Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Hugo Lefeuvre[1], Vlad-Andrei Badoiu[2], Yi Chien[3], Felipe Huici[4], Nathan Dautenhahn[3], Pierre Olivier[1]

[1]The University of Manchester, [2]University Politehnica of Bucharest, [3]Rice University, [4]Unikraft.io

NDSS'23, 28th February 2023, San Diego, CA

Software compartmentalization promises…

Software compartmentalization =

Software compartmentalization promises…

Software compartmentalization =

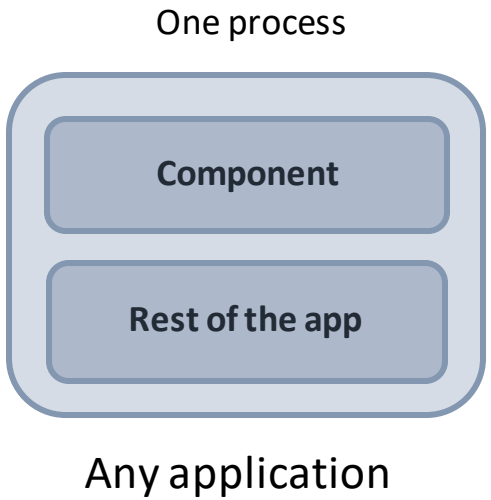
- Decompose software into lesser-privileged components
- Components only have access to *what they need to do their job*

Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

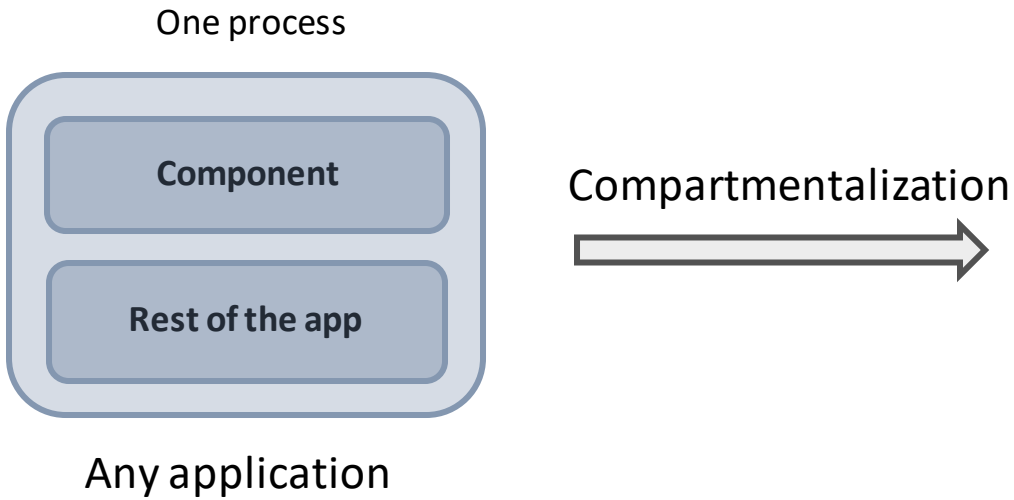


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

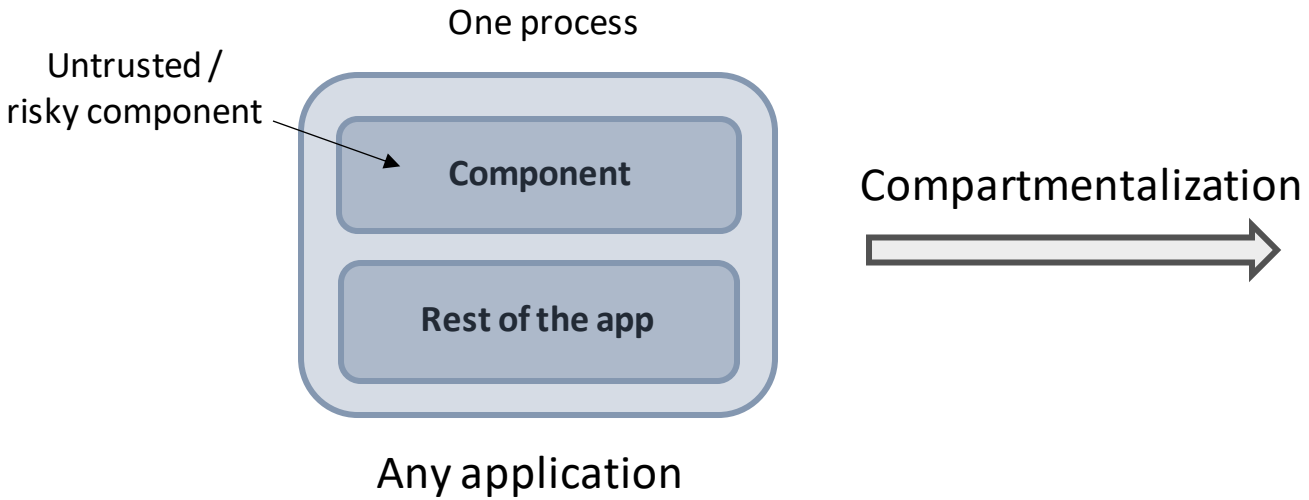


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

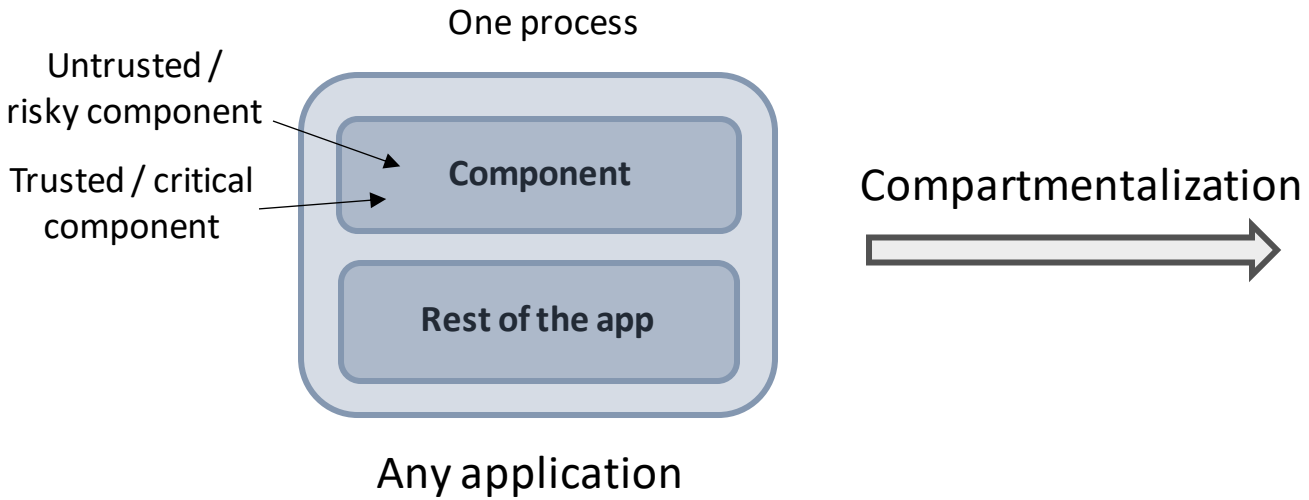


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

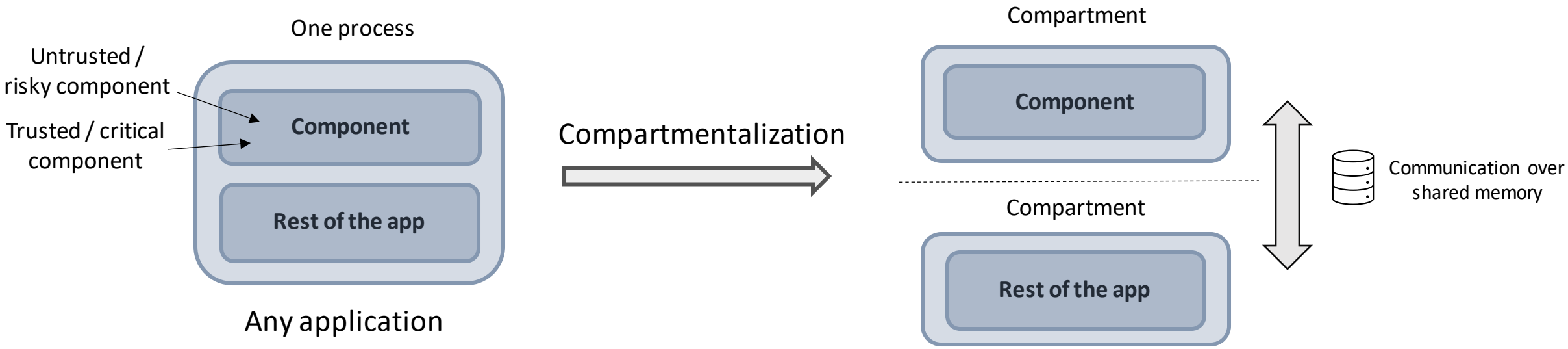


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

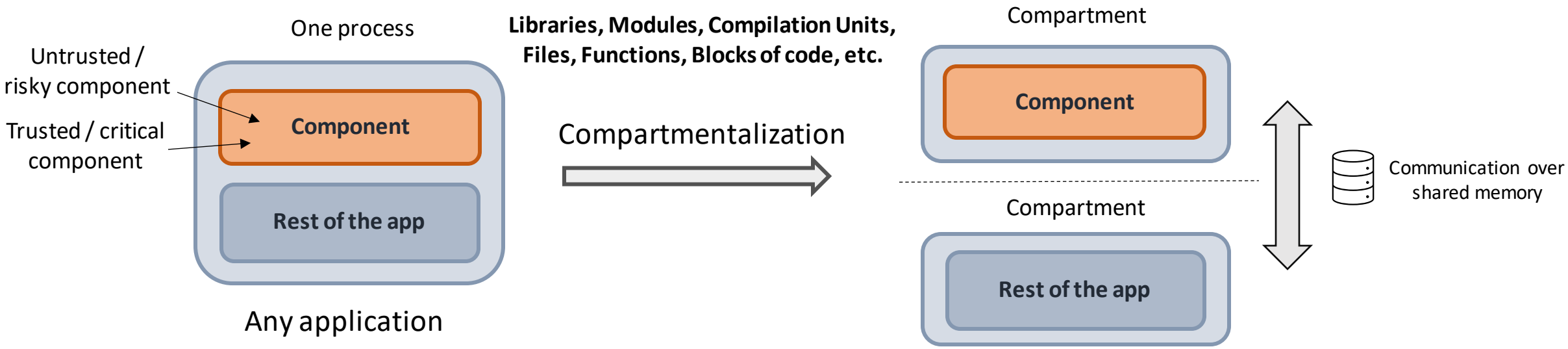


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

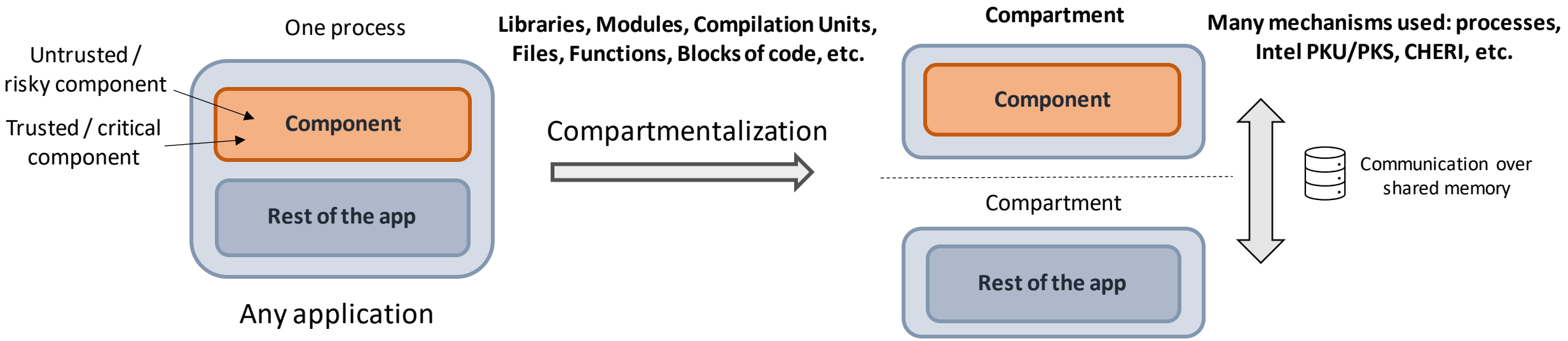


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

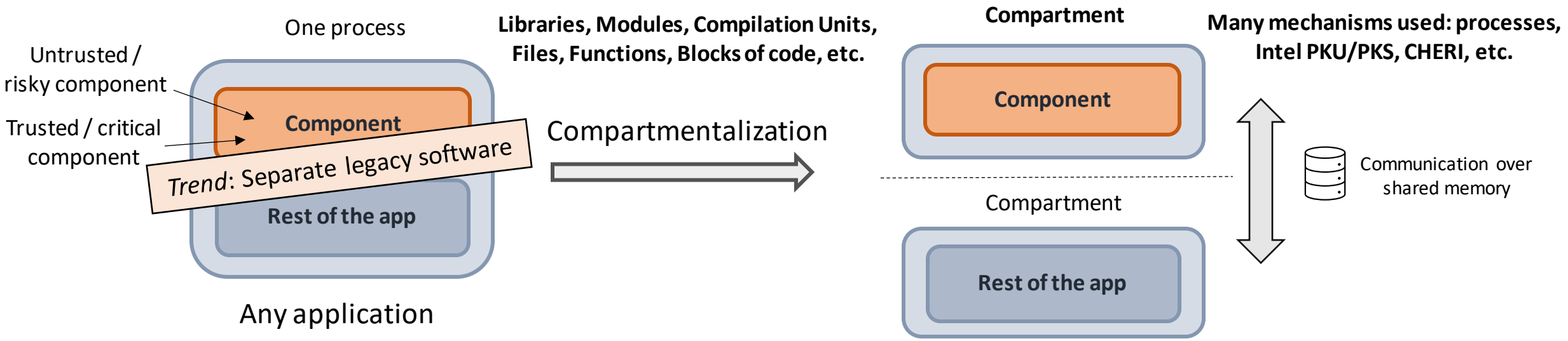


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

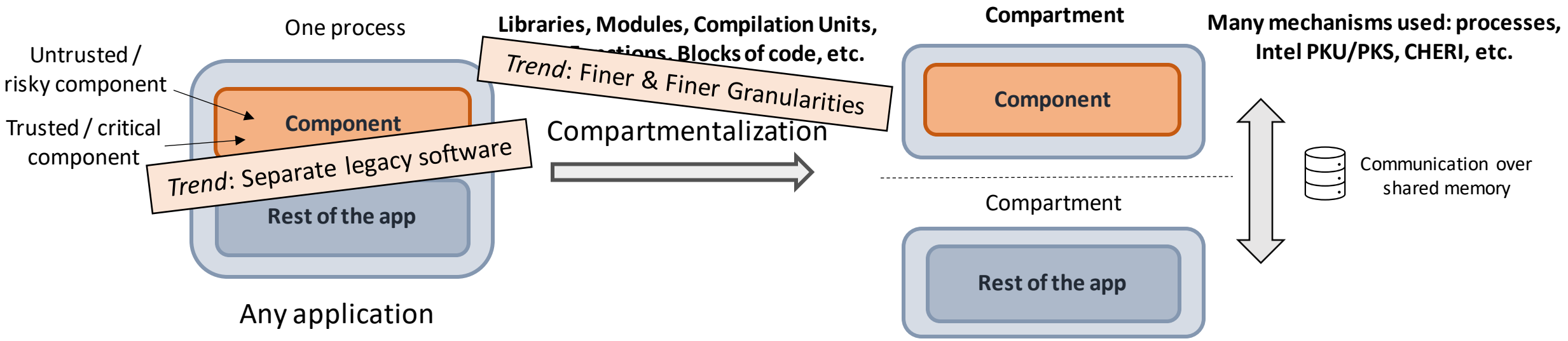


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

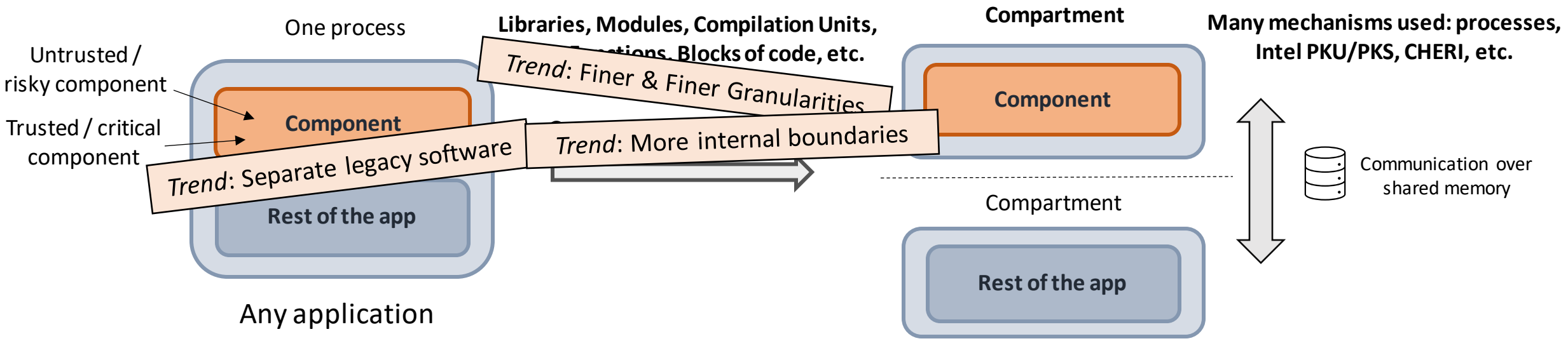


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

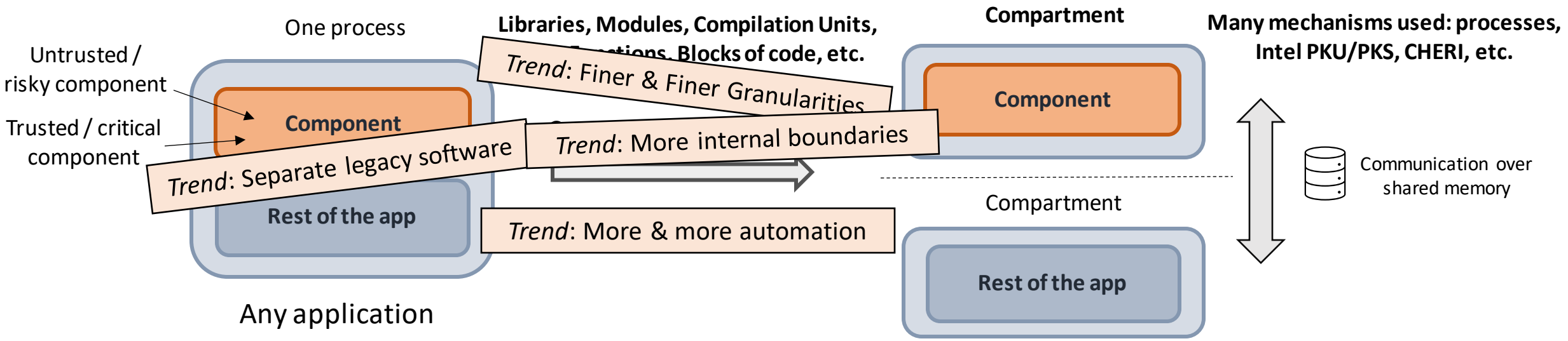


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*

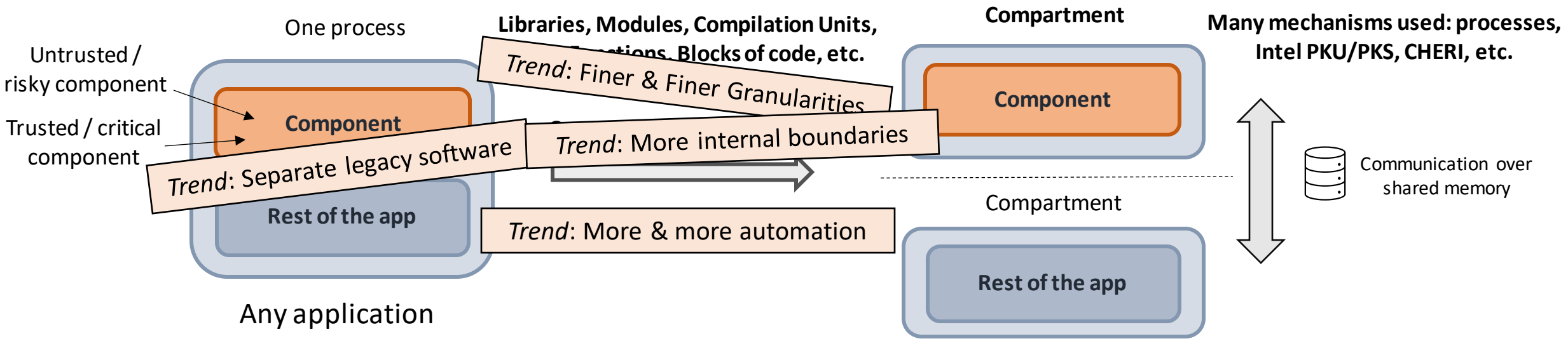


Software compartmentalization promises…

Software compartmentalization =

Decompose software into lesser-privileged components

Components only have access to *what they need to do their job*



Goals of these works: compartmentalization of **legacy** software

… with a **low engineering effort** … at a **low performance cost**

But interfaces make separation hard…

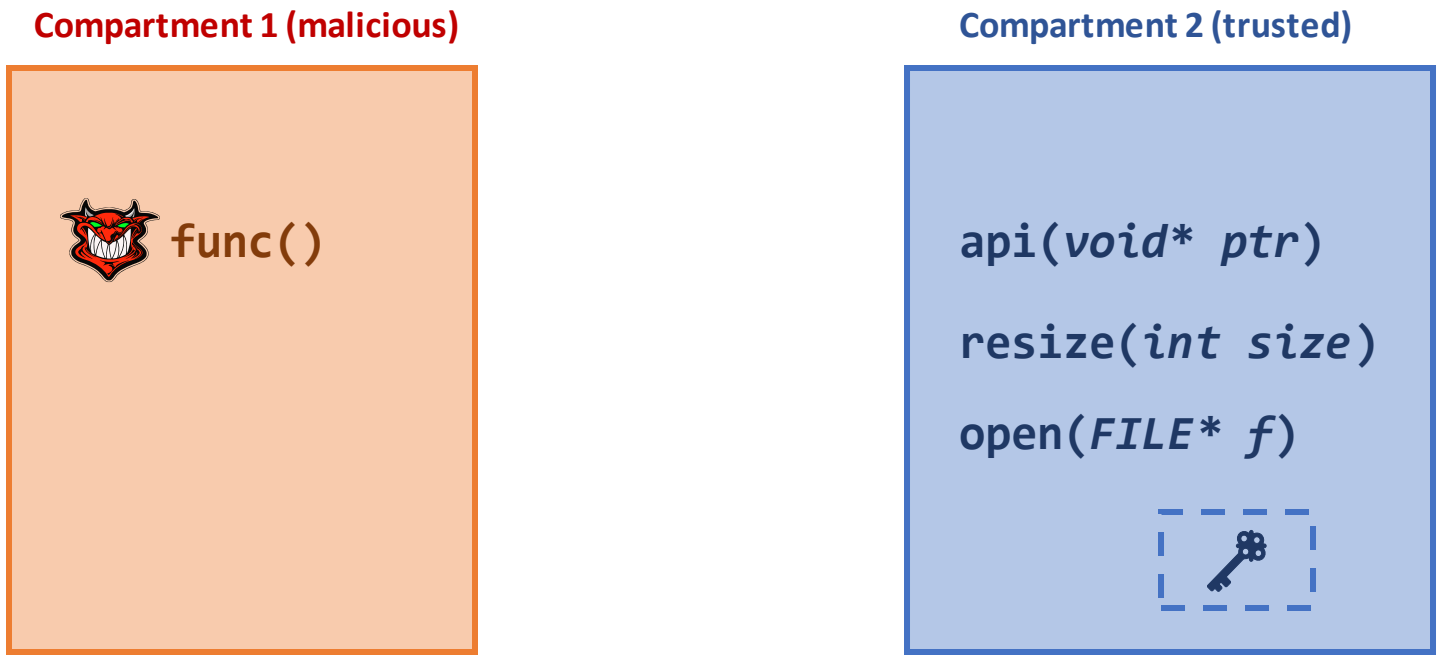
- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**

But interfaces make separation hard…

- Things are not as easy as they seem:

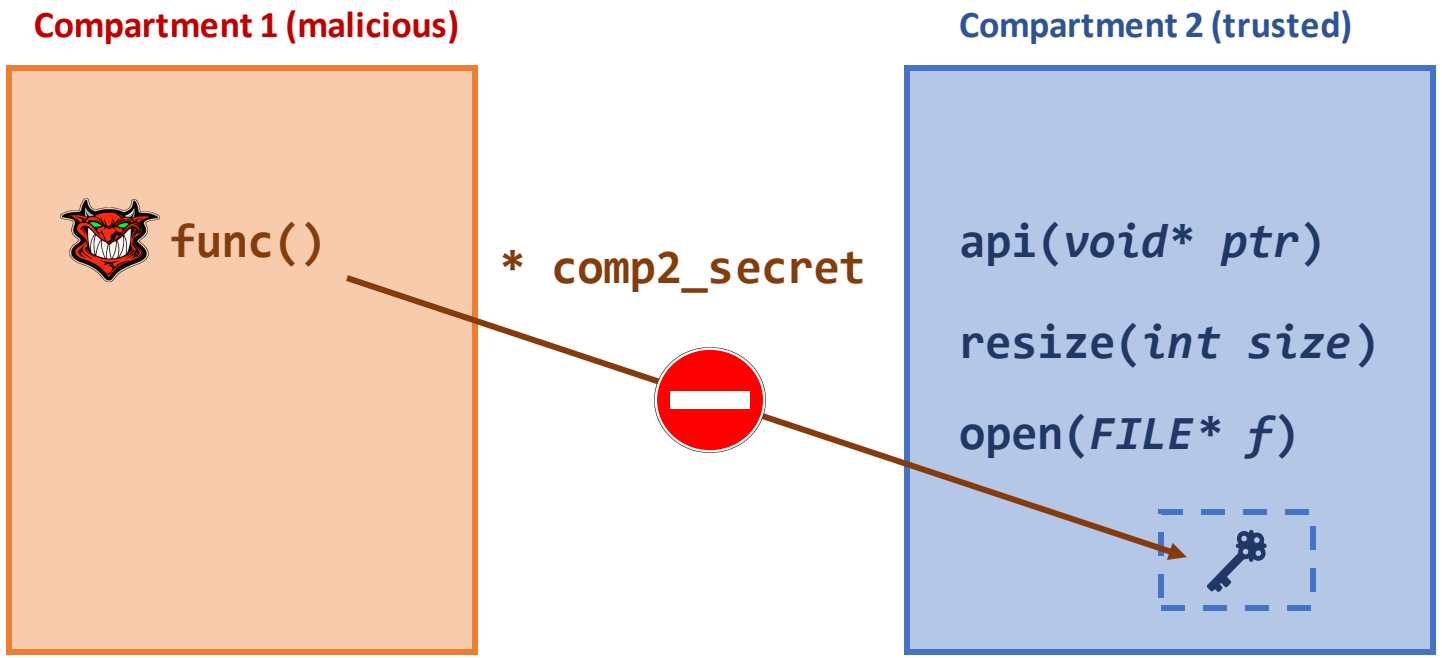
  Once separation is enforced, **cross-component interfaces become the attack surface**



