

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.





We closely study the root cause trends of vulnerabilities & search for patterns

% of memory safety vs. non-memory safety CVEs by patch year



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

RELATED

< • • • >



US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft Security

Memory Safety Still the Main Security Issue in Systems Software

Microsoft: 70 percent memory safety issues

Percentage of memory safety issues has been hovering

We closely study the root cause trends of vulnerabilities

% of memory safety vs. non-memory safety CVEs by part

Catalin Cimpanu for Zero Day



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.



Image: Google

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

RELATED

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

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

Percentage of memory safety issues has been hovering



National Security Agency | Cybersecurity Information Sheet

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

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



Apple: Side-loading on iOS would

RELATED



Security Nork to earn several highly



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



US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft Security



ZDNet Security

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

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

- 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

- 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.

- 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

Motivation & Presentation

 Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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

	HTTP Parser	
Crypto		
Library		

Kilpatrick, Douglas. "Privman: A Library for Partitioning Applications." In USENIX Annual Technical Conference, FREENIX Track,

pp. 273-284. 2003.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

Motivation & Presentation

- Software compartmentalization decompose software into lesserprivileged 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.

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

- 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

- 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**

- 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

	Compartment 1 (crypto library)	Compartment 2 (HTTP parser)	
Crypto keys	Read access	No access	Lampson's access control matrix
HTTP request data	No access	Read access	Lampson, Butler W. (1971). "Protection". Proceedings of the 5th Princeton Conference on Information Sciences and Systems. p. 437.

- 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	Lampson's access control matrix
HTTP request data	No access	Read access	Lampson, Butler W. (1971). "Protection". Proceedings of the 5th Princeton Conference on Information Sciences and Systems. p. 437.

• 3 trust models:

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





- 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





• 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



- 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



• 3 security properties considered:

• 3 security properties considered:



• **Confidentiality:** an attacker cannot read/leak information from outside of a subverted compartment

- 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



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


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;
```

Illustrative Example

int global;

return 0:

```
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);
     /* ... */
```

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

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

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.

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



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



Security domain switch comp2 → comp1

Software Compartmentalization It's in the Air!



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)





Broad Agency Announcement Compartmentalization and Privilege Management (CPM) INFORMATION INNOVATION OFFICE HR001123S0028 April 4, 2023

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

- 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



- 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





- 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







- 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

```
double data[DATA_SIZE];
```

/* ... */

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

```
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);</pre>
```

/* ... */

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)

```
double data[DATA_SIZE];
```

```
/* ... */
```

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

```
/* ... */
```

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

/* ... */

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

```
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);</pre>
```

/* ... */

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

```
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);</pre>
```

/* ... */

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

```
double data[DATA_SIZE];
```

```
/* ... */
```

```
int library function(int index, double object) {
     if (index >= DATA_SIZE || index < 0)</pre>
           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)

- 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)

- 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
• E • E c	xposure of addresses xposure of compartment- onfidential 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)

- 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?

Malicious compartment (e.g. compromised library)

ConfFuzz

Victim compartment (e.g. main app code)

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

Malicious compartment (e.g. compromised library)

ConfFuzz

Victim compartment (e.g. main app code)

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

Malicious compartment (e.g. compromised library)

ConfFuzz

Victim compartment (e.g. main app code)

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



- 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



Methodology:

1) Choose an app



Methodology:

