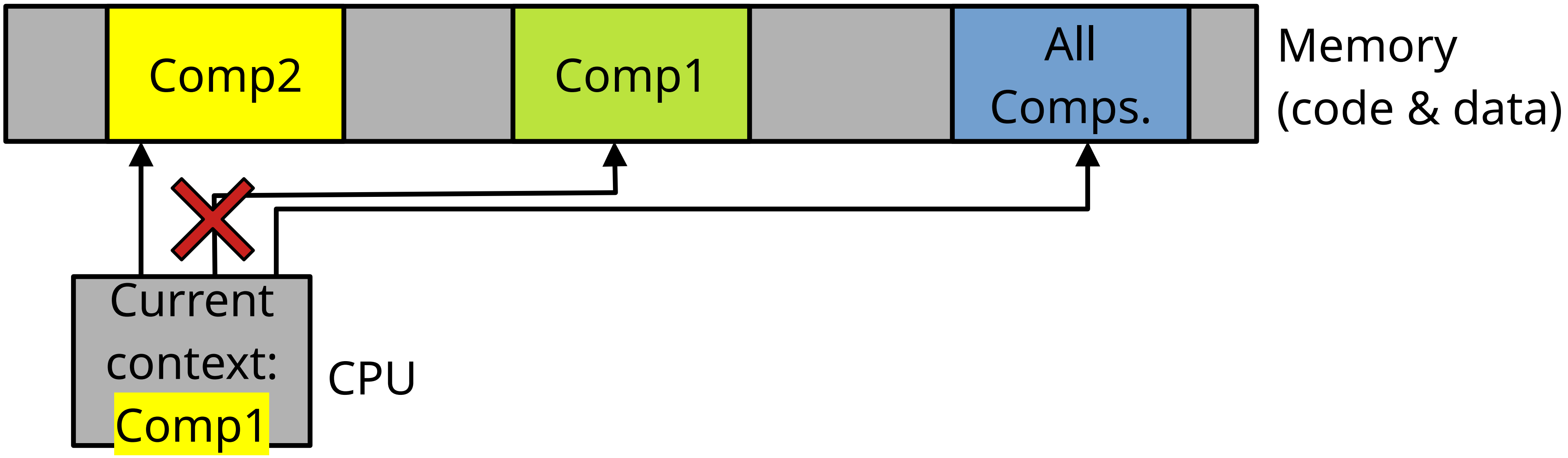
- How does it work from the hardware point of view?



Software Compartmentalization

Illustrative Example

- How does it work from the hardware point of view?



Security domain switch comp2 → comp1

Software Compartmentalization

It's in the Air!

UK Research and Innovation

Apply for funding | Manage your award | What we offer | News and ever

Browse our areas of investment and support | Our main funds and t
Career development | Supporting collaboration | Supporting inno
Artificial intelligence | Research financial sustainability | Public e

Home > What we offer > Browse our areas of investment and support > Digital security by design

Area of investment and support

Digital security by design

The digital security by design (DSbD) challenge funds business and researchers to update the foundation of the insecure digital computing infrastructure by creating a new, more secure hardware and software ecosystem.

Budget: £70 million

Duration: From 2020 to 2025

Partners involved: Innovate UK, Engineering and Physical Sciences Research Council (EPSRC)

Open all

DARPA

Broad Agency Announcement

Compartmentalization and Privilege Management (CPM)

INFORMATION INNOVATION OFFICE

HR001123S0028

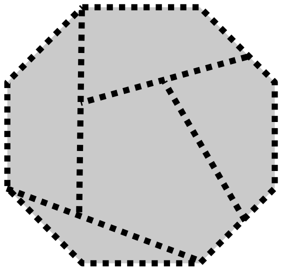
April 4, 2023

How to Compartmentalize an App

- **Compartmentalizing an existing monolithic application can be abstracted into the following 3 sequential steps:**

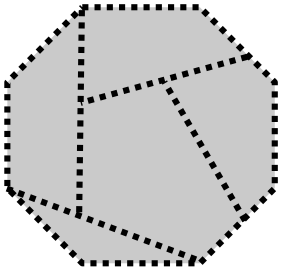
How to Compartmentalize an App

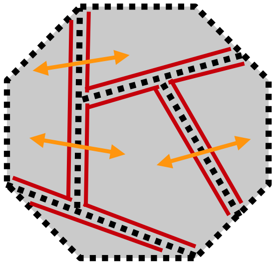
- **Compartmentalizing an existing monolithic application can be abstracted into the following 3 sequential steps:**
  1) ***Policy* definition**: how many compartments, what part of the app goes into what compartment



How to Compartmentalize an App

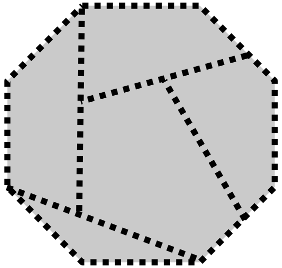
- **Compartmentalizing an existing monolithic application can be abstracted into the following 3 sequential steps:**
  1) ***Policy* definition**: how many compartments, what part of the app goes into what compartment
  2) ***Shared/private data* classification, *interface* sanitization**: because the app was not designed with compartmentalization in mind

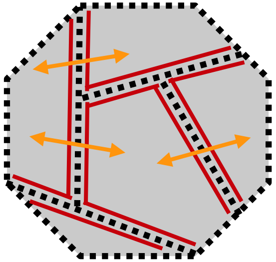


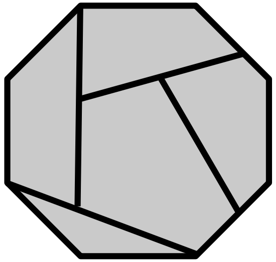


How to Compartmentalize an App

- **Compartmentalizing an existing monolithic application can be abstracted into the following 3 sequential steps:**
  1) ***Policy* definition**: how many compartments, what part of the app goes into what compartment
  2) ***Shared/private data* classification, *interface* sanitization**: because the app was not designed with compartmentalization in mind
  3) **Integrate an *isolation mechanism* and a compatible *data sharing strategy*** to enforce compartmentalization at runtime



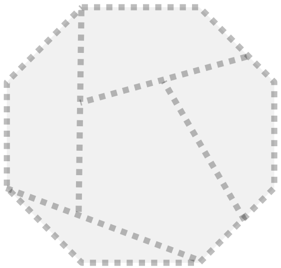
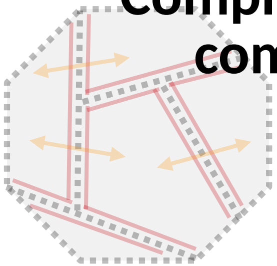




How to Compartmentalize an App

- **Compartmentalizing an existing monolithic application can be abstracted into the following 3 sequential steps:**
  1) ***Policy* definition**: how many compartments, what part of the app goes into what compartment
  2) ***Shared/private data* classification, *interface* sanitization**: because the app was not designed with compartmentalization in mind
  3) **Integrate an *isolation mechanism* and a compatible *data sharing strategy*** to enforce compartmentalization at runtime

Completely neglected in most major compartmentalization efforts!