But interfaces make separation hard…

- Things are not as easy as they seem:

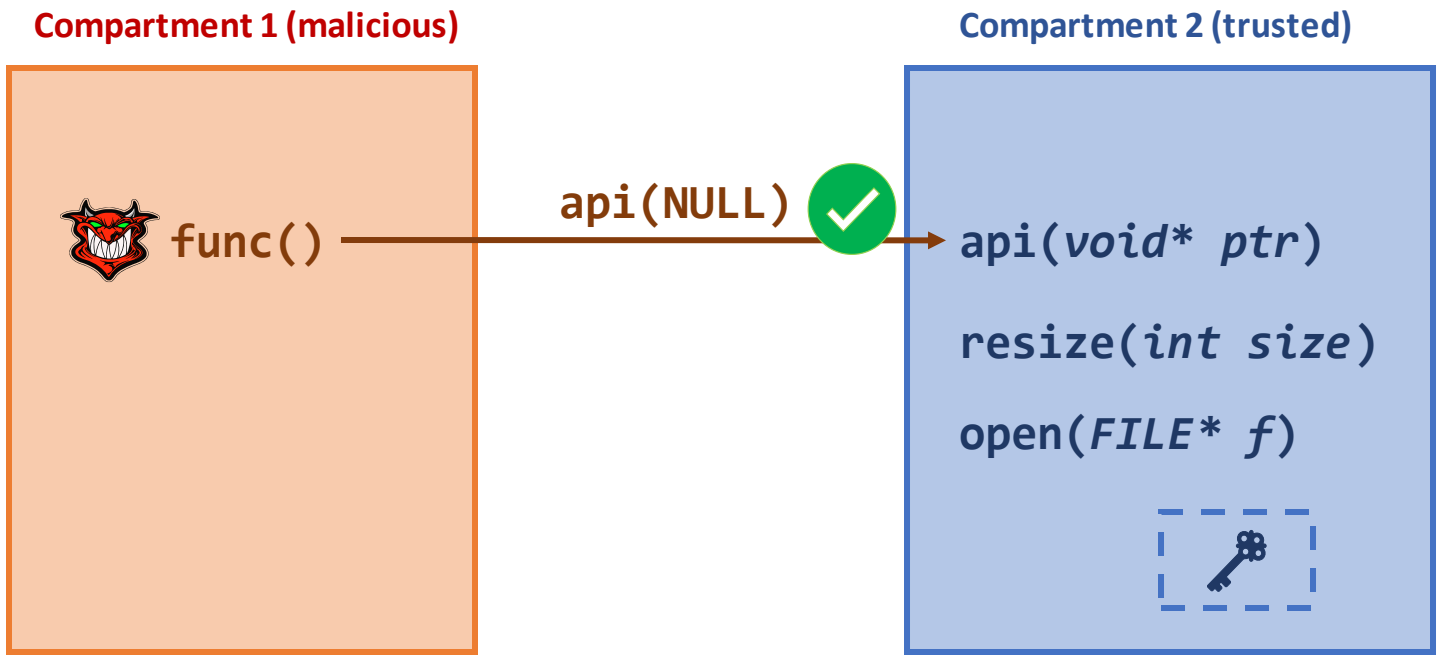
  Once separation is enforced, **cross-component interfaces become the attack surface**



But interfaces make separation hard…

- Things are not as easy as they seem:

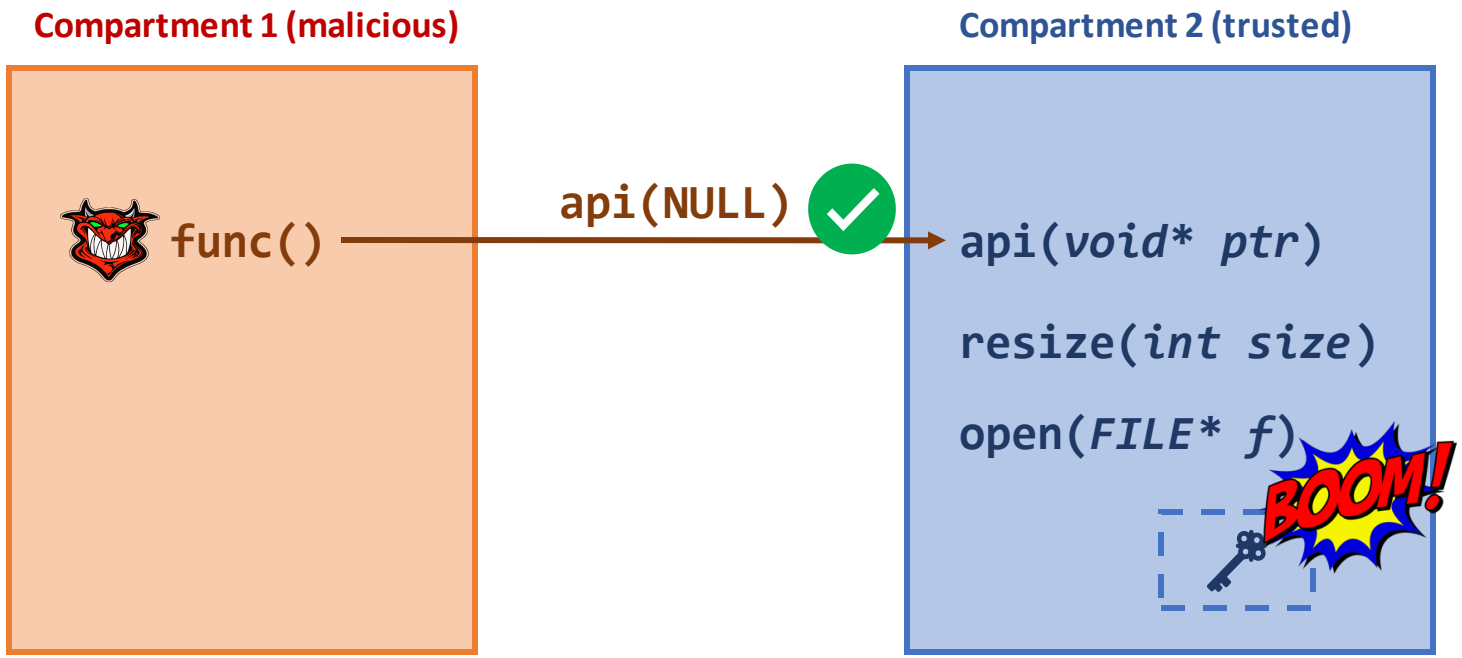
  Once separation is enforced, **cross-component interfaces become the attack surface**



But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



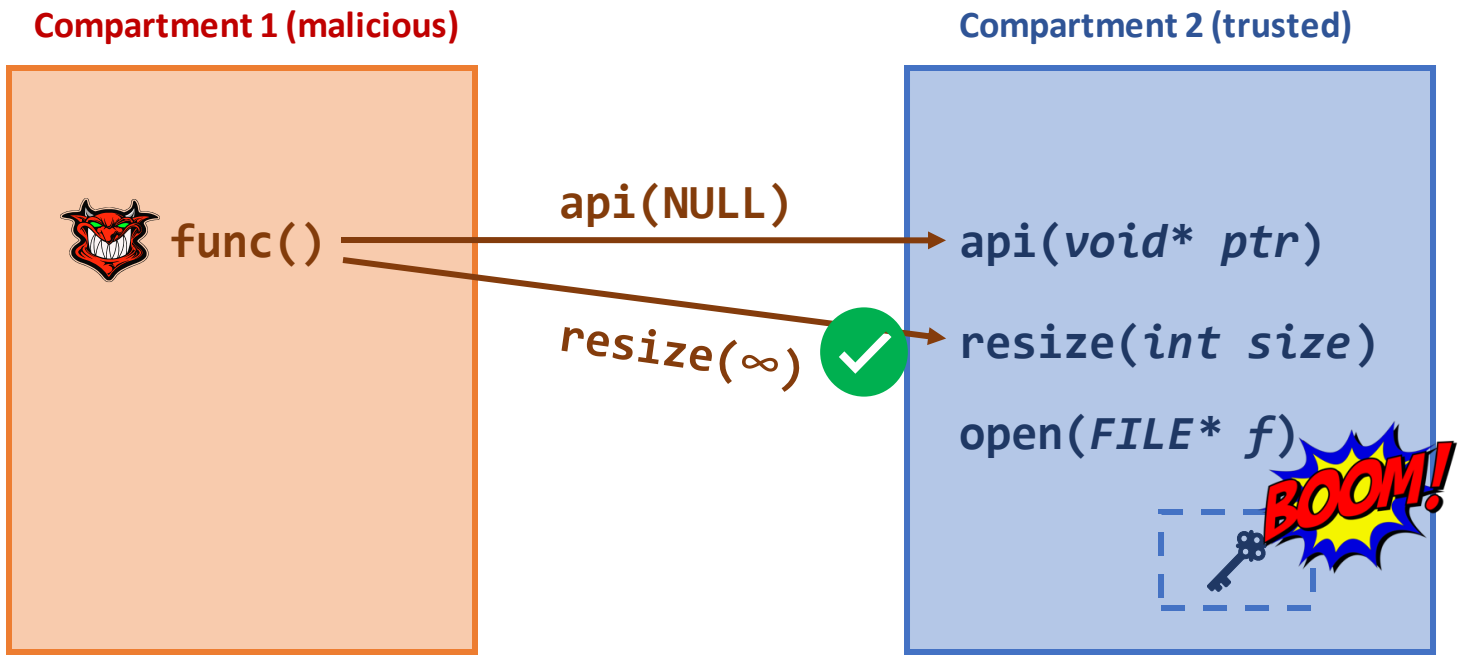
- Dereference of corrupted pointer

```
void api(void* ptr) {
  // …
  * (char *) ptr = '\0';
  // …
}
```

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



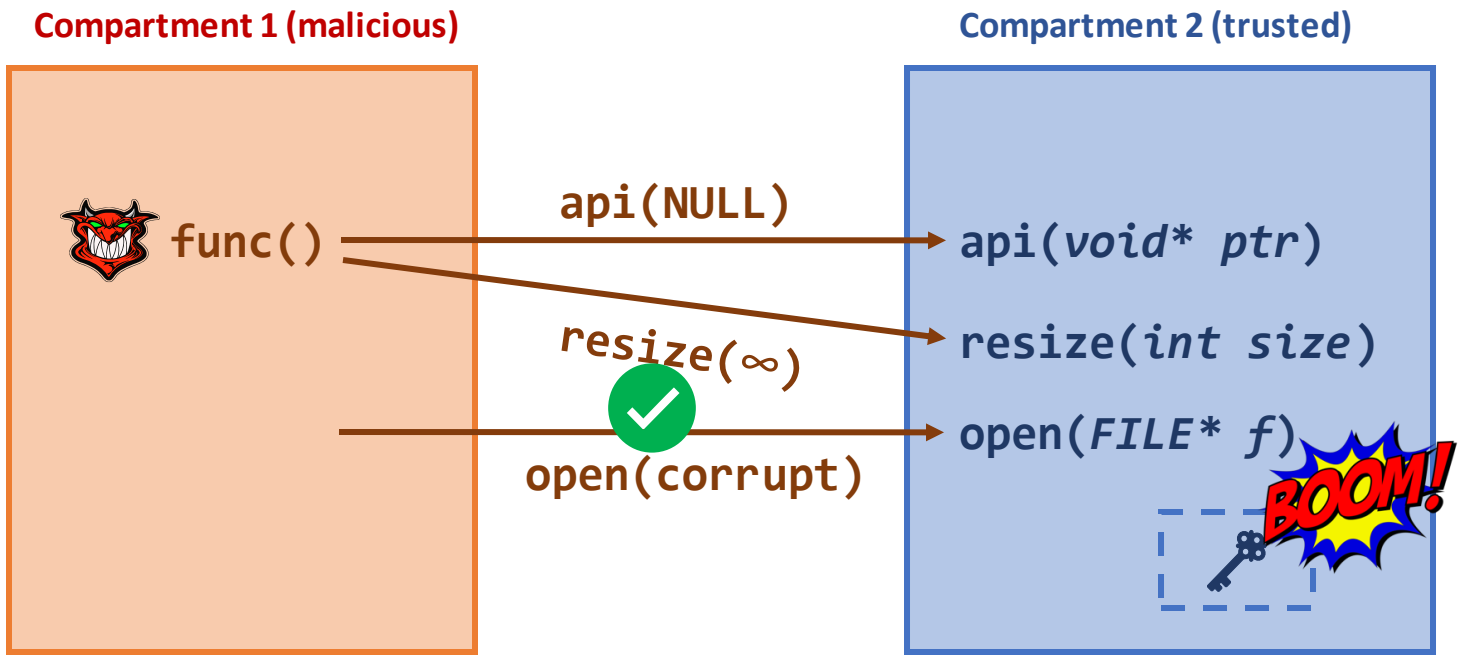
- Dereference of corrupted pointer
- Usage of corrupted indexing information

```
void resize(int size) {
  // …
  var = matrix[size];
  // …
}
```