 Choose an app
 Choose a meaningful potential compartmentalization interface



Methodology: 1) Choose an app

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



- 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

Year Openation Comparison Rare Wear Callers Coverage Read Write Exce Alloc Null March Humatkown [42] 38 3 1 100% (4/4) 10 (8) 7(7) 0(0) 1 4 application Ibbasyndl March 218 8 1 134% (48/41) 7(7) 7(7) 7(7) 2(1) 0 3 hud9 Ibbsvndl ovite API 0 0 0 1 84% (43/15) 0(0) 0(0) 0(0) 0<	TM	Application	Composition ADI	Defenences	Cr	ashes	Vietime	API	Coverage	I	mpact (of w	hich arb		
NTTPd Himmarkdown [42] 192 13 3 1 100% (4/4) 10 (8) 7 (7) 0 (0) 1 4 spell Himarkdown 278 8 1 1 100% (4/4) 7 (7) 7 (7) 2 (1) 0 30 bipd Hibargl2 write API) 0 0 0 1 86% (137) 7 (7) 7 (7) 2 (1) 0 30 cvif Hibbargl2 [67], [5] 16 5 1 1 100% (147) 3(5) 0 (0) 10 0<	INI	Application	Compartment AP1	Kelerences	Raw	Dedup.	vicums	Callers	Coverage	Read	Write	Exec	Alloc	Null
Normal mad_markdown image mad_markdown image mad_markdown image made			libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
spell Inbaspell mod 278 8 1 1 34% (48/14) 7(7) 7(7) 2(1) 0 3 bind libbn2 (67) (5) 16 5 1 1 84% (48/14) 7(7) 7(7) 2(1) 0 <t< td=""><td></td><td>HIIPa</td><td>mod_markdown</td><td></td><td>381</td><td>71</td><td>5</td><td>1</td><td>100% (1/1)</td><td>62 (52)</td><td>17 (14)</td><td>2 (1)</td><td>0</td><td>30</td></t<>		HIIPa	mod_markdown		381	71	5	1	100% (1/1)	62 (52)	17 (14)	2 (1)	0	30
Find Ibbm2 (write AP) 0		aspell	libaspell		278	8	1	1	34% (48/141)	7 (7)	7 (7)	2 (1)	0	3
Vertice bind (bind) (cind) (cind) </td <td>3</td> <td>bind9</td> <td>libxml2 (write API)</td> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>86% (13/15)</td> <td>0 (0)</td> <td>0 (0)</td> <td>0 (0)</td> <td>0</td> <td>0</td>	3	bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
Verticity Liborghtp2 Control 61 7 2 1 50% (13729) 3(3) 5(5) 0(0) 1 3 Verticity Liborodec 316 20 3 4 31% (1960) 13 (12) 12 (12) 0(0) 0 5 7 Fibre Liborodec 51 1 1 2 12% (1960) 13 (12) 12 (1) 0(0) 0 7 7 Ibbrog Liboromat 217 9 2 3 5% (1019) 8 (7) 1 (1) 0 (0) 0 7 7 Ibbrog Libor 121 3 4 2 1 0 (3) 4 (4) 0 (0) 0 0 1 100% (1011) 10 (0) 0 (0) 0 0 0 1 100% (1010) 0 (0) 0 (0) 0 0 0 1 100% (47) 0 (0) 0 (0) 0 0 0 0 0 1 1 100% (47)	3	bzip2	libbz2	[67], [5]	16	5	1	1	62% (5/8)	5 (2)	1 (0)	0 (0)	0	0
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		cURL	libnghttp2		61	7	2	1	50% (18/36)	3 (3)	5 (5)	0 (0)	1	3
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		exif	libexif		400	7	1	1	10% (13/129)	3 (3)	0 (0)	0 (0)	0	5
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	8		libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		FFmpeg	libavfilter		51	1	1	2	12% (2/16)	1 (1)	0 (0)	0 (0)	0	1
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$			libavformat		217	9	2	3	52% (10/19)	8 (7)	1 (1)	0 (0)	0	7
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	3	file	libmagic		150	5	1	1	63% (7/11)	5 (2)	1 (1)	0 (0)	0	4
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	3		libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
Inkscape libpopler [16] 66 3 1 1 46% (14/30) 2 (1) 2 (2) 0 (0) 0 1 libxml2-tests likxml2 (write API) 0 0 0 1 100% (9/9) 4 (3) 4 (4) 0 (0) 0 2 lightpd mod_deflate 117 26 2 1 100% (6/6) 16 (1) 5 (0) 1 (1) 2 9 mage libpostscript [5] 67 14 2 1 (0)% (6/6) 16 (1) 5 (0) 1 (1) 2 (9) 2 (0) 2 39 Magick libpore [67] 778 44 1 2 22% (17/7) 2 (2) 9 (9) 2 (0) 2 39 Nginx libpore 1144 10 1 93% (1/13) 3 (3) 0 (0) 6 (2) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1) 1 (1)<		git	libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
Bitscape libscape		Inkoono	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
Bioxml2-tests libxml2 (write API) 0 0 0 1 100% (47/47) 0 (0) <td>xoc</td> <td>inkscape</td> <td>libpoppler</td> <td>[16]</td> <td>81</td> <td>4</td> <td>2</td> <td>1</td> <td>100% (9/9)</td> <td>4 (3)</td> <td>4 (4)</td> <td>0 (0)</td> <td>0</td> <td>2</td>	xoc	inkscape	libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	lpu	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	Sai	lighttpd	mod_deflate	100000	117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		Image	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$			libpng	[67]	778	44	1	2	22% (17/77)	2 (2)	9 (9)	2 (0)	2	39
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		Magick	libtiff	[67]	197	14	2	1	30% (13/43)	3 (3)	6 (6)	0 (0)	0	13