Interface Security

Interface Security

Motivation

```
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
      data[index] = object;

      /* … */
}

int main() {
      int index = get_index();
      double object = get_index();

      if (index < DATA_SIZE)
            library_function(index, object);

      /* … */
}
```

Policy: put `library_function` in a compartment, and `main` in another

Interface Security

Motivation

```c
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
	data[index] = object;

	/* … */
}

int main() {
	int index = get_index();
	double object = get_index();

	if (index < DATA_SIZE)
		library_function(index, object);

	/* … */
}
```

Policy: put `library_function` in a compartment, and `main` in another

Compartmentalization: put a gate at the level of the call to `library_function`
There is no shared data (data only accessed from 1 compartment, parameters passed by copy)

Interface Security

Motivation

```
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
      data[index] = object;

      /* … */
}

int main() {
      int index = get_index();
      double object = get_index();

      if (index < DATA_SIZE)
            library_function(index, object);

      /* … */
}
```

Isolate `library_function` and `main` in different compartment creates a new internal trust boundary: the interface between the two compartments: here it is the call to `library_function`

Interface Security

Motivation

```
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
      data[index] = object;

      /* … */
}

int main() {
      int index = get_index();
      double object = get_index();

      if (index < DATA_SIZE)
            library_function(index, object);

      /* … */
}
```

Isolate `library_function` and `main` in different compartment creates a new internal trust boundary: the interface between the two compartments: here it is the call to `library_function`

Assume `main` is malicious and send corrupted values through this interface

Interface Security

Motivation

```c
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
	data[index] = object;

	/* … */
}

int main() {
	int index = get_index();
	double object = get_index();

	if (index < DATA_SIZE)
		library_function(index, object);

	/* … */
}
```

Isolate `library_function` and `main` in different compartment creates a new internal trust boundary: the interface between the two compartments: here it is the call to `library_function`

Assume `main` is malicious and send corrupted values through this interface

The lack of check in `library_function` gives an untrusted caller (e.g. `main`) an arbitrary memory write primitive

Interface Security

Motivation

```
double data[DATA_SIZE];

/* … */

int library_function(int index, double object) {
      if (index >= DATA_SIZE || index < 0)
            return -1;
      data[index] = object;

      /* … */
}

int main() {
      int index = get_index();
      double object = get_index();

      if (index < DATA_SIZE)
            library_function(index, object);

      /* … */
}
```

Fix: have a check within the trusted compartment

Compartment Interface Vulnerabilities (CIVs)

Definition

- **CIVs = Vulnerabilities arising due to lack of or improper Control and Data flow validation at compartment boundaries**
- **Classes of CIVs:**

Data Leakages	*Data Corruption*	*Temporal Violations*

Compartment Interface Vulnerabilities (CIVs)

Definition

- **CIVs = Vulnerabilities arising due to lack of or improper Control and Data flow validation at compartment boundaries**
- **Classes of CIVs:**

Data Leakages	*Data Corruption*	*Temporal Violations*
• Exposure of addresses • Exposure of compartment-confidential data		

Compartment Interface Vulnerabilities (CIVs)

Definition

- **CIVs = Vulnerabilities arising due to lack of or improper Control and Data flow validation at compartment boundaries**
- **Classes of CIVs:**

Data Leakages	***Data Corruption***	***Temporal Violations***
• Exposure of addresses • Exposure of compartment-confidential data	• Dereference of corrupted pointer • Usage of corrupted indexing information • Usage of corrupted object	

Compartment Interface Vulnerabilities (CIVs)

Definition

- **CIVs = Vulnerabilities arising due to lack of or improper Control and Data flow validation at compartment boundaries**
- **Classes of CIVs:**

Data Leakages	*Data Corruption*	*Temporal Violations*
• Exposure of addresses • Exposure of compartment-confidential data	• Dereference of corrupted pointer • Usage of corrupted indexing information • Usage of corrupted object	• Expectation of API usage ordering • Usage of corrupted synchronization primitive • Shared memory TOCTOU

Interface Security

Problem Statement

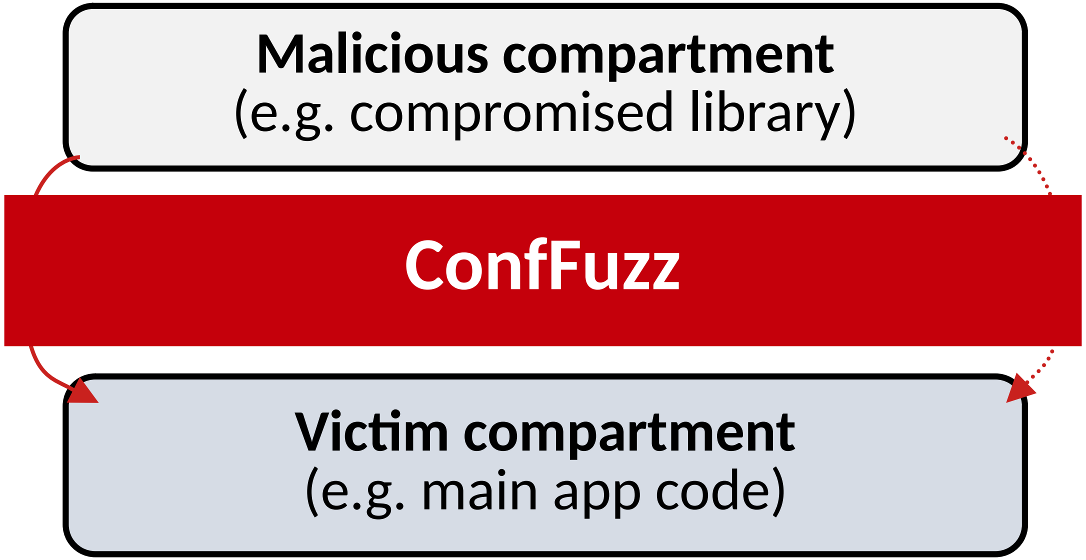
- **The vast majority of modern compartmentalization framework ignore the problem of interface safety!**