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



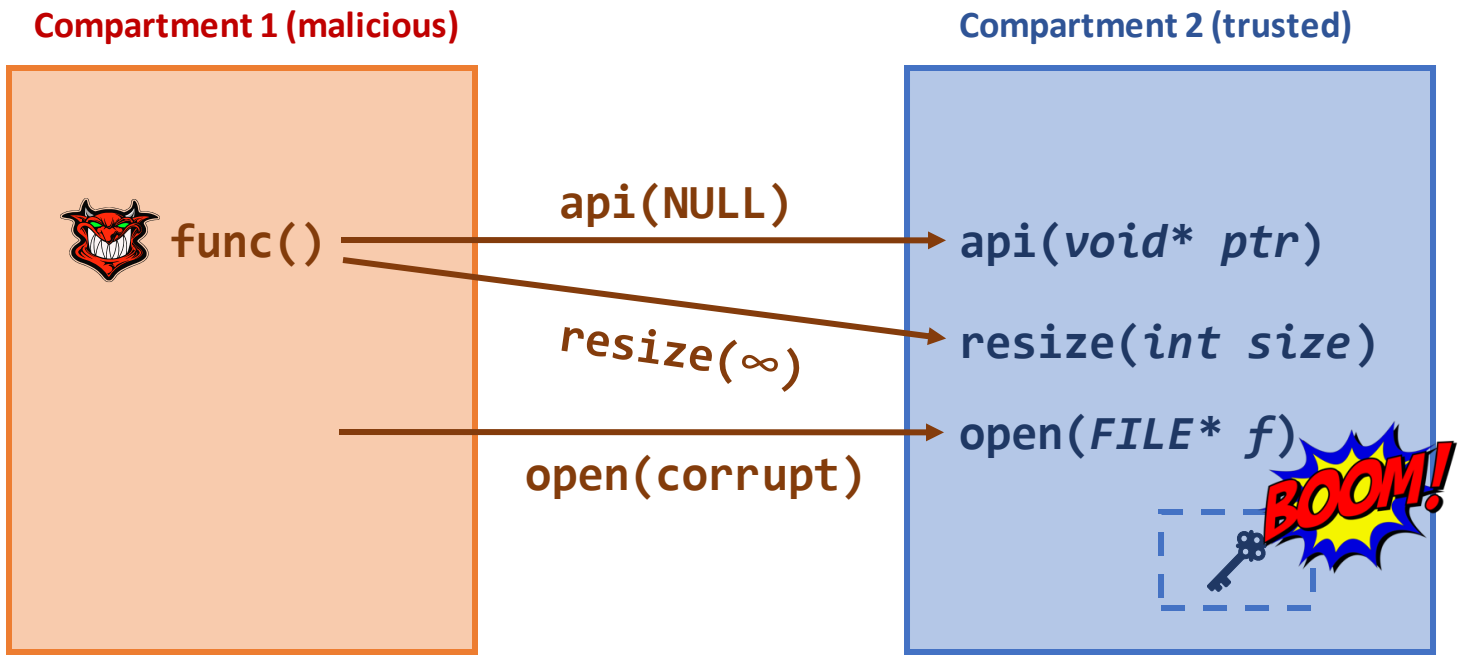
- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object

```
void open(FILE* f) {
  // …
  close(f);
  // …
}
```

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**

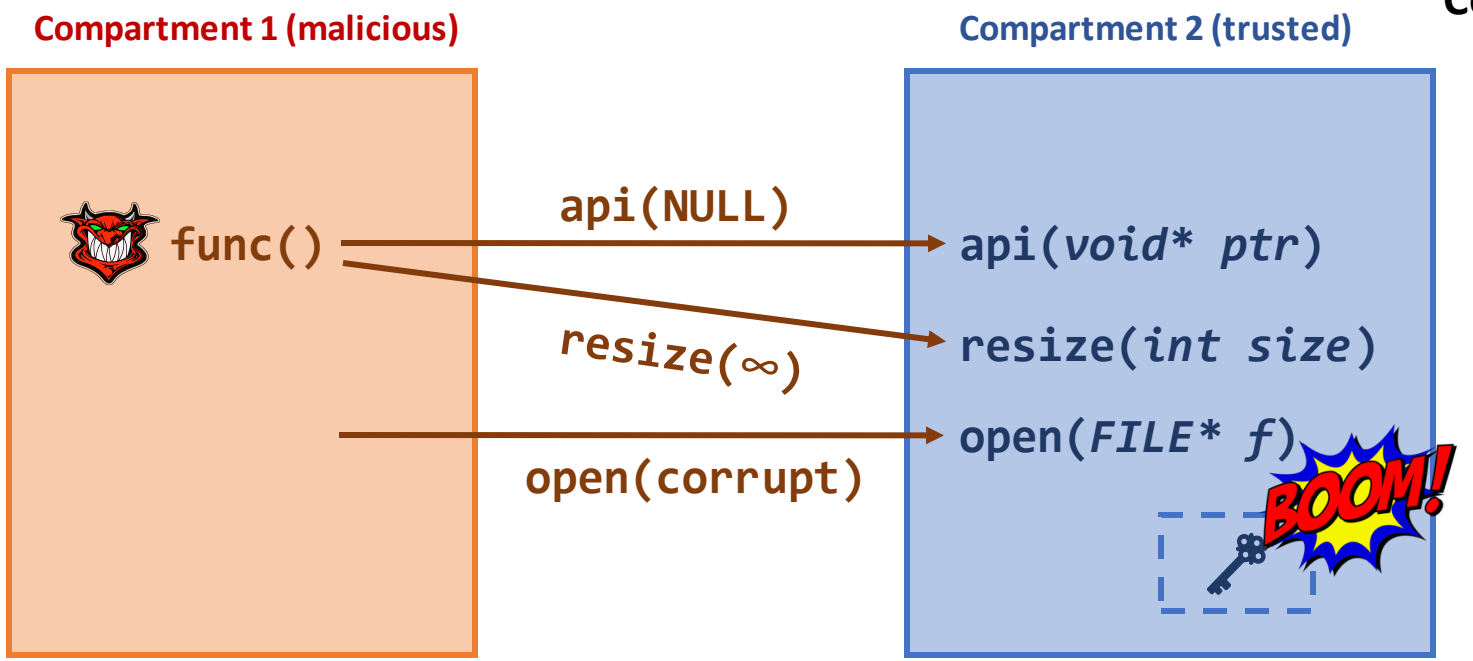


- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object
- … and many others!

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



We unify these vulnerabilities as:

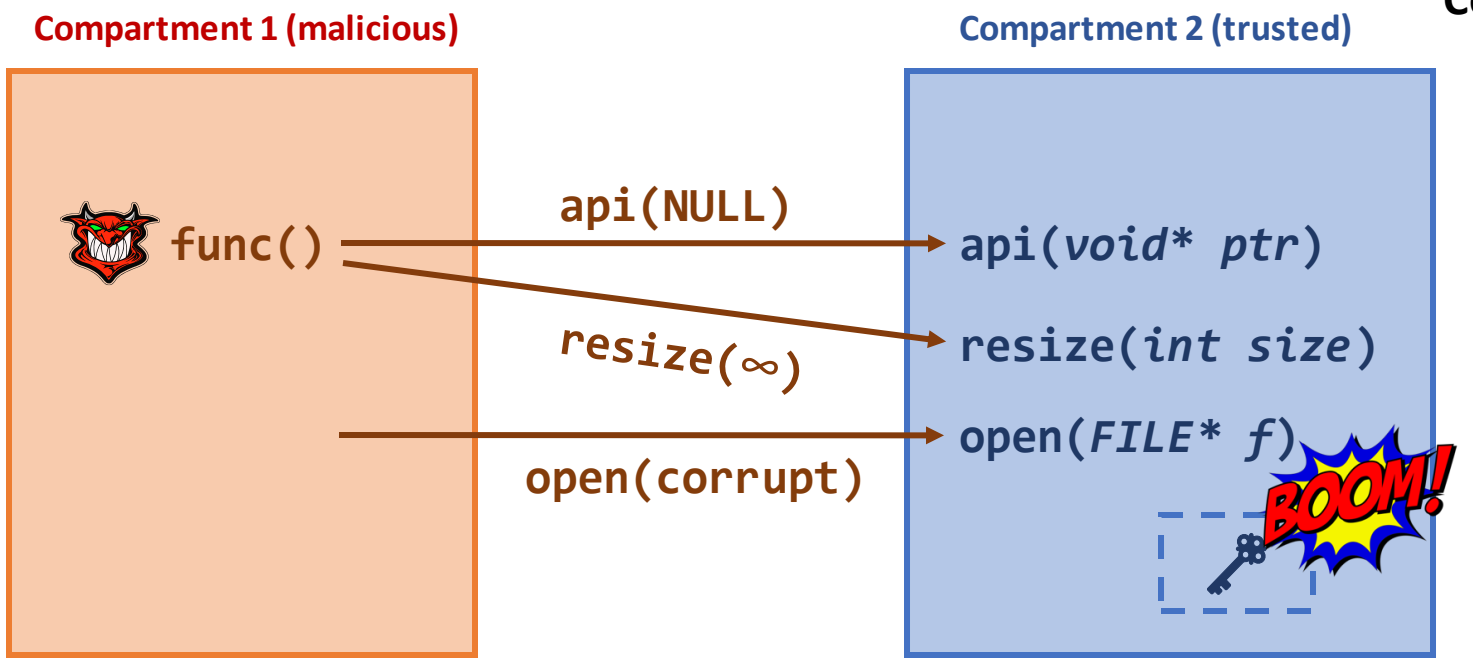
Compartment Interface Vulnerabilities (CIVs)

- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object
- … and many others!

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



We unify these vulnerabilities as:

Compartment Interface Vulnerabilities (CIVs)

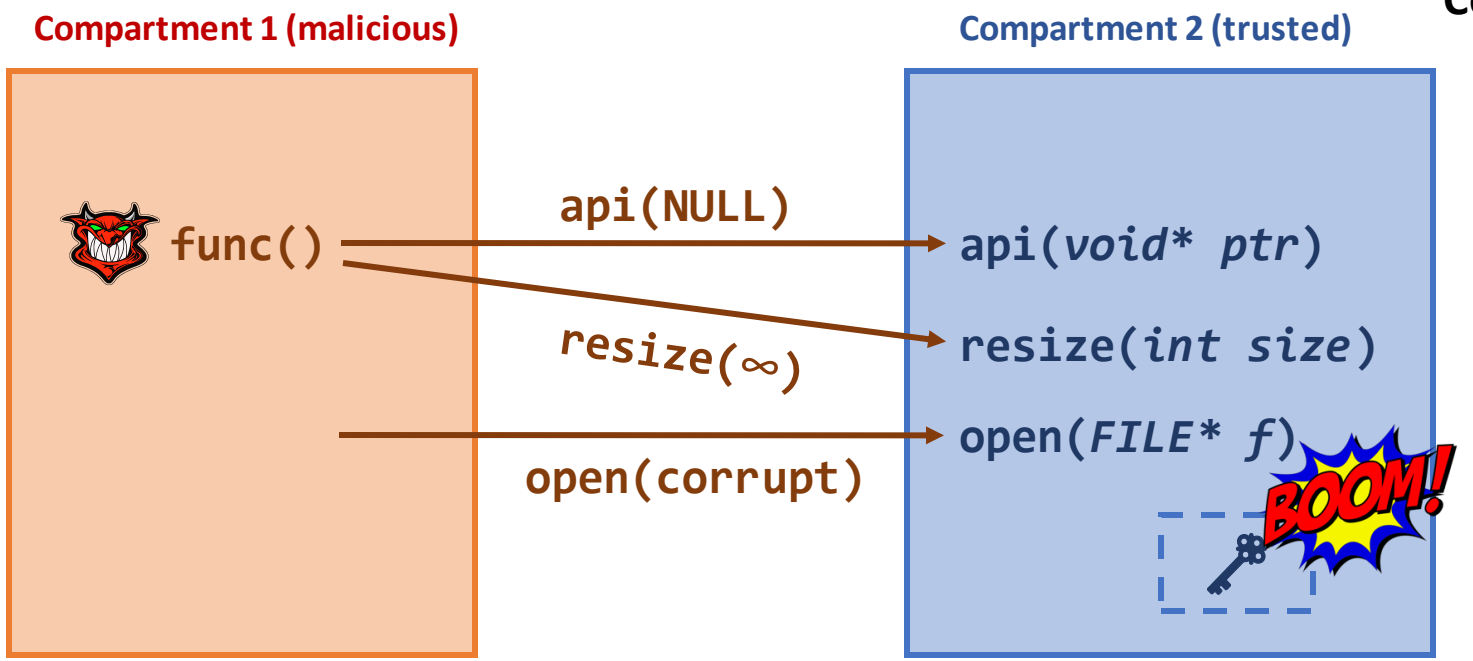
- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object
- … and many others!