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	9	Nginx	libpcre		144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$			mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4 (1)	1 (1)	1	10
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		Olaylan	libmarkdown	[42]	64	5	3	1	100% (4/4)	3 (1)	0 (0)	0 (0)	1	2
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$		Okular	libpoppler	[16]	195	9	1	1	6% (24/379)	8 (6)	7 (7)	0 (0)	1	4
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	1	Dadia	mod_redisbloom		389	23	1	1	42% (8/19)	18 (13)	6 (4)	0 (0)	0	13
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $		Reuis	mod_redisearch		381	21	1	1	54% (18/33)	15 (14)	14 (11)	0 (0)	0	12
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $		rsync	libpopt		167	8	1	1	90% (9/10)	4 (3)	2 (0)	0 (0)	0	5
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	1	squid	libxml2		226	12	1	1	70% (7/10)	9 (5)	3 (3)	4 (1)	0	4
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 CURL libssl [5] 198 27 1 1 25% (14/56) 18 (10) 5 (4) 1 (1) 0 17 GPA libggme 174 9 1 1 4% (3/72) 7 (2) 0 (0) 0 (0) 0 6 GPG libgrame [45] 4037 16 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 1 6 Memcached internal_hashtable [45], [60], [15], [34] 599 46 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 0 2 2	1	su	libaudit		0	0	0	1	66% (2/3)	0 (0)	0 (0)	0 (0)	0	0
Witeshark libzlib 42 1 1 1 85% (6/7) 0 (0		Winashork	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4
Total: 5508 379 47 38 N/A 246 (192) 124 (105) 12 (5) 24 195 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 GPA 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) 0 22 Nginx internal_libssl-keys [45], [60], [15], [34] 599 46 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 0 22 Ibssl [5], [1], [22], [51] 346 39 2 1 11% (11/96) 16 (13) 19 (13)		wireshark	libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1
cURL libsl [5] 198 27 1 1 25% (14/56) 18 (10) 5 (4) 1 (1) 0 17 GPA libgpme 174 9 1 1 4% (3/72) 7 (2) 0 (0) 0 (0) 0 6 GPA libgrame [5] 4221 105 1 1 4% (3/72) 7 (2) 0 (0) 0 (0) 0 6 GPG libgrame [45] 4037 16 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 1 6 Memcached internal_libssl-keys [45], [60], [15], [34] 599 46 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 0 22 Nginx internal_alibssl-keys [45], [60], [15], [34] 599 46 1 1 50% (6/12) 10 (5) 2 (0) 0 (0) 0 22 sudo internal_auth-api [5], [1], [22], [51] 346 39 2 1 <td></td> <td>Total:</td> <td></td> <td></td> <td>5508</td> <td>379</td> <td>47</td> <td>38</td> <td>N/A</td> <td>246 (192)</td> <td>124 (105)</td> <td>12 (5)</td> <td>24</td> <td>195</td>		Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195
GPA libggme 174 9 1 1 4% (3/72) 7 (2) 0 (0) 0 (0) 0 6 GPG libgcrypt [5] 421 105 1 1 5% (5/75) 64 (60) 4 (0) 0 (0) 0 1 1 </td <td></td> <td>cURL</td> <td>libssl</td> <td>[5]</td> <td>198</td> <td>27</td> <td>1</td> <td>1</td> <td>25% (14/56)</td> <td>18 (10)</td> <td>5 (4)</td> <td>1 (1)</td> <td>0</td> <td>17</td>		cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $		GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6
$ \frac{Memcached}{Nginx} = \frac{internal_hashtable}{libssl-keys} = \frac{[45]}{[45]} = \frac{4037}{16} = \frac{1}{16} = \frac{1}{16} = \frac{50\%}{60/2} = \frac{100\%}{10} =$	x	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
Nginx internal_libssl-keys [45], [60], [15], [34] 599 46 1 1 50% (2/4) 32 (1) 28 (0) 0 (0) 0 22 Ibssl [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 4 ibapparmor 97 3 1 1 100% (2/2) 2 (2) 2 (0) 0 (0) 0	po.	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
INGINX Iibssi [5], [1], [22], [51] 346 39 2 1 11% (11/96) 16 (13) 19 (13) 2 (1) 0 26 sudo internal_auth-api [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 <t< td=""><td>afe</td><td>Malan</td><td>internal_libssl-keys</td><td>[45], [60], [15], [34]</td><td>599</td><td>46</td><td>1</td><td>1</td><td>50% (2/4)</td><td>32 (1)</td><td>28 (0)</td><td>0 (0)</td><td>0</td><td>22</td></t<>	afe	Malan	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
sudo internal_auth-api 191 5 1 1 100% (5/5) 5 (4) 0 (0) 0 (0) 0 4 sudo libapparmor 97 3 1 1 100% (5/5) 5 (4) 0 (0) 0 (0) 0 4 Total: 9863 250 9 8 N/A 154 (97) 60 (17) 3 (2) 78 100	S	Nginx	libssl	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2 (1)	0	26
sudo libapparmor 97 3 1 1 100% (2/2) 2 (2) 2 (0) 0 (0) 0 <th0< th=""> 0</th0<>	8	and a	internal_auth-api		191	5	1	1	100% (5/5)	5 (4)	0 (0)	0 (0)	0	4
Total: 9863 250 9 8 N/A 154 (97) 60 (17) 3 (2) 78		sudo	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	M