Interface Security

Problem Statement

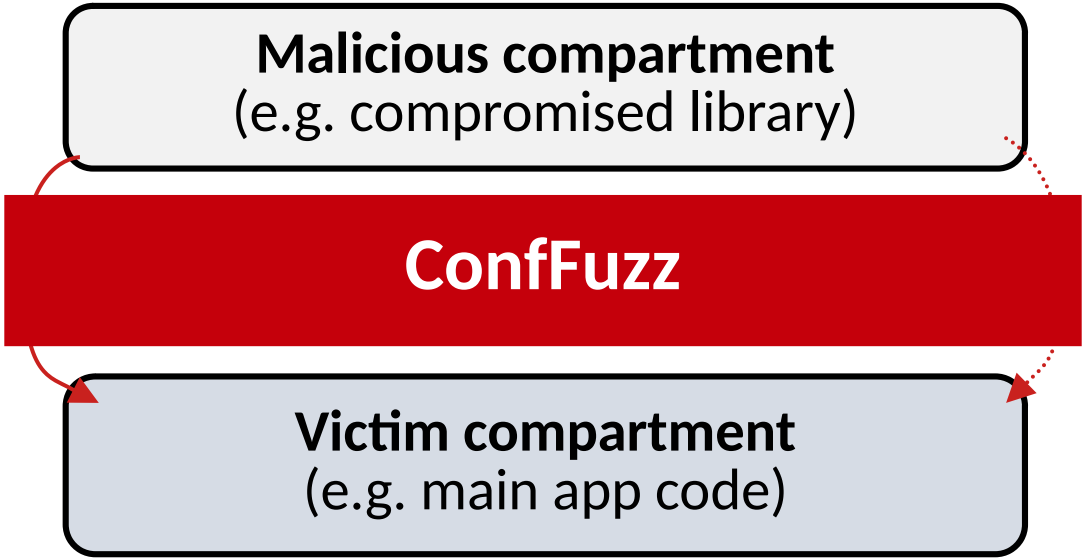
- **The vast majority of modern compartmentalization framework ignore the problem of interface safety!**
- Is there any point in compartmentalising our applications with these frameworks without considering interfaces? **How bad is the problem of CIVs?**
  - *How many CIVs are there at legacy, unported APIs?*
  - *Are all APIs similarly affected by CIVs? (e.g., library v.s. module APIs)*
  - *How hard are these CIVs to address when compartmentalizing?*
  - *How bad are they? i.e., if you don't fix them, what can attackers do?*

ConfFuzz: Fuzzing for CIVs



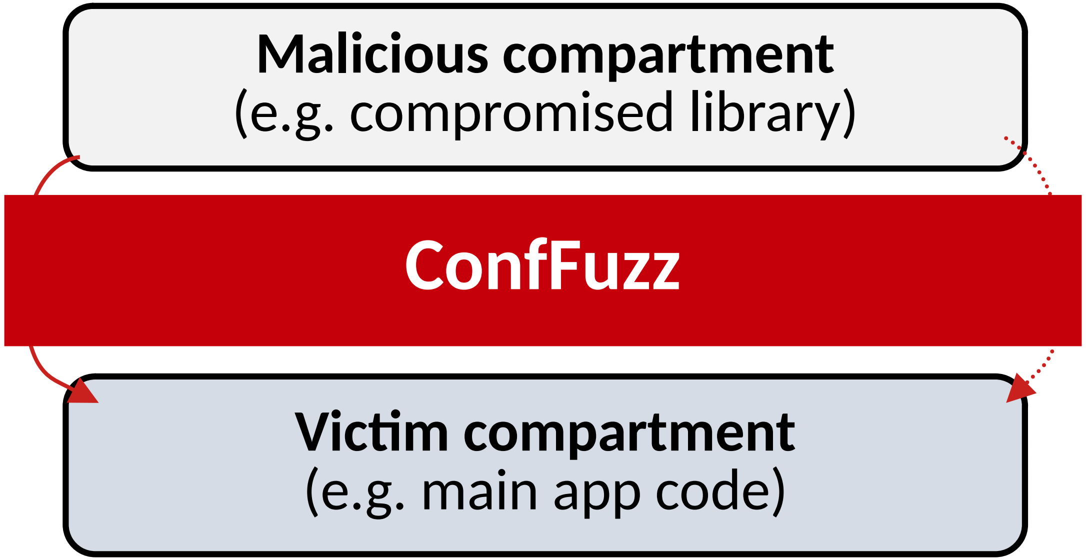
- We built a **fuzzer injecting malformed data at possible compartment interfaces**
  - E.g. library/main app. Code

ConfFuzz: Fuzzing for CIVs



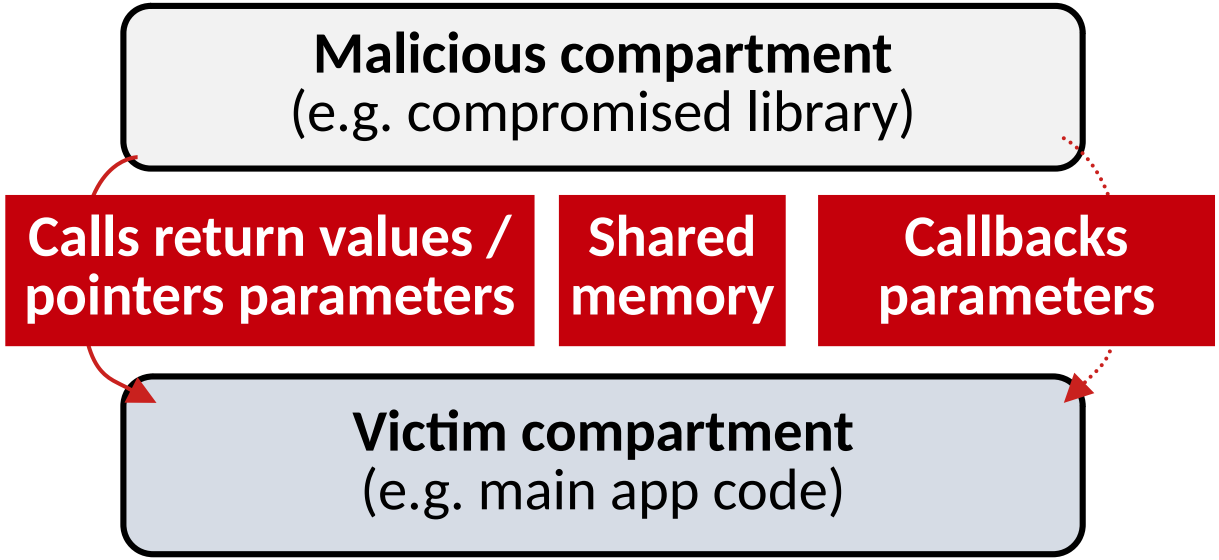
- We built a **fuzzer injecting malformed data at possible compartment interfaces**
  - E.g. library/main app. Code
- It runs on monolithic (non-compartmentalized) software to uncover a maximum of CIVs