CIVs encompass traditional confused deputies, Iago attacks, "DUIs"

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



We unify these vulnerabilities as:

Compartment Interface Vulnerabilities (CIVs)

- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object
- … and many others!

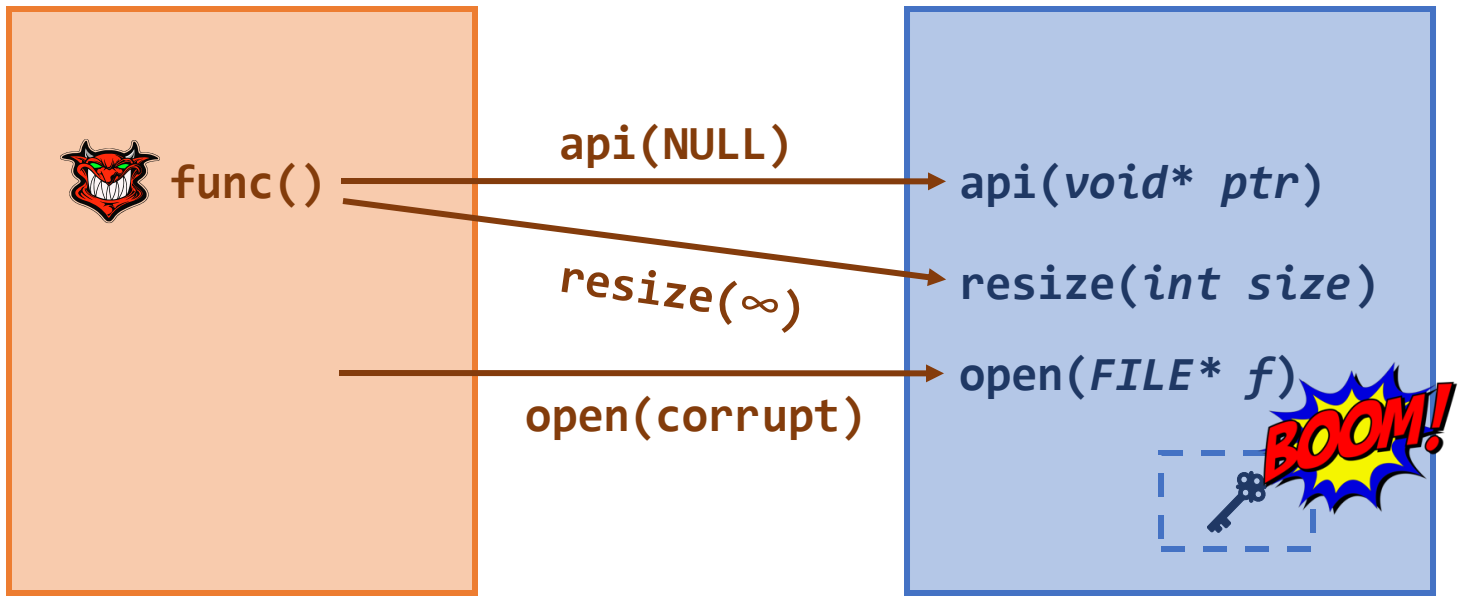
CIVs encompass traditional confused deputies, Iago attacks, "DUIs"

LOTS of them in unmodified components.

But interfaces make separation hard…

- Things are not as easy as they seem:

  Once separation is enforced, **cross-component interfaces become the attack surface**



We unify these vulnerabilities as:

Compartment Interface Vulnerabilities (CIVs)

- Dereference of corrupted pointer
- Usage of corrupted indexing information
- Usage of corrupted object
- … and many others!

CIVs encompass traditional confused deputies, Iago attacks, "DUIs"

LOTS of them in unmodified components. Affect **all** compartmentalization frameworks to various degrees.

When, and why do CIVs arise?

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

When, and why do CIVs arise?

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***

When, and why do CIVs arise?

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		

When, and why do CIVs arise?

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		

Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)

When, and why do CIVs arise?

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	

Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)

When, and why do CIVs arise?

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	

Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)

Usage of interface-crossing data without appropriate sanitization

When, and why do CIVs arise?

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

Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)

Usage of interface-crossing data without appropriate sanitization

When, and why do CIVs arise?

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
Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)	**Usage of interface-crossing data without appropriate sanitization**	**Many causes, missing copies, double fetches, lack of enforcement of API semantics, ...**

When, and why do CIVs arise?

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
Data over-sharing, sharing of uninitialized memory (incl. compiler-added padding)	**Usage of interface-crossing data without appropriate sanitization**	**Many causes, missing copies, double fetches, lack of enforcement of API semantics, ...**

Full taxonomy of CIVs in our paper!

How bad is "the CIV problem"?

Research questions:

How bad is "the CIV problem"?

Research questions:

- *How many CIVs are there at legacy, unported APIs?*

How bad is "the CIV problem"?

Research questions:

- *How many CIVs are there at legacy, unported APIs?*
- *Are all APIs similarly affected by CIVs? (e.g., library v.s. module APIs)*

How bad is "the CIV problem"?

Research questions:

- *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 is "the CIV problem"?

Research questions:

- *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?*

How bad is "the CIV problem"?

Need to understand these to achieve adequate counter-measures

Research questions:

- *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?*

How bad is "the CIV problem"?

Need to understand these to achieve adequate counter-measures

Research questions:

- *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?*

This work's approach:

How bad is "the CIV problem"?

Need to understand these to achieve adequate counter-measures

Research questions:

- *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?*

This work's approach:

- Design a **fuzzer / tool specialized to find CIVs**: *ConfFuzz*

How bad is "the CIV problem"?

Need to understand these to achieve adequate counter-measures

Research questions:

- *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?*

This work's approach:

- Design a **fuzzer / tool specialized to find CIVs**: *ConfFuzz*
- Apply it at scale to many applications and library interfaces to **gather a data set of real-world CIVs**

How bad is "the CIV problem"?

Need to understand these to achieve adequate counter-measures

Research questions:

- *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?*

This work's approach:

- Design a **fuzzer / tool specialized to find CIVs**: *ConfFuzz*
- Apply it at scale to many applications and library interfaces to **gather a data set of real-world CIVs**
- **Study, systematize, patternize** the resulting data set

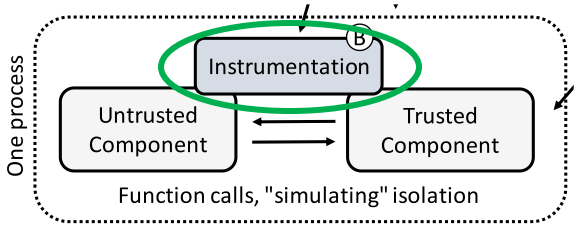
Fuzzing for CIVs

High-Level Overview:

Fuzzing for CIVs

High-Level Overview:

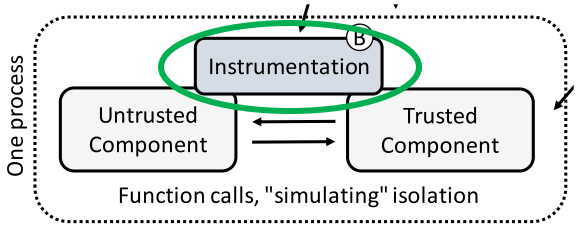
- Instrument application to intercept cross compartment function calls



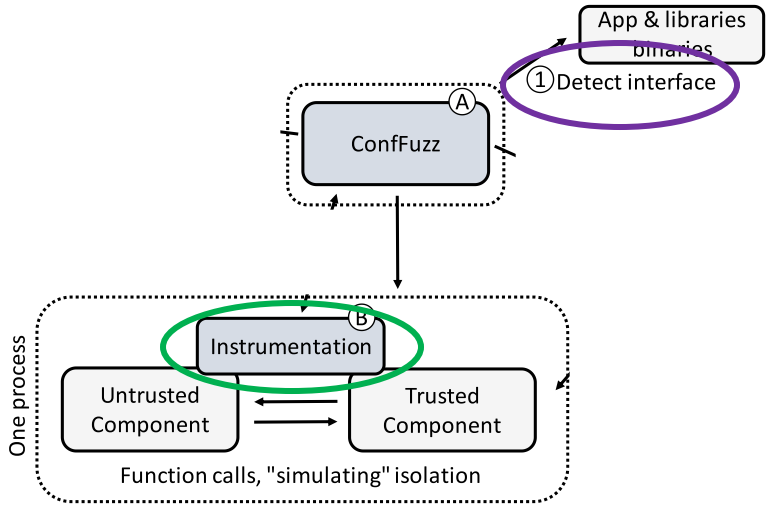
Fuzzing for CIVs

High-Level Overview:

- Instrument application to intercept cross compartment function calls
  - Based on Intel Pin (DBI)



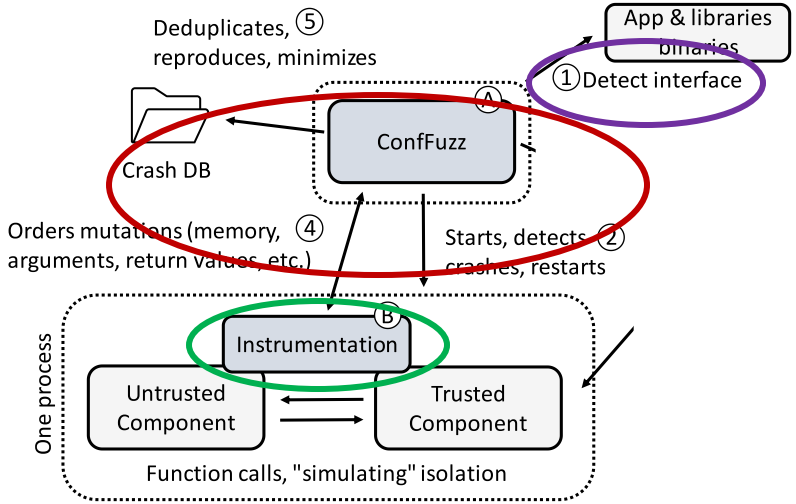
Fuzzing for CIVs



High-Level Overview:

- Instrument application to intercept cross compartment function calls
  - Based on Intel Pin (DBI)
- Interface (boundaries, types) is automatically detected using binary debug (DWARF) information

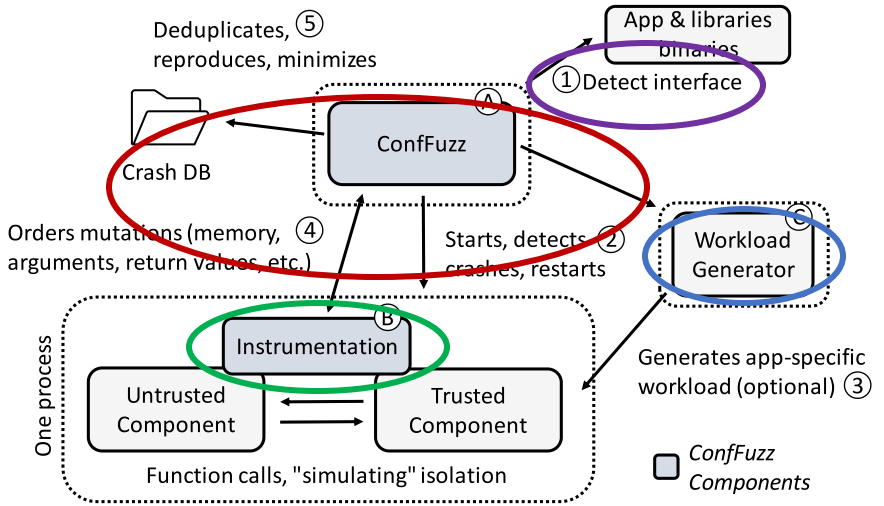
Fuzzing for CIVs



High-Level Overview:

- Instrument application to intercept cross compartment function calls
  - Based on Intel Pin (DBI)
- Interface (boundaries, types) is automatically detected using binary debug (DWARF) information
- A fuzzing monitor drives the exploration with custom CIV mutations

Fuzzing for CIVs



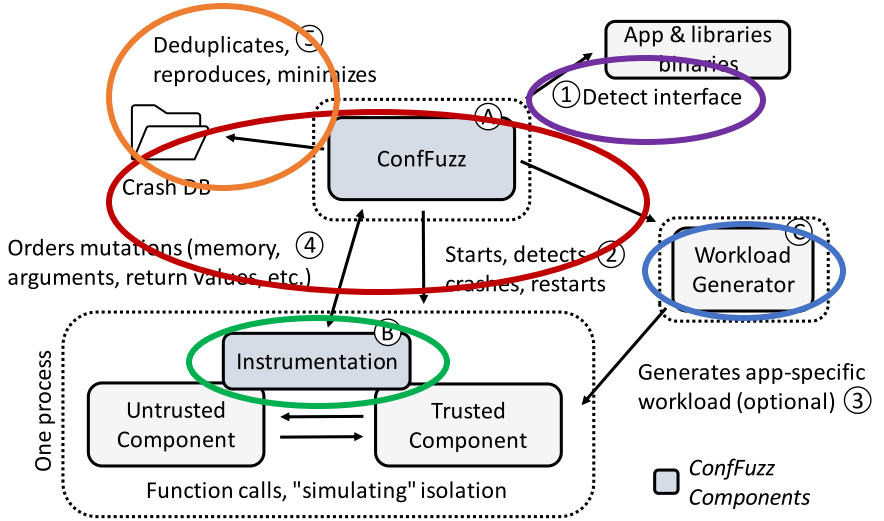
High-Level Overview:

- Instrument application to intercept cross compartment function calls
  - Based on Intel Pin (DBI)
- Interface (boundaries, types) is automatically detected using binary debug (DWARF) information
- A fuzzing monitor drives the exploration with custom CIV mutations
- The workload is application-specific (benchmark, test suite, etc.)