	TM	Application	Comportment A DI	Deferences	Cr	ashes	Victime	API	Coverage	Impact (of which arbitrary)				
	IN	Application	Compartment API	Kelerences	Raw	Dedup.	vicums	Callers	Coverage	Read	Write	Exec	Alloc	Null
5 applications		UTTDA	libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4
S applications $\overline{\ }$		HIIPa	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)	1000000000	0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0
o Apis in total 🥄		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
			libavcodec		316	20	3	4	31% (19/60)	13 (12)	12 (12)	0 (0)	3	7
	, · ·	FFmpeg	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
		ait	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	2 (2)	0 (0)	1	1
		gn	libpcre		81	2	1	1	44% (8/18)	2 (2)	0 (0)	0 (0)	2	0
	2	Inkscape	libpng	[67]	66	3	1	1	46% (14/30)	2 (1)	2 (2)	0 (0)	0	1
	coq	museupe	libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2
	pu	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0
	Sa	lighttpd	mod_deflate	1200	117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9
		Image	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9
		Magick	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		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
		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
	X	GPG	libgcrypt	[5]	4221	105	1	1	15% (15/95)	64 (60)	4 (0)	0 (0)	77	20
	ebc	Memcached	internal_hashtable	[45]	4037	16	1	1	50% (6/12)	10 (5)	2 (0)	0 (0)	1	6
	Saf	Nginy	internal_libssl-keys	[45], [60], [15], [34]	599	46	1	1	50% (2/4)	32 (1)	28 (0)	0 (0)	0	22
	~ 1	right	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
	1	Juno	libapparmor		97	3	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)	0	1A