ConfFuzz: Fuzzing for CIVs



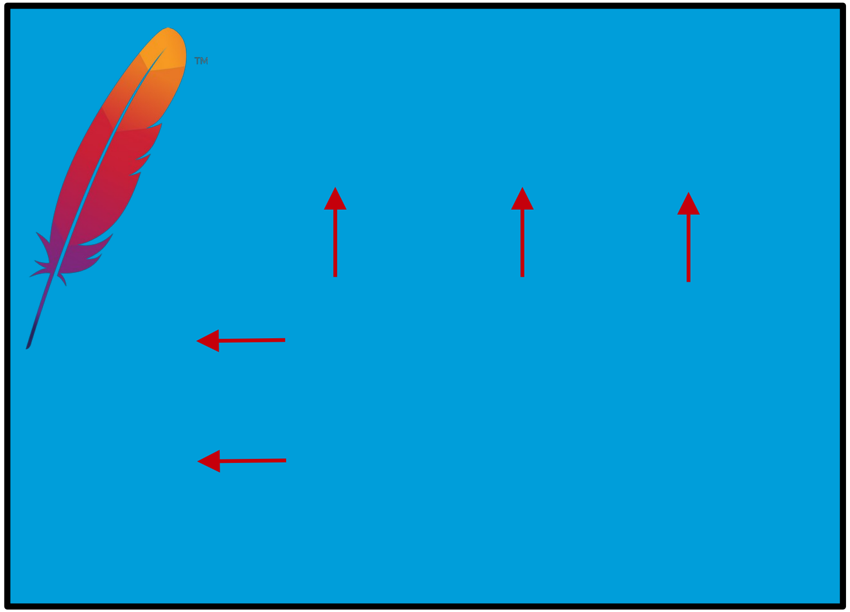
- We built a **fuzzer injecting malformed data at possible compartment interfaces**
  - E.g. library/main app. Code
- It runs on monolithic (non-compartmentalized) software to uncover a maximum of CIVs
- We apply it to many compartmentalization scenarios and **study the bugs we uncover**

ConfFuzz: Fuzzing for CIVs



- ConfFuzz covers the entire attack surface of a victim compartment
- Can fuzz both ways:
  - **SandBox**: malicious compartment calls the victim
  - **SafeBox**: the victim calls the malicious compartment

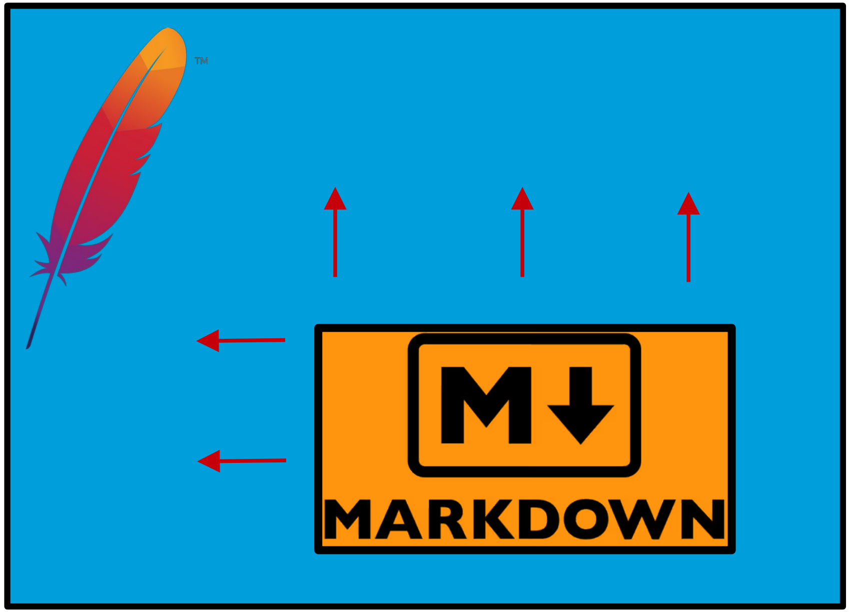
ConfFuzz: Fuzzing for CIVs



Methodology:

1) Choose an app

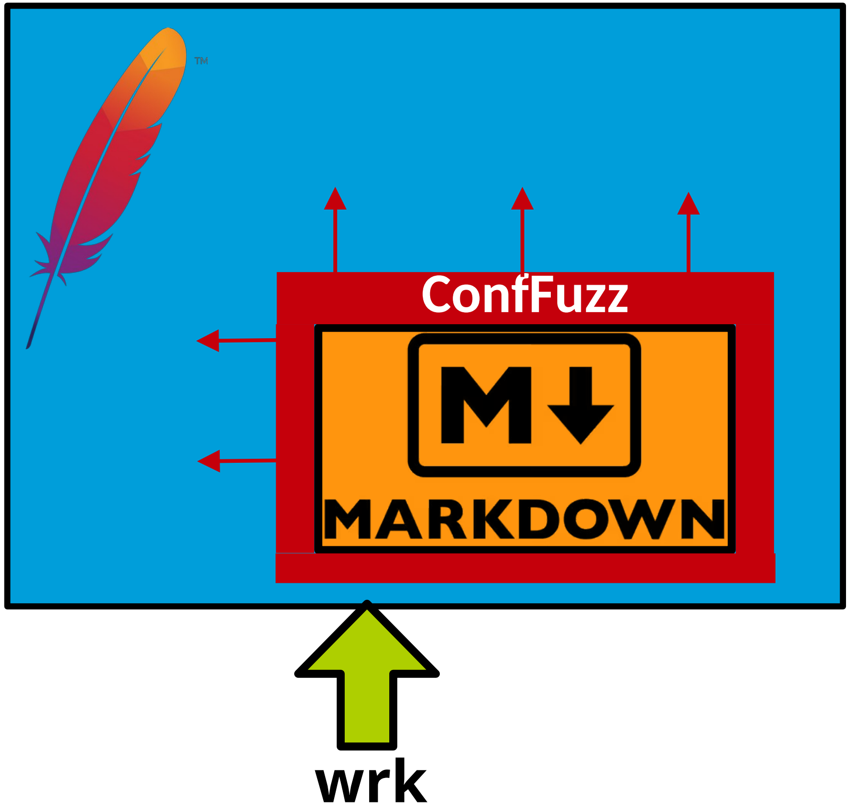
ConfFuzz: Fuzzing for CIVs



Methodology:

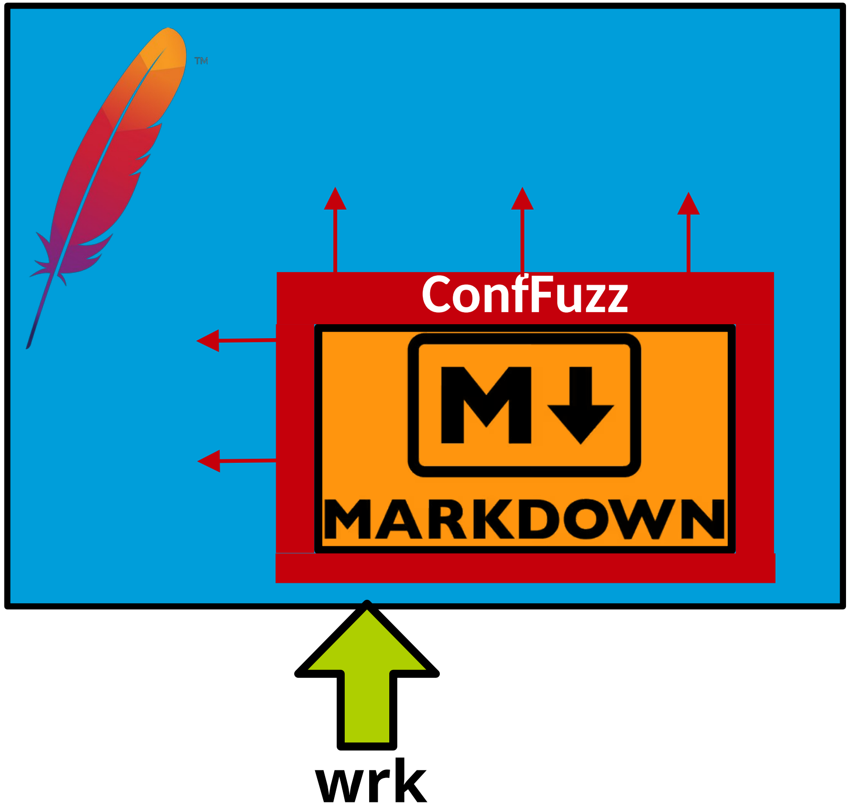
1) Choose an app
2) Choose a meaningful potential compartmentalization interface

ConfFuzz: Fuzzing for CIVs



Methodology:
1) Choose an app
2) Choose a meaningful potential compartmentalization interface
3) Run the application with Asan, hook ConfFuzz to the interface
4) Stress the app, fuzz and gather as many bugs as we can

ConfFuzz: Fuzzing for CIVs



Methodology:
1) Choose an app
2) Choose a meaningful potential compartmentalization interface
3) Run the application with Asan, hook ConfFuzz to the interface
4) Stress the app, fuzz and gather as many bugs as we can
5) Rince and repeat

TM	Application	Compartment API	References	Crashes		Victims	API Coverage		Impact (of which arbitrary)				
				Raw	*Dedup.*		*Callers*	*Coverage*	*Read*	*Write*	*Exec*	*Alloc*	*Null*
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	2
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

25 applications

36 APIs in total

TM	Application	Compartment API	References	Crashes		Victims	API Coverage		Impact (of which arbitrary)				
				Raw	*Dedup.*		*Callers*	*Coverage*	*Read*	*Write*	*Exec*	*Alloc*	*Null*
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	2
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

Library APIs

Module APIs

Internal APIs

25 applications

36 APIs in total

TM	Application	Compartment API	References	Crashes		Victims	API Coverage		Impact (of which arbitrary)				
				Raw	*Dedup.*		*Callers*	*Coverage*	*Read*	*Write*	*Exec*	*Alloc*	*Null*
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	2
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

Library APIs | Module APIs | Internal APIs

25 applications

36 APIs in total

16 of which taken from the literature

TM	Application	Compartment API	References	Crashes Raw	Crashes Dedup.	Victims	API Coverage Callers	API Coverage Coverage	Impact (of which arbitrary) Read	Write	Exec	Alloc	Null
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	2
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

Library APIs

Module APIs

Internal APIs

25 applications

36 APIs in total

16 of which taken from the literature

Found 629 unique CIVs

TM	Application	Compartment API	References	Crashes Raw	Crashes Dedup.	Victims	API Coverage Callers	API Coverage Coverage	Impact (of which arbitrary) Read	Write	Exec	Alloc	Null
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	[illegible]
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

Library APIs | Module APIs | Internal APIs

5 security impact types

25 applications

36 APIs in total

16 of which taken from the literature

Found 629 unique CIVs

TM	Application	Compartment API	References	Crashes		Victims	API Coverage		Impact (of which arbitrary)				
				Raw	Dedup.		Callers	Coverage	Read	Write	Exec	Alloc	Null
Sandbox	HTTPd	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
		mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
	aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
	cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
	exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
	FFmpeg	libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
		libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
		libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	lighttpd	mod_deflate		117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
	Image Magick	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
		libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
		mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
	Okular	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
		libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
	Redis	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
		mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
	rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
	Wireshark	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
		libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
	Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
Safebox	cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Nginx	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
		libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
	sudo	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
		libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	3
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	104

```c
// CIV 1: option setting API leads to arbitrary R/W
ulong SSL_CTX_set_options(SSL_CTX *ctx, ulong op) {
  return ctx->options |= op;
}

// CIV 2: cross-API object SSL_CTX with function
// pointers leads to arbitrary execution
SSL *SSL_new(SSL_CTX *ctx) {
  /* ... */
  s->method = ctx->method;
  /* ... */
  if (!s->method->ssl_new(s)) // arbitrary execution
    goto err;
} /* ... */
```