Fuzzing for CIVs

High-Level Overview:

- Instrument application to intercept cross compartment function calls
  - Based on Intel Pin (DBI)
- Interface (boundaries, types) is automatically detected using binary debug (DWARF) information
- A fuzzing monitor drives the exploration with custom CIV mutations
- The workload is application-specific (benchmark, test suite, etc.)
- The fuzzing monitor automatically triages and stores crash reports



Study results: Overview

Using ConfFuzz we gathered a substantial data set

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

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		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

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	2
	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	2
	Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78	103

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**

Study results: CIV prevalence

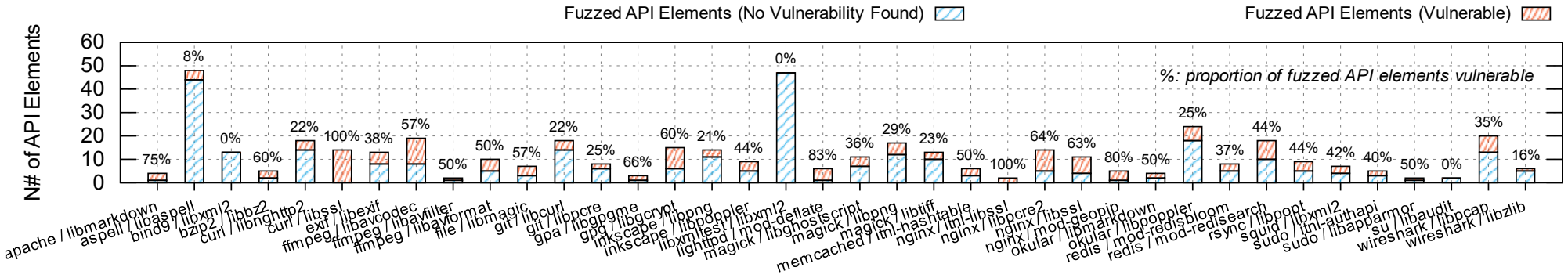
"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

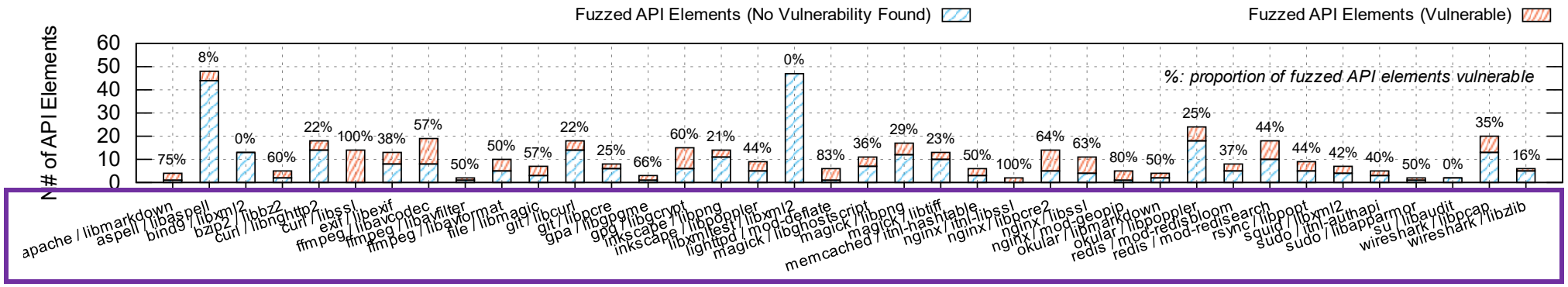
- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API



Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API

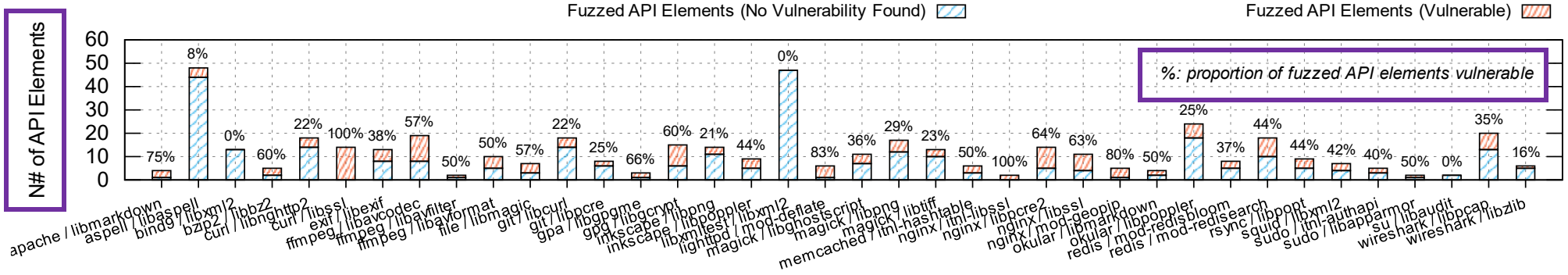


For each scenario

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API

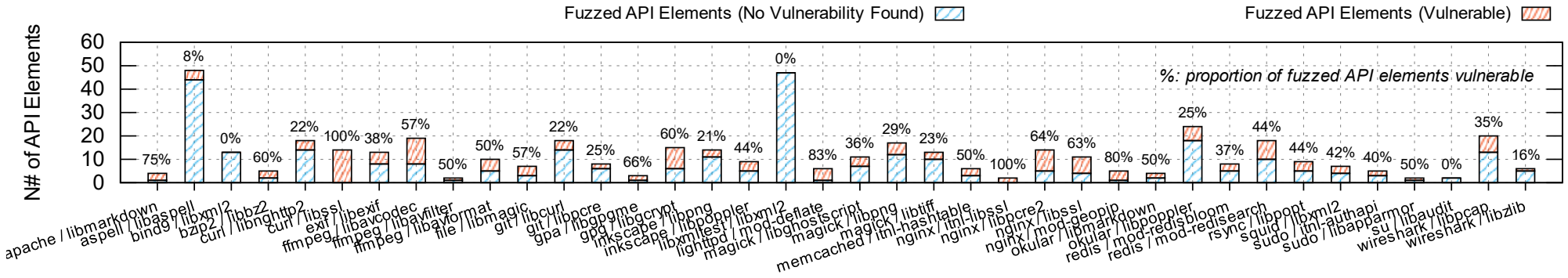


Number of vulnerable API endpoints (= has CIVs) versus non-vulnerable

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

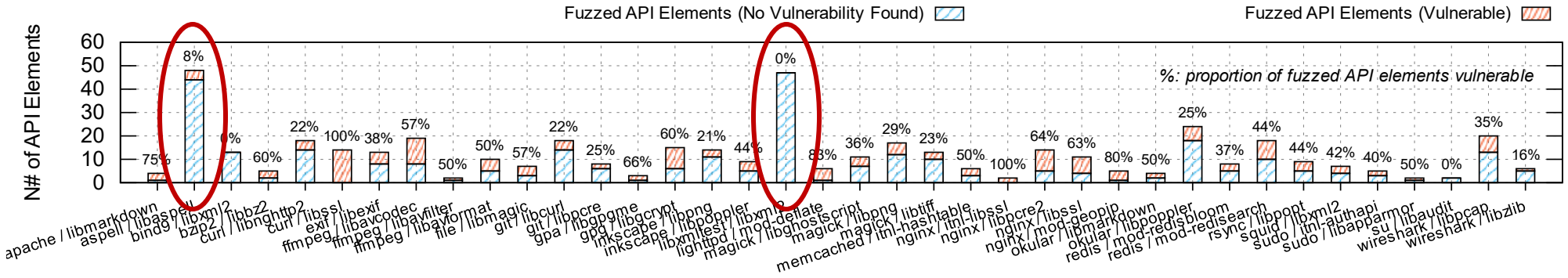
- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API



Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API

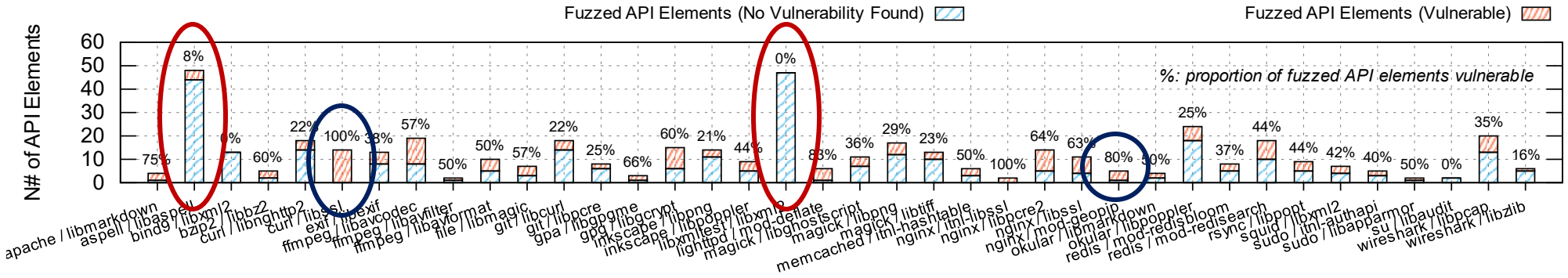


- There are large and (almost) totally CIV-free APIs

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API

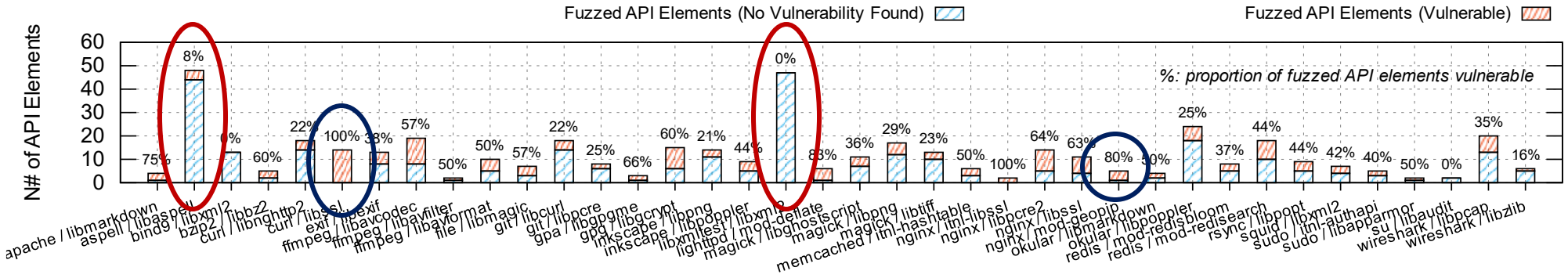


- There are large and (almost) totally CIV-free APIs
- There are small and fully vulnerable APIs

Study results: CIV prevalence

"How many CIVs are there at legacy APIs? Are all APIs similarly affected?"

- Confirms: CIVs are widespread among unmodified APIs/code
- But: there are **clear disparities among APIs**
  - CIV counts vary 0 – 105 CIVs for a single API



- There are large and (almost) totally CIV-free APIs
- There are small and fully vulnerable APIs

No correlation between API size and CIV count!

Study results: CIV Patterns

"Are all APIs similarly affected? How hard are they to fix?"

- We observe recurring API design patterns that lead to CIVs

Study results: CIV Patterns

"Are all APIs similarly affected? How hard are they to fix?"

- We observe recurring API design patterns that lead to CIVs
- These reinforce the idea that **the presence of CIVs is influenced by structural properties** rather than API size or quantity of shared data

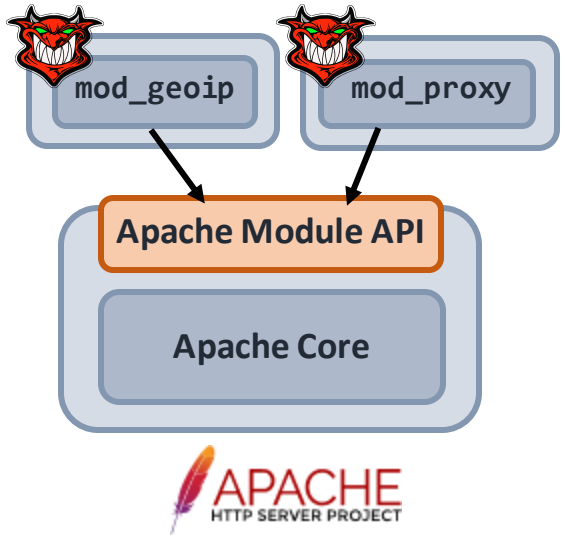
Study results: CIV Patterns

"Are all APIs similarly affected? How hard are they to fix?"

- We observe recurring API design patterns that lead to CIVs
- These reinforce the idea that **the presence of CIVs is influenced by structural properties** rather than API size or quantity of shared data
- Highlight one of these patterns here
- Many more in the paper!