Library APIs

Module APIs **Internal APIs**

Compartment API

libmarkdown

Application

TM

25 applications

36 APIs in total 、

7 (7) [42] 13 100% (4/4) 192 3 10 (8) 0(0)HTTPd 17 (14) mod markdown 381 71 5 100% (1/1) 62 (52) 2(1)0 30 278 7 (7) aspell libaspell 8 34% (48/141) 7(7) 2(1)0 3 libxml2 (write API) bind9 0 0 86% (13/15) 0(0)0 0(0)0(0)0 0 1 bzip2 [67], [5] 1 (0) libbz2 16 5 1 1 62% (5/8) 5(2)0(0)0 0 cURL 61 7 2 50% (18/36) 3 (3) 5 (5) 1 0(0)1 3 400 exif 10% (13/129) 3 (3) 0 (0) м libexif 7 1 0(0)0 5 1 31% (19/60) 13 (12) 12 (12) libaycodec 316 20 3 0(0)3 7 4 FFmpeg libavfilter 51 1 1 2 12% (2/16) 1(1)0(0)0(0)0 1 52% (10/19) 217 8 (7) 1(1)9 2 3 0(0)0 150 63% (7/11) libmagic 5 5(2)1(1)0(0)file 0 4 [22] 2 (2) 2 (2) libcurl 13 4 2 90% (18/20) 0(0)1 1 1 git libpcre 81 2 44% (8/18) 2(2)0(0)1 0(0)2 0 1 [67] 2 (1) 2 (2) 66 3 1 46% (14/30) 0(0)0 1 1 Inkscape Sandbox 81 libpoppler [16] 4 2 100% (9/9) 4(3)4(4)0(0)0 2 libxml2 (write API) 0 0 0 100% (47/47) 0(0)0(0)libxml2-tests 1 0(0)0 0 117 26 2 100% (6/6) 5 (0) lighttpd mod deflate 16 (11) 2 9 1 1(1)[5] 67 14 2 100% (11/11) 4(2)1(1)9 1 0(0)3 Image libpng [67] 778 44 1 22% (17/77) 2(2)9 (9) 2(0)2 39 Magick 30% (13/43) [67] 197 14 2 3 (3) 6 (6) 0(0)13 1 0 144 3 (3) 10 93% (14/15) 8 (7) libpcre 0(0)6 2 Nginx [52] 35% (5/14) mod geoip 276 25 2 1 21 (17) 4(1)1(1)1 10 [42] 100% (4/4) 0(0)libmarkdown 64 5 3 3(1)0(0)1 2 Okular libpoppler 195 6% (24/379) 7 (7) [16] 9 1 8 (6) 0(0)1 4 23 389 42% (8/19) 6 (4) mod redisbloom 18 (13) 0(0)0 13 Redis mod redisearch 381 21 54% (18/33) 15 (14) 14 (11) 12 1 0(0)0 libpopt 167 8 90% (9/10) 4 (3) 2(0)5 1 0(0)0 rsync 1 70% (7/10) 226 12 9 (5) 3 (3) squid 1 1 4(1)0 4 0 0 0 66% (2/3) 0(0)0(0)0(0)0 0 SIL 50% (20/40) 5 (5) libpcap 162 8 2 8 (3) 0(0)0 4 1 Wireshark 42 85% (6/7) 0(0)0(0)0(0)0 1 1 1 379 47 38 N/A 246 (192) Total: 5508 124 (105) 12 (5) 24 195 libssl [5] 25% (14/56) 18 (10) 27 5(4)17 cURL 198 1 1 1(1)0 libgpgme 9 **GPA** 174 1 1 4% (3/72) 7(2)0(0)0(0)0 6 GPG 4221 15% (15/95) 64 (60) 4(0)libgcrypt [5] 105 0(0)77 20 1 Safebox [45] 50% (6/12) Memcached internal hashtable 4037 16 10(5)2(0)0(0)1 6 1 1 [45], [60], [15], [34] 50% (2/4) 32(1)28 (0) internal libssl-keys 599 46 0(0)0 22 Nginx [5], [1], [22], [51] 346 39 2 11% (11/96) 16 (13) 19 (13) 2(1)0 26 1 100% (5/5) 5 (4) 0(0)internal auth-api 191 5 1 1 0(0)0 4 sudo 2 (2) 2 (0) 97 3 100% (2/2) 0 (0) libapparmor 1 1 0 250 9 8 N/A 154 (97) 3 (2) 78 Total: 9863 60 (17)

Crashes

Raw

Dedup.

Victims

References

API Coverage

Coverage

Read

Callers

Impact (of which arbitrary)

Exec

Write

Alloc

1

Null

4
Module APIs Internal APIs Library APIs Impact (of which arbitrary) Crashes **API** Coverage Application **Compartment API** TM References Victims Dedup. Null Raw Callers Coverage Read Write Exec Alloc [42] 192 13 100% (4/4) 7 (7) libmarkdown 3 10 (8) 0(0)4 25 applications 1 HTTPd 17 (14) mod markdown 381 71 5 100% (1/1) 62 (52) 2(1)0 30 278 7 (7) aspell libaspell 8 34% (48/141) 7(7) 2(1)0 3 bind9 libxml2 (write API) 0 86% (13/15) 0(0)0 0 0(0)0(0)0 0 36 APIs in total bzip2 libbz2 [67], [5] 16 5 62% (5/8) 5(2)1(0)0(0)0 0 cURL 61 7 2 50% (18/36) 3 (3) 5 (5) 1 0(0)1 3 exif 400 10% (13/129) 3 (3) 0(0)X libexif 1 0(0)0 5 1 316 libaycodec 20 3 31% (19/60) 13 (12) 12(12)0(0)3 4 7 FFmpeg libavfilter 51 1 1 2 12% (2/16) 1(1)0(0)0(0)0 1 217 52% (10/19) 8 (7) 1(1)9 2 3 0(0)0 150 63% (7/11) libmagic 5 5(2)1(1)0(0)file 0 4 [22] 2 (2) 2 (2) libcurl 13 4 2 90% (18/20) 0(0)1 1 1 git libpcre 81 2 44% (8/18) 2 (2) 0(0)1 0(0)2 0 16 of which [67] 2 (1) 2 (2) 66 3 46% (14/30) 0(0)1 1 0 1 Inkscape bpoppler [16] 81 4 2 100% (9/9) 4(3)4(4)0(0)0 2 taken from the libxml2-tests libxml2 (write API) 0 0 0 100% (47/47) 0(0)1 0(0)0(0)0 0 San 26 2 mod deflate 117 100% (6/6) 16 (11) 5(0)2 9 lighttpd 1 1(1)67 14 2 100% (11/11) 9 [5] 1 4(2)1(1)0(0)3 literature Image [67] 778 44 22% (17/77) 2(2)9 (9) 2(0)2 39 Т Magick [67] 197 14 2 30% (13/43) 3 (3) 6 (6) 0(0)13 1 0 144 10 93% (14/15) 8 (7) 3 (3) libpcre 0(0)2 6 Nginx [52] 276 mod geoip 25 2 35% (5/14) 21 (17) 4(1)1(1)1 10 [42] libmarkdown 64 5 3 100% (4/4) 3(1)0(0)0(0)1 2 Okular libpoppler 195 6% (24/379) 7 (7) [16] 9 8 (6) 0(0)1 4 389 23 42% (8/19) mod redisbloom 18 (13) 6(4)0(0)0 13 Redis mod redisearch 381 21 1 0(0)0