Safeboxing libssl:

2 CIVs leading to arbitrary read, write, and execute impact. Both functions are exposed to the application.

```c
int sudo_passwd_verify(struct passwd *pw, char *pass,
  sudo_auth *auth, struct sudo_conv_callback *cb) {
  /* ... abbreviated ... */
  sav = pass[8]; // read CIV
  pass[8] = '\0'; // write CIV
} /* ... abbreviated ... */
```

Safeboxing sudo's authentication API:

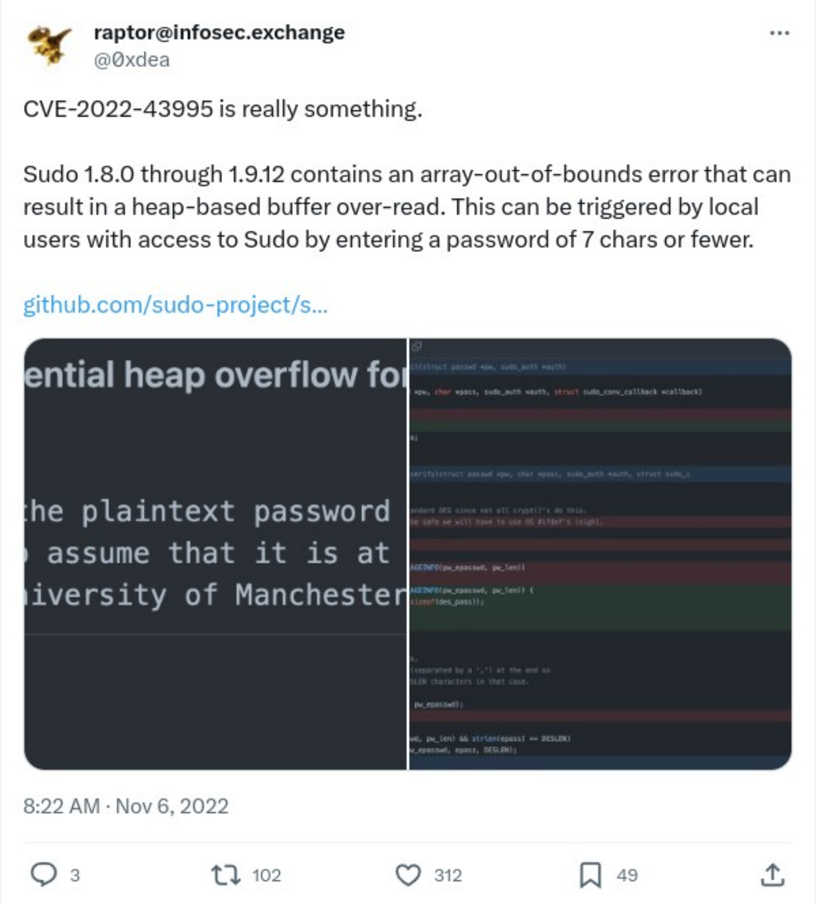
read & write CIV with

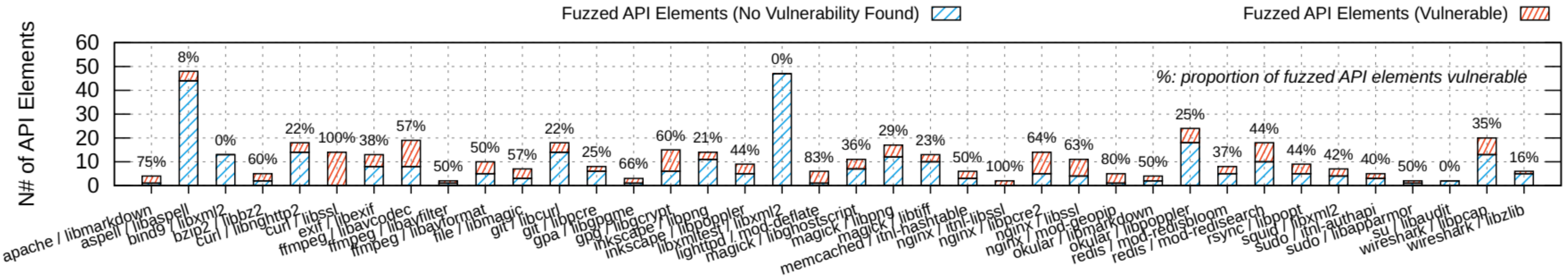
password < 8 characters

```c
int sudo_passwd_verify(struct passwd *pw, char *pass,
  sudo_auth *auth, struct sudo_conv_callback *cb) {
  /* ... abbreviated ... */
  sav = pass[8]; // read CIV
  pass[8] = '\0'; // write CIV
} /* ... abbreviated ... */
```

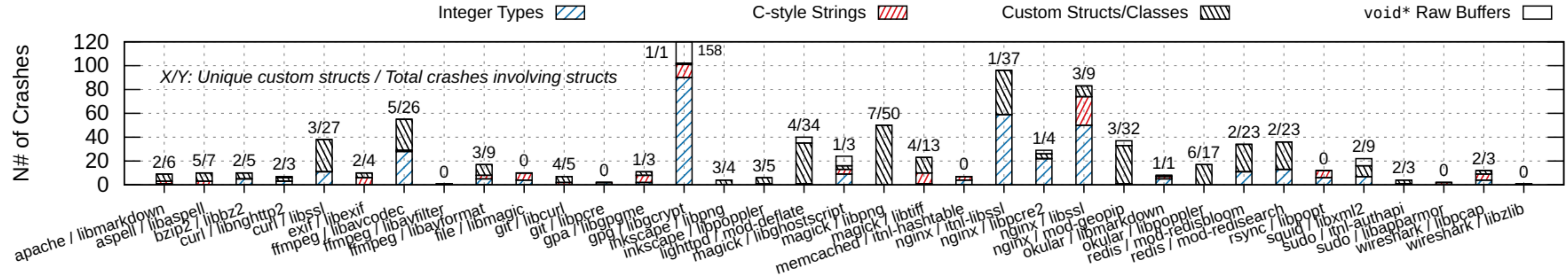
Safeboxing sudo's authentication API:
read & write CIV with
password < 8 characters

By the way, the password comes from the command line CVE-2022-43995





Proportion of covered vulnerable endpoints versus covered endpoints for each scenario (see Table III for coverage).