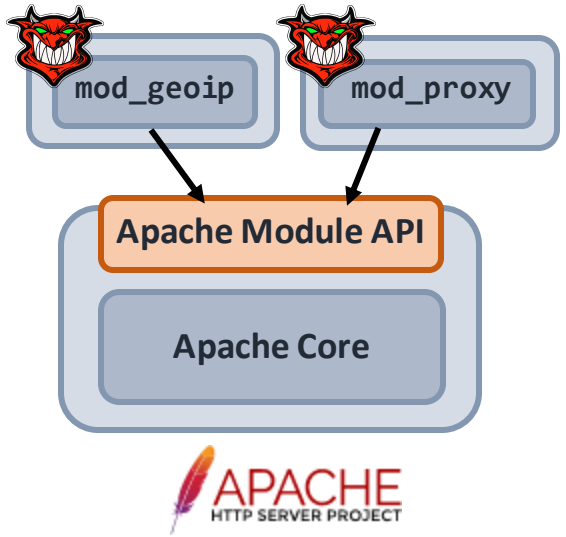
CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)



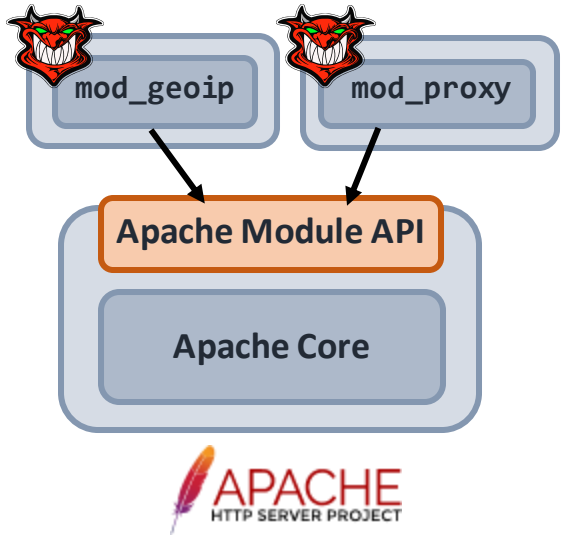
CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)
- **More CIVs** and **worse CIVs** on average



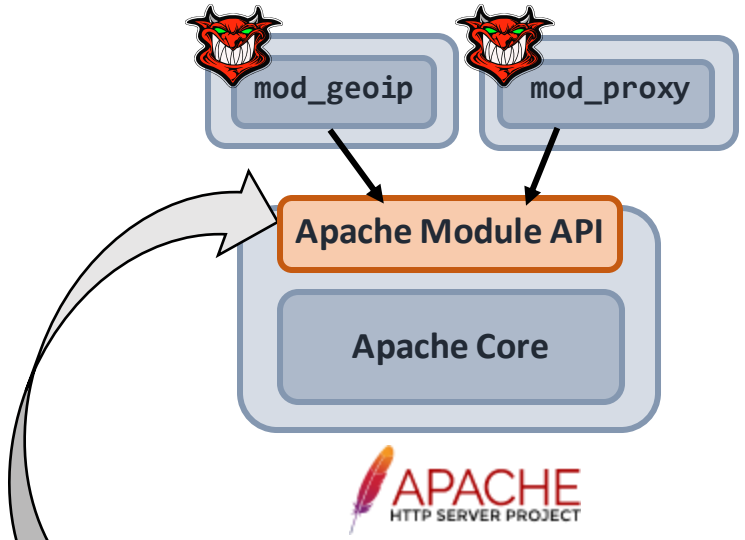
CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)
- **More CIVs** and **worse CIVs** on average
- Unlike library APIs, module APIs must be **very generic** and **yield high performance**
  - Consequence: the application's **core internal state** is **exposed to the module**



CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)
- **More CIVs** and **worse CIVs** on average
- Unlike library APIs, module APIs must be **very generic** and **yield high performance**
  - Consequence: the application's **core internal state** is **exposed to the module**

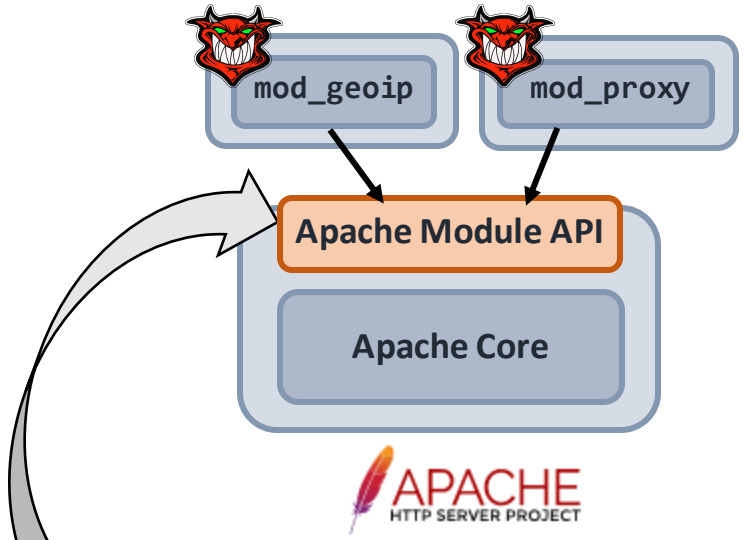


request structure crossing the interface

```
// A structure that represents the current request
struct request_rec {
    apr_pool_t *pool; // request's memory pool
    conn_rec *conn; // connection with the client
    server_rec *server; // request's virtual host
    request_rec *main; // main request pointer
    apr_thread_mutex_t *inv_mtx; // callback mutex
} // ... abbreviated
```

CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)
- **More CIVs** and **worse CIVs** on average
- Unlike library APIs, module APIs must be **very generic** and **yield high performance**
  - Consequence: the application's **core internal state** is **exposed to the module**



request structure crossing the interface

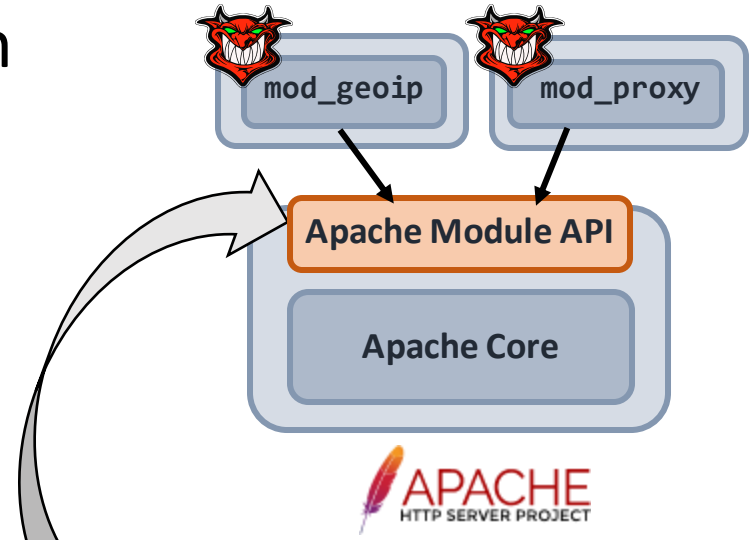
```
// A structure that represents the current request
struct request_rec {
    apr_pool_t *pool; // request's memory pool
    conn_rec *conn; // connection with the client
    server_rec *server; // request's virtual host
    request_rec *main; // main request pointer
    apr_thread_mutex_t *inv_mtx; // callback mutex
} // ... abbreviated
```

core internals

>75 fields, 60% of which pointers

CIV Pattern: Modular APIs

- Module APIs are the most CIV-vulnerable APIs in the study (HTTPd, Nginx, Redis, lighttpd, etc.)
- **More CIVs** and **worse CIVs** on average
- Unlike library APIs, module APIs must be **very generic** and **yield high performance**
  - Consequence: the application's **core internal state** is **exposed to the module**



request structure crossing the interface

```
// A structure that represents the current request
struct request_rec {
    apr_pool_t *pool; // request's memory pool
    conn_rec *conn; // connection with the client
    server_rec *server; // request's virtual host
    request_rec *main; // main request pointer
    apr_thread_mutex_t *inv_mtx; // callback mutex
} // ... abbreviated
```

core internals

>75 fields, 60% of which pointers

Somewhat counter-intuitive: modularity does not imply low compartmentalization complexity

Study results: Security Impact

- *"How bad are they?"*

Study results: Security Impact

- *"How bad are they?"*
- Confirms: **CIVs are high-impact**
  - >75% of scenarios present at least a write vulnerability

Study results: Security Impact

- *"How bad are they?"*
- Confirms: **CIVs are high-impact**
  - >75% of scenarios present at least a write vulnerability
  - ~70% of write and read, and ~50% of execute vulnerabilities are arbitrary

Study results: Security Impact

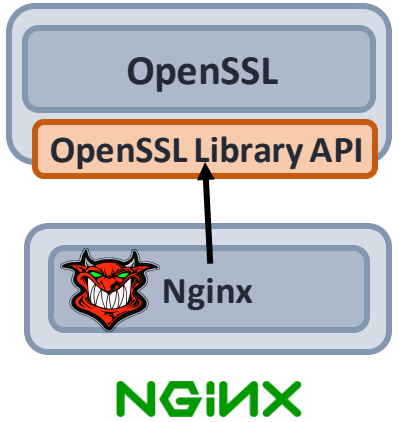
- *"How bad are they?"*
- Confirms: **CIVs are high-impact**
  - >75% of scenarios present at least a write vulnerability
  - ~70% of write and read, and ~50% of execute vulnerabilities are arbitrary
  - Only 8/39 scenarios have execution impact CIVs
    - but read and/or write primitives can likely be combined to achieve code execution

Study results: Security Impact

- *"How bad are they?"*
- Confirms: **CIVs are high-impact**
  - >75% of scenarios present at least a write vulnerability
  - ~70% of write and read, and ~50% of execute vulnerabilities are arbitrary
  - Only 8/39 scenarios have execution impact CIVs
    - but read and/or write primitives can likely be combined to achieve code execution
- Here: illustrate security impact with a concrete scenario
  - Key extraction in OpenSSL safebox
- More in the paper

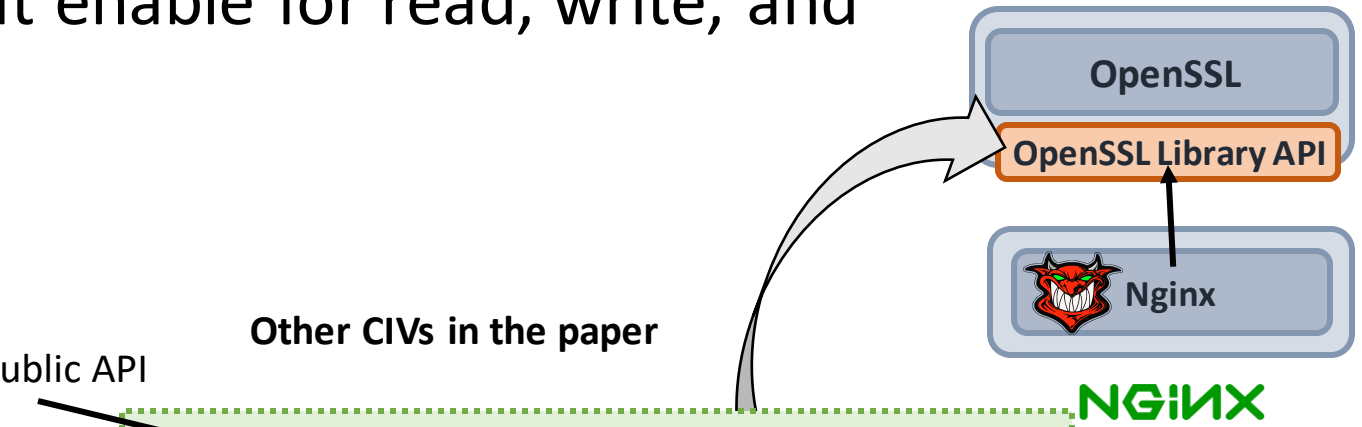
Security Impact: OpenSSL Key Extraction (1)

- Assume we isolate OpenSSL to protect SSL keys
  - for example, from a compromised Nginx
- The key compartment interface is the **OpenSSL public API**



Security Impact: OpenSSL Key Extraction (1)

- Assume we isolate OpenSSL to protect SSL keys
  - for example, from a compromised Nginx
- The key compartment interface is the **OpenSSL public API**
- We find several CIVs that enable for read, write, and execution impact



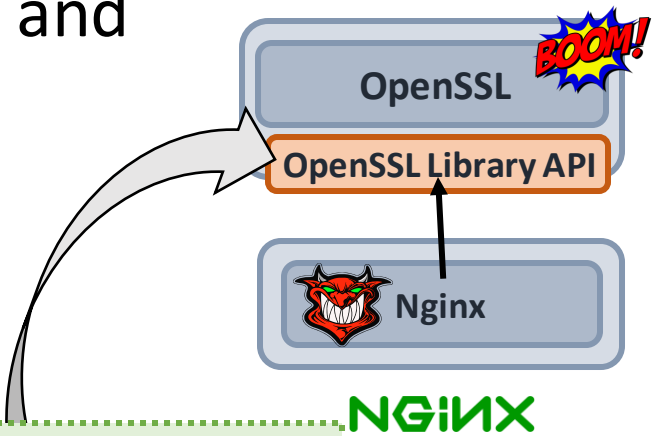
OpenSSL option setting primitive – part of the public API