Impact (of which arbitrary) Crashes **API** Coverage Application **Compartment API** TM References Victims Dedup. Null Raw Callers Coverage Read Write Exec Alloc [42] 192 13 100% (4/4) 7(7) libmarkdown 3 10 (8) 0(0)4 25 applications HTTPd 17 (14) mod markdown 381 71 5 100% (1/1) 62 (52) 2(1)0 30 278 7 (7) aspell libaspell 8 34% (48/141) 7(7) 2(1)0 3 bind9 libxml2 (write API) 0 86% (13/15) 0(0)0 0 0(0)0(0)0 0 36 APIs in total bzip2 libbz2 [67], [5] 16 5 62% (5/8) 5(2)1(0)0(0)0 0 cURL 61 7 2 50% (18/36) 3 (3) 5 (5) 1 0(0)1 3 exif 400 10% (13/129) 3 (3) 0(0)M libexif 0(0)0 5 1 1 316 31% (19/60) libaycodec 20 3 13 (12) 12(12)0(0)3 4 7 FFmpeg libavfilter 51 1 2 12% (2/16) 1(1)0(0)0(0)0 1 1 217 52% (10/19) 8 (7) 1(1)9 2 3 0(0)0 150 63% (7/11) libmagic 5 5(2)1(1)0(0)file 0 4 [22] 2 (2) 2 (2) libcurl 13 4 2 90% (18/20) 0(0)1 1 1 git libpcre 81 2 44% (8/18) 2 (2) 0(0)1 0(0)2 0 16 of which [67] 2 (2) 66 3 46% (14/30) 2(1)0(0)1 1 0 1 Inkscape bpoppler [16] 81 4 2 100% (9/9) 4(3)4(4)0(0)0 2 taken from the libxml2-tests libxml2 (write API) 0 0 0 100% (47/47) 0(0)0(0)1 0(0)0 0 San mod_deflate 117 26 2 100% (6/6) 16 (11) 5(0)2 9 lighttpd 1 1(1)67 14 2 100% (11/11) 9 [5] 1 4(2)1(1)0(0)3 literature Image 778 [67] 44 22% (17/77) 2(2)9 (9) 2(0)2 39 Т Magick [67] 197 14 2 30% (13/43) 3 (3) 6 (6) 0(0)13 1 0 144 10 93% (14/15) 8 (7) 3 (3) libpcre 0(0)2 6 Nginx [52] 276 35% (5/14) mod geoip 25 2 1 21 (17) 4(1)1(1)1 10 [42] libmarkdown 64 5 3 100% (4/4) 3(1)0(0)0(0)1 2 Okular libpoppler 195 6% (24/379) 7 (7) [16] 9 8 (6) 0(0)1 4 23 42% (8/19) 6 (4) mod redisbloom 389 18 (13) 0(0)0 13 Redis mod redisearch 21 54% (18/33) 15 (14) 14 (11) 12 1 0(0)0 38 libpopt 167 8 90% (9/10) 2(0)4(3)0(0)0 5 rsync 1 70% (7/10) 226 12 9 (5) 3 (3) squid 1 1 4(1)0 4 Found 629 0 0 0 66% (2/3) 0(0)0(0)0(0)0 0 su ibaudit 50% (20/40) 5 (5) libpcap 162 8 2 8 (3) 0(0)0 4 1 Wireshark unique 42 85% (6/7) 0(0)0(0)0(0)0 1 1 379 47 Total: 5508 38 N/A 246 (192) 124 (105) 12 (5) 24 195 CIVs libssl [5] 25% (14/56) 18 (10) 27 5(4)17 cURL 198 1 1 1(1)0 libgpgme **GPA** 174 9 1 1 4% (3/72) 7(2)0(0)0(0)0 6 15% (15/95) 64 (60) 4(0)GPG libgcrypt [5] 4221 105 0(0)77 20 1 Safebox [45] 50% (6/12) Memcached internal hashtable 4037 16 10(5)2(0)0(0)1 6 1 1 50% (2/4) [45], [60], [15], [34] 32(1)28 (0) internal libssl-keys 599 46 0(0)0 22 Nginx [5], [1], [22], [51] 346 39 2 11% (11/96) 16 (13) 19 (13) 2(1)0 26 1 100% (5/5) 5 (4) internal auth-api 191 5 1 0(0)0(0)0 1 4 sudo 97 3 100% (2/2) 2(2)2(0)0(0)libapparmor 1 1 0 154 (97) 9863 250 9 N/A 60 (17) 78 Total: 8 3 (2)