High-level type classes involved in CIVs for each scenario.

Takeways

- **CIVs are widespread and compartmentalization without securing interfaces is mostly meaningless**

Takeways

- **CIVs are widespread and compartmentalization without securing interfaces is mostly meaningless**
- **Clear disparities among APIs**
  - There are large and almost totally CIV-free APIs
  - There are small and fully vulnerable APIs
  - **No correlation between API size and CIV count**
  - Some API design patterns (e.g. modules) are highly vulnerable because of a large amount of state exposure

Takeways

- **CIVs are widespread and compartmentalization without securing interfaces is mostly meaningless**
- **Clear disparities among APIs**
  - There are large and almost totally CIV-free APIs
  - There are small and fully vulnerable APIs
  - **No correlation between API size and CIV count**
  - Some API design patterns (e.g. modules) are highly vulnerable because of a large amount of state exposure
- **CIVs are high-impact**
  - 75% of scenarios have at least 1 write vulnerability
  - 70% of R/W and 50% of execute vulnerabilities are arbitrary

Takeways

- **CIVs are widespread and compartmentalization without securing interfaces is mostly meaningless**
- **Clear disparities among APIs**
  - There are large and almost totally CIV-free APIs
  - There are small and fully vulnerable APIs
  - **No correlation between API size and CIV count**
  - Some API design patterns (e.g. modules) are highly vulnerable because of a large amount of state exposure
- **CIVs are high-impact**
  - 75% of scenarios have at least 1 write vulnerability
  - 70% of R/W and 50% of execute vulnerabilities are arbitrary
- **Fixing CIVs goes beyond writing simple checks**
  - Requires API redesign in many cases, hard to automate

The Path Forward

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high
  - **Performance overhead** brought by that practice is high

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high
  - **Performance overhead** brought by that practice is high
  - **Security benefits are hard to quantify**, difficult to compare approaches

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high
  - **Performance overhead** brought by that practice is high
  - **Security benefits are hard to quantify**, difficult to compare approaches
  - **Lack of visibility on the cost and benefits** pre-compartmentalization

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high
  - **Performance overhead** brought by that practice is high
  - **Security benefits are hard to quantify**, difficult to compare approaches
  - **Lack of visibility on the cost and benefits** pre-compartmentalization
  - Existing efforts lack **flexibility**

Outstanding Challenges

- **As of today, why did compartmentalization failed to establish itself as a standard engineering practice?**
  - Existing successful approaches running in production have been **designed from scratch** with compartmentalization in mind
  - **Engineering effort** required to compartmentalize a monolithic application is high
  - **Performance overhead** brought by that practice is high
  - **Security benefits are hard to quantify**, difficult to compare approaches
  - **Lack of visibility on the cost and benefits** pre-compartmentalization
  - Existing efforts lack **flexibility**
  - **Availability** is generally scoped out

Research Needed!

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**
  - But unlikely that everything can be automated (interface safety/redesign)

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**
  - But unlikely that everything can be automated (interface safety/redesign)
- Need efficient **isolation and sharing mechanisms** (e.g. CHERI[1])

[1]Watson, Robert NM, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave et al. "CHERI: A hybrid capability-system architecture for scalable software compartmentalization." In 2015 IEEE Symposium on Security and Privacy, pp. 20-37. IEEE, 2015.

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**
  - But unlikely that everything can be automated (interface safety/redesign)
- Need efficient **isolation and sharing mechanisms** (e.g. CHERI[1])
- Need **metrics** to validate, evaluate, quantify and compare approaches (e.g. policies)

[1]Watson, Robert NM, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave et al. "CHERI: A hybrid capability-system architecture for scalable software compartmentalization." In 2015 IEEE Symposium on Security and Privacy, pp. 20-37. IEEE, 2015.

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**
  - But unlikely that everything can be automated (interface safety/redesign)
- Need efficient **isolation and sharing mechanisms** (e.g. CHERI[1])
- Need **metrics** to validate, evaluate, quantify and compare approaches (e.g. policies)
- Need tools to **estimate** costs & benefits pre-compartmentalization

[1]Watson, Robert NM, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave et al. "CHERI: A hybrid capability-system architecture for scalable software compartmentalization." In 2015 IEEE Symposium on Security and Privacy, pp. 20-37. IEEE, 2015.

Research Needed!

- Need approaches focusing on **retrofitting** compartmentalization in legacy monolithic applications (including OS kernels)
- Need more **automation**
  - But unlikely that everything can be automated (interface safety/redesign)
- Need efficient **isolation and sharing mechanisms** (e.g. CHERI[1])
- Need **metrics** to validate, evaluate, quantify and compare approaches (e.g. policies)
- Need tools to **estimate** costs & benefits pre-compartmentalization
- Need more focus on **flexibility**[2] and **availability**

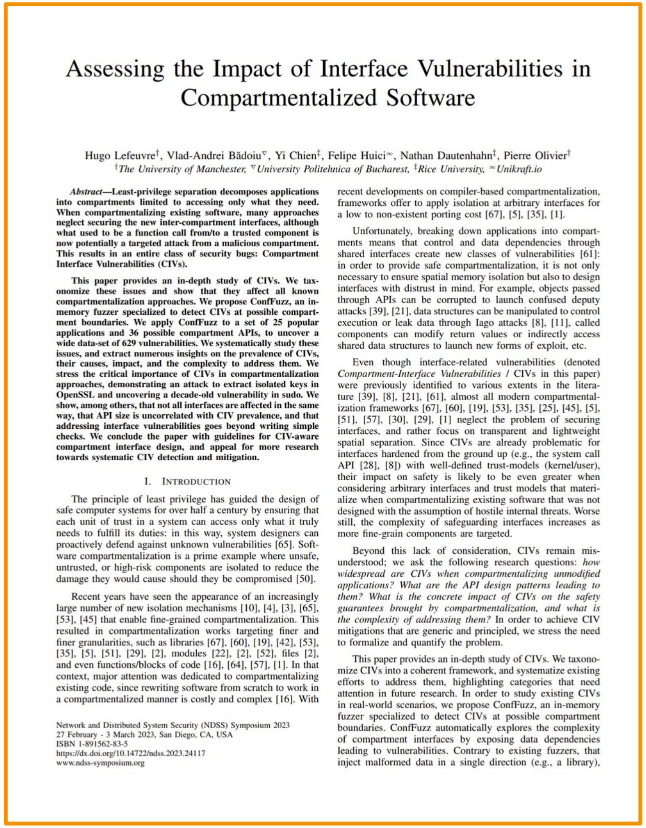
[1]Watson, Robert NM, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave et al. "CHERI: A hybrid capability-system architecture for scalable software compartmentalization." In 2015 IEEE Symposium on Security and Privacy, pp. 20-37. IEEE, 2015.
[2]Lefeuvre, Hugo, Vlad-Andrei Bădoiu, Alexander Jung, Stefan Lucian Teodorescu, Sebastian Rauch, Felipe Huici, Costin Raiciu, and Pierre Olivier. "FlexOS: towards flexible OS isolation." In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 467-482. 2022.