Other CIVs in the paper

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

Security Impact: OpenSSL Key Extraction (1)

- Assume we isolate OpenSSL to protect SSL keys
  - for example, from a compromised Nginx
- The key compartment interface is the **OpenSSL public API**
- We find several CIVs that enable for read, write, and execution impact



OpenSSL option setting primitive – part of the public API

Dereference arbitrary pointer, set it, and return it:
arbitrary read and arbitrary write oracle

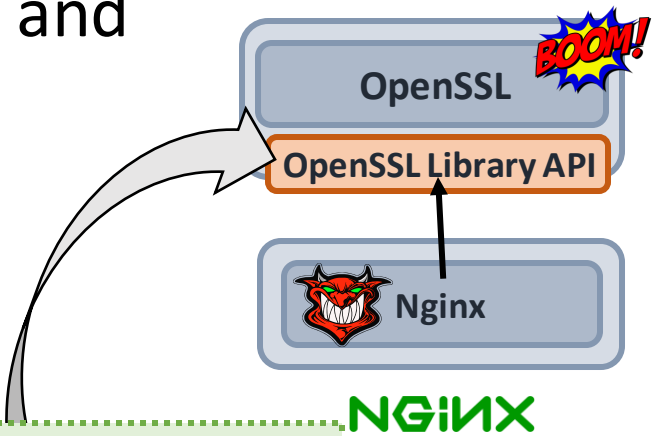
→ ***Key extracted***

Other CIVs in the paper

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

Security Impact: OpenSSL Key Extraction (1)

- Assume we isolate OpenSSL to protect SSL keys
  - for example, from a compromised Nginx
- The key compartment interface is the **OpenSSL public API**
- We find several CIVs that enable for read, write, and execution impact
- **Because of CIVs, isolating at the OpenSSL boundary is weak**



Other CIVs in the paper

OpenSSL option setting primitive – part of the public API

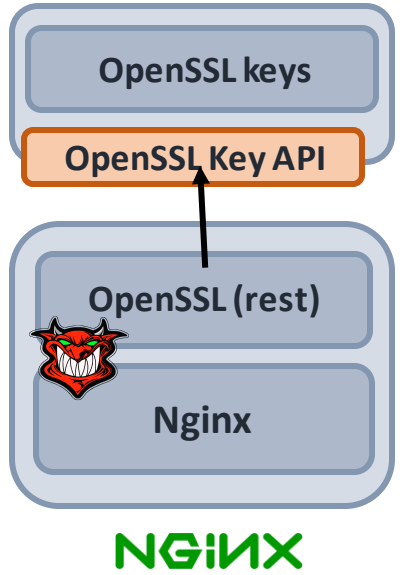
Dereference arbitrary pointer, set it, and return it:
arbitrary read and arbitrary write oracle

→ ***Key extracted***

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

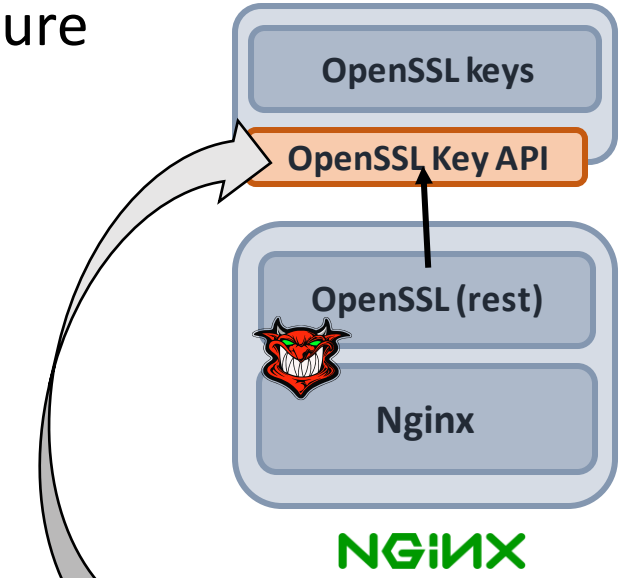
Security Impact: OpenSSL Key Extraction (2)

- Same scenario, but let's plug at a different interface
- This time, the **OpenSSL internal key API**
  - This is a very popular use-case / application in the literature



Security Impact: OpenSSL Key Extraction (2)

- Same scenario, but let's plug at a different interface
- This time, the **OpenSSL internal key API**
  - This is a very popular use-case / application in the literature
- Here too, we find key-extraction CIVs

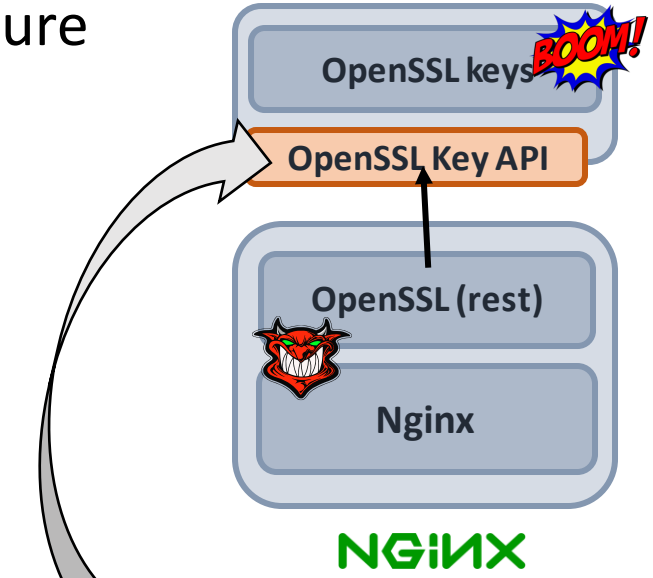


Prototype of one of the key API endpoints

```
void aesni_ecb_encrypt(const uchar *in, uchar *out,
size_t length, const AES_KEY *key, int enc);
```

Security Impact: OpenSSL Key Extraction (2)

- Same scenario, but let's plug at a different interface
- This time, the **OpenSSL internal key API**
  - This is a very popular use-case / application in the literature
- Here too, we find key-extraction CIVs



Prototype of one of the key API endpoints

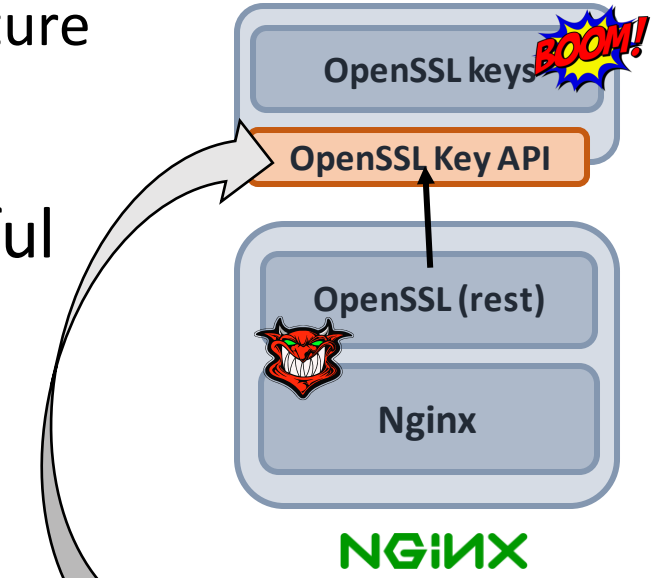
```
void aesni_ecb_encrypt(const uchar *in, uchar *out,
size_t length, const AES_KEY *key, int enc);
```

Point *in to the key, encrypt with known *key, then decrypt

→ ***Key extracted***

Security Impact: OpenSSL Key Extraction (2)

- Same scenario, but let's plug at a different interface
- This time, the **OpenSSL internal key API**
  - This is a very popular use-case / application in the literature
- Here too, we find key-extraction CIVs
- Fixing them requires to make the component stateful



Prototype of one of the key API endpoints

```
void aesni_ecb_encrypt(const uchar *in, uchar *out,
size_t length, const AES_KEY *key, int enc);
```

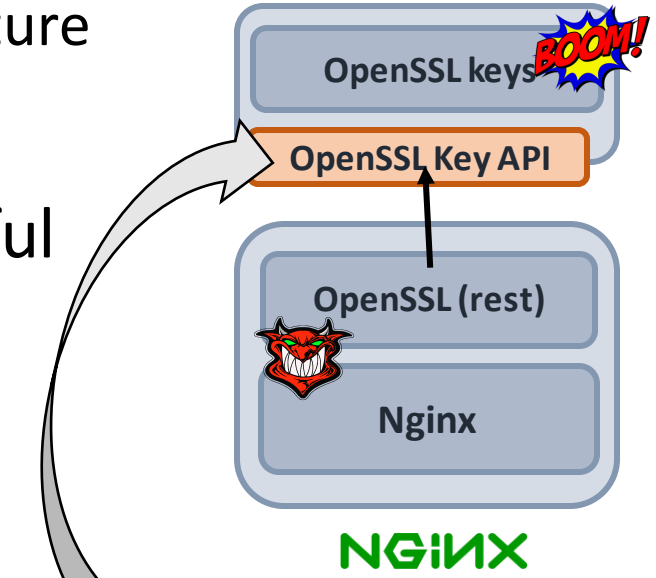
Point *in to the key, encrypt with known *key, then decrypt

→ ***Key extracted***

Security Impact: OpenSSL Key Extraction (2)

- Same scenario, but let's plug at a different interface
- This time, the **OpenSSL internal key API**
  - This is a very popular use-case / application in the literature
- Here too, we find key-extraction CIVs
- Fixing them requires to make the component stateful

Because of CIVs, all OpenSSL isolation use-cases from the literature are pointless & fixes are non-trivial



Prototype of one of the key API endpoints

```
void aesni_ecb_encrypt(const uchar *in, uchar *out,
size_t length, const AES_KEY *key, int enc);
```

Point *in to the key, encrypt with known *key, then decrypt

→ ***Key extracted***

What can we do about CIVs? (1)

Ways forward to tackle CIVs:

What can we do about CIVs? (1)

Ways forward to tackle CIVs:

1. Progress towards more **systematic, automatic CIV defenses**
   - we highlight limitations of existing defenses in the paper

What can we do about CIVs? (1)

Ways forward to tackle CIVs:

1. Progress towards more **systematic, automatic CIV defenses**
   - we highlight limitations of existing defenses in the paper
2. (Re-)Design interfaces to be **CIV-resilient by design**
   - we provide a set of guidelines to achieve this

What can we do about CIVs? (1)

Ways forward to tackle CIVs:

1. Progress towards more **systematic, automatic CIV defenses**
   - we highlight limitations of existing defenses in the paper
2. (Re-)Design interfaces to be **CIV-resilient by design**
   - we provide a set of guidelines to achieve this

Fundamentally hard: need to understand API semantics

Both approaches are complimentary

What can we do about CIVs? (2)

→ (Re-)Design interfaces to be CIV-resilient by design

Not enough time here to go over the guidelines, but broadly

What can we do about CIVs? (2)

→ (Re-)Design interfaces to be CIV-resilient by design

Not enough time here to go over the guidelines, but broadly

- **Not every interface is a good compartmentalization boundary**
  - choose your interface wisely

What can we do about CIVs? (2)

→ (Re-)Design interfaces to be CIV-resilient by design

Not enough time here to go over the guidelines, but broadly

- **Not every interface is a good compartmentalization boundary**
  - choose your interface wisely
- **Simplify** as much as possible interface-crossing objects
  - no system resource handles, not complex structs, synchronization primitives
  - if not possible, you should probably **plug at a different interface**

What can we do about CIVs? (2)

→ (Re-)Design interfaces to be CIV-resilient by design

Not enough time here to go over the guidelines, but broadly

- **Not every interface is a good compartmentalization boundary**
  - choose your interface wisely
- **Simplify** as much as possible interface-crossing objects
  - no system resource handles, not complex structs, synchronization primitives
  - if not possible, you should probably **plug at a different interface**
- **Enforce API semantics** (ordering, concurrency support)

Takeaways

Takeaways

- CIVs should be at the center of every compartmentalization approach

Takeaways

- CIVs should be at the center of every compartmentalization approach
- API design patterns influence CIV prevalence and severity
  - it's not so much about the size of the API
  - … it's about the complexity of API-crossing objects

Takeaways

- CIVs should be at the center of every compartmentalization approach
- API design patterns influence CIV prevalence and severity
  - it's not so much about the size of the API
  - … it's about the complexity of API-crossing objects
- Addressing CIVs requires more than a few checks
  - … strong solutions often require refactoring the API
  - automatic compartmentalization is harder that setting & enforcing bounds

Takeaways

- CIVs should be at the center of every compartmentalization approach
- API design patterns influence CIV prevalence and severity
  - it's not so much about the size of the API
  - … it's about the complexity of API-crossing objects
- Addressing CIVs requires more than a few checks
  - … strong solutions often require refactoring the API
  - automatic compartmentalization is harder that setting & enforcing bounds
- We need more research to address the problem of CIVs

Takeaways

- CIVs should be at the center of every compartmentalization approach
- API design patterns influence CIV prevalence and severity
  - it's not so much about the size of the API
  - … it's about the complexity of API-crossing objects
- Addressing CIVs requires more than a few checks
  - … strong solutions often require refactoring the API
  - automatic compartmentalization is harder that setting & enforcing bounds
- We need more research to address the problem of CIVs

NDSS'23 Paper: https://arxiv.org/abs/2212.12904
Project website: https://conffuzz.github.io Code & Dataset under BSD-3 & CC-BY