Module APIs

Library APIs

Internal APIs

Library APIs Modu	le API	Pls Internal APIs 5 security impact															
											types						
	TM	Application	Compartment API	References	Crashes Raw Dedup		Victims	API Coverage		In Read	Impact (of which arbitrary)			Null			
25 applications			libmarkdown	[42]	192	13	3	1	100% (4/4)	10 (8)	7 (7)	0 (0)	1	4			
25 applications \sim		HTTPd	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			
26 ADIs in total		bind9	libxml2 (write API)		0	0	0	1	86% (13/15)	0 (0)	0 (0)	0 (0)	0	0			
		bzip2	libbz2	[67], [5]	Crashes Raw Dedi [42] 192 13 381 71 278 8 0 0 77] 16 5 61 7 400 7 400 7 16 5 17] 16 51 1 217 9 150 5 [22] 13 81 2 [67] 66 3[16] 81 0 0 117 26 [5] 67 [67] 778 144 10 [52] 276 [42] 64 5 16] 195 9 389 23 387 21 167 8 226 12 0 0 162 8	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	20	1		10% (13/129)	3 (3)	0 (0)	0 (0)	0	5			
		FEmnor	libayfilter		510	20	3	4	12% (2/16)	15 (12)	12(12)		5	1			
	<	Trinpeg	libayformat		217	9	2	2	52% (10/19)	8 (7)			0	7			
		file	libmagic		150	5	1	1	63% (7/11)	5(2)	1(1)	0 (0)	0	4			
16 of which	8	git	libcurl	[22]	13	4	2	1	90% (18/20)	2 (2)	$\frac{1}{2}(2)$	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			
	XOC		libpoppler	[16]	81	4	2	1	100% (9/9)	4 (3)	4 (4)	0 (0)	0	2			
akan from the 🛛 🗕	ndt	libxml2-tests	libxml2 (write API)		0	0	0	1	100% (47/47)	0 (0)	0 (0)	0 (0)	0	0			
	Sa	lighttpd	mod_deflate	1.000	117	26	2	1	100% (6/6)	16 (11)	5 (0)	1 (1)	2	9			
toraturo		Image	libghostscript	[5]	67	14	2	1	100% (11/11)	4 (2)	1 (1)	0 (0)	3	9			
terature		Magick	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	(70)	144	10	1	1	93% (14/15)	8 (7)	3 (3)	0 (0)	6	2			
		0	mod_geoip	[52]	276	25	2	1	35% (5/14)	21 (17)	4(1)	1(1)	1	10			
		Okular	libmarkdown	[42]	04	5	3	1	100% (4/4)	3(1)	0(0)	0 (0)	1	2			
		Redis	mod redisbloom	[10]	380	23	1	1	0% (24/3/9)	8 (0)	(1)		1	4			
			mod_redisearch		302	23	1	1	54% (18/33)	15 (13)	14 (11)	0 (0)	0	13			
		rsync	libnont		167	8	1	1	90% (9/10)	4(3)	2 (0)	0 (0)	0	5			
Found 629		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			
		Winnehead	libpcap		162	8	2	1	50% (20/40)	8 (3)	5 (5)	0 (0)	0	4			
		wiresnark	libzlib		42	1	1	1	85% (6/7)	0 (0)	0 (0)	0 (0)	0	1			
unique		Total:			5508	379	47	38	N/A	246 (192)	124 (105)	12 (5)	24	195			
CIVs		cURL	libssl	[5]	198	27	1	1	25% (14/56)	18 (10)	5 (4)	1 (1)	0	17			
	ľ.	GPA	libgpgme		174	9	1	1	4% (3/72)	7 (2)	0 (0)	0 (0)	0	6			
	Safebox	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			
			IIDSSI	[5], [1], [22], [51]	346	39	2	1	11% (11/96)	16 (13)	19 (13)	2(1)	0	20			
		sudo	libapparmor		07	3	1	1	100% (5/5)	2 (2)	2 (0)	0 (0)	0 -	4			
			поарранног		91	5	1	1	100% (2/2)	2 (2)	2 (0)	0 (0)		12-1			
		Total:			9863	250	9	8	N/A	154 (97)	60 (17)	3 (2)	78				

```
// 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.