Links & credits

Availability & Credits

Everything is Open!

- ConfFuzz NDSS'23 Paper: https://arxiv.org/pdf/2212.12904.pdf
- Project website: https://conffuzz.github.io
- Main author: **Hugo Lefeuvre** https://research.manchester.ac.uk/en/persons/hugo.lefeuvre



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Hugo Lefeuvre[†], Vlad-Andrei Bădoiu[▽], Yi Chien[‡], Felipe Huici[∞], Nathan Dautenhahn[‡], Pierre Olivier[†]
[†]*The University of Manchester,* [▽]*University Politehnica of Bucharest,* [‡]*Rice University,* [∞]*Unikraft.io*

***Abstract*—Least-privilege separation decomposes applications into compartments limited to accessing only what they need. When compartmentalizing existing software, many approaches neglect securing the new inter-compartment interfaces, although what used to be a function call from/to a trusted component is now potentially a targeted attack from a malicious compartment. This results in an entire class of security bugs: Compartment Interface Vulnerabilities (CIVs).**

This paper provides an in-depth study of CIVs. We taxonomize these issues and show that they affect all known compartmentalization approaches. We propose ConfFuzz, an in-memory fuzzer specialized to detect CIVs at possible compartment boundaries. We apply ConfFuzz to a set of 25 popular applications and 36 possible compartment APIs, to uncover a wide data-set of 629 vulnerabilities. We systematically study these issues, and extract numerous insights on the prevalence of CIVs, their causes, impact, and the complexity to address them. We stress the critical importance of CIVs in compartmentalization approaches, demonstrating an attack to extract isolated keys in OpenSSL and uncovering a decade-old vulnerability in sudo. We show, among others, that not all interfaces are affected in the same way, that API size is uncorrelated with CIV prevalence, and that addressing interface vulnerabilities goes beyond writing simple checks. We conclude the paper with guidelines for CIV-aware compartment interface design, and appeal for more research towards systematic CIV detection and mitigation.

I. INTRODUCTION

The principle of least privilege has guided the design of safe computer systems for over half a century by ensuring that each unit of trust in a system can access only what it truly needs to fulfill its duties: in this way, system designers can proactively defend against unknown vulnerabilities [65]. Software compartmentalization is a prime example where unsafe, untrusted, or high-risk components are isolated to reduce the damage they would cause should they be compromised [50].

Recent years have seen the appearance of an increasingly large number of new isolation mechanisms [10], [4], [3], [65], [53], [45] that enable fine-grained compartmentalization. This resulted in compartmentalization works targeting finer and finer granularities, such as libraries [67], [60], [19], [42], [53], [35], [5], [51], [29], [2], modules [22], [2], [52], files [2], and even functions/blocks of code [16], [64], [57], [1]. In that context, major attention was dedicated to compartmentalizing existing code, since rewriting software from scratch to work in a compartmentalized manner is costly and complex [16]. With recent developments on compiler-based compartmentalization, frameworks offer to apply isolation at arbitrary interfaces for a low to non-existent porting cost [67], [5], [35], [1].

Unfortunately, breaking down applications into compartments means that control and data dependencies through shared interfaces create new classes of vulnerabilities [61]: in order to provide safe compartmentalization, it is not only necessary to ensure spatial memory isolation but also to design interfaces with distrust in mind. For example, objects passed through APIs can be corrupted to launch confused deputy attacks [39], [21], data structures can be manipulated to control execution or leak data through Iago attacks [8], [11], called components can modify return values or indirectly access shared data structures to launch new forms of exploit, etc.

Even though interface-related vulnerabilities (denoted *Compartment-Interface Vulnerabilities* / CIVs in this paper) were previously identified to various extents in the literature [39], [8], [21], [61], almost all modern compartmentalization frameworks [67], [60], [19], [53], [35], [25], [45], [5], [51], [57], [30], [29], [1] neglect the problem of securing interfaces, and rather focus on transparent and lightweight spatial separation. Since CIVs are already problematic for interfaces hardened from the ground up (e.g., the system call API [28], [8]) with well-defined trust-models (kernel/user), their impact on safety is likely to be even greater when considering arbitrary interfaces and trust models that materialize when compartmentalizing existing software that was not designed with the assumption of hostile internal threats. Worse still, the complexity of safeguarding interfaces increases as more fine-grain components are targeted.

Beyond this lack of consideration, CIVs remain misunderstood; we ask the following research questions: *how widespread are CIVs when compartmentalizing unmodified applications? What are the API design patterns leading to them? What is the concrete impact of CIVs on the safety guarantees brought by compartmentalization, and what is the complexity of addressing them?* In order to achieve CIV mitigations that are generic and principled, we stress the need to formalize and quantify the problem.

This paper provides an in-depth study of CIVs. We taxonomize CIVs into a coherent framework, and systematize existing efforts to address them, highlighting categories that need attention in future research. In order to study existing CIVs in real-world scenarios, we propose ConfFuzz, an in-memory fuzzer specialized to detect CIVs at possible compartment boundaries. ConfFuzz automatically explores the complexity of compartment interfaces by exposing data dependencies leading to vulnerabilities. Contrary to existing fuzzers, that inject malformed data in a single direction (e.g., a library),

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24117
www.ndss-symposium.org