```
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

```
int sudo passwd verify(struct passwd *pw, char *pass,
  sudo_auth *auth, struct sudo_conv_callback *cb) {
  /* ... abbreviated ... */
  sav = pass[8]; // read CIV
                                               @0xdea
  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

raptor@infosec.exchange

CVE-2022-43995 is really something.

Sudo 1.8.0 through 1.9.12 contains an array-out-of-bounds error that can result in a heap-based buffer over-read. This can be triggered by local users with access to Sudo by entering a password of 7 chars or fewer.

github.com/sudo-project/s...



Fuzzed API Elements (No Vulnerability Found)

Fuzzed API Elements (Vulnerable)



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.



• 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

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

- 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

- 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

- 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

- 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

- 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

- 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**

- 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

 Need approaches focusing on retrofiting compartmentalization in legacy monolithic applications (including OS kernels)

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

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

- Need approaches focusing on retrofiting 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. CHERI¹)

¹Watson, 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.

- Need approaches focusing on retrofiting 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. CHERI¹)
- Need **metrics** to validate, evaluate, quantify and compare approaches (e.g. policies)

¹Watson, 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.

- Need approaches focusing on retrofiting 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. CHERI¹)
- Need metrics to validate, evaluate, quantify and compare approaches (e.g. policies)
- Need tools to **estimate** costs & benefits pre-compartmentalization

¹Watson, 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.

- Need approaches focusing on retrofiting 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. CHERI¹)
- Need metrics to validate, evaluate, quantify and compare approaches (e.g. policies)

1 () ()

- Need tools to **estimate** costs & benefits pre-compartmentalization
- Need more focus on **flexibility**² and **availability**

¹Watson, 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. ²Lefeuvre, 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^v, Yi Chien[‡], Felipe Huici[∞], Nathan Dautenhahn[‡], Pierre Olivier[†] [†]The University of Manchester, ^vUniversity Politehnica of Bucharest, [†]Rice University, [∞]Unikraft.io

Abtrard—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/a a trusted comparent is now potentially a targeted attack-from a malicious compartment. This results in an entire class of security bugs: Compartment Interface Vulnershilties (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 inmemory 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 hulf a century by ensuing 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 damace 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], [5], [5], [4] hat enable fine-grained compartmentalization. This resulted in compartmentalization works targeting finer and finer granularities, such as libraries [67], [60], [19], [42], [53], [39], [5], [51], [51], [21], [21], [22], [21], [23], [16], [21], and even function/blocks of code [16], [64], [57], [11], 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

Network and Distributed System Security (NDSS) Symposium 2023 27 February - 3 March 2023, San Diego, CA, USA ISBN 1-89156-83-5 https://dx.doi.org/10.14722/ndss.2023.24117 www.ndss-symposium.org 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 comparments 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 imfraces with distant in mind. For cample, objects passed through APIs can be compared to launch confused deputy securitors or leak data through laport antacks [81, 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 compartmental ization 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 compartmentating summifying applications? What is the concrete impact of CIVs on the adjet them? What is the concrete impact of CIVs on the adjet we concrete how given by comparison of the concrete impact of the impact of the concrete impact of the concrete impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the concrete impact of the impact of the concrete impact of the concrete impact of the concrete impact of the impact of the concrete impact of

This paper provides an indepth study of CIVs. We taxonomic 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 ConfFazz, 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 fuzzer, hat inject malformed data in a single direction (e.g., a library).

102