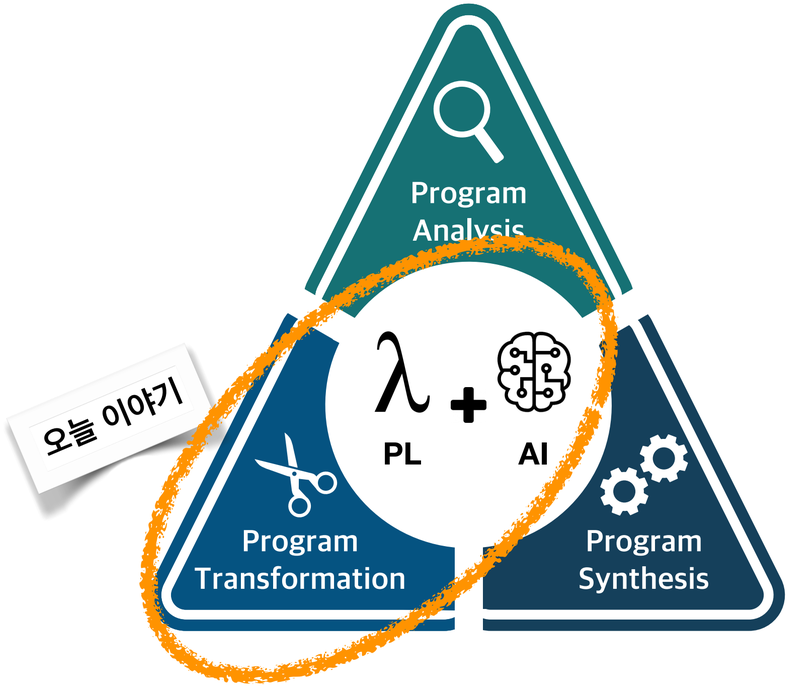
소프트웨어 거품: 문제점과 자동제거 시스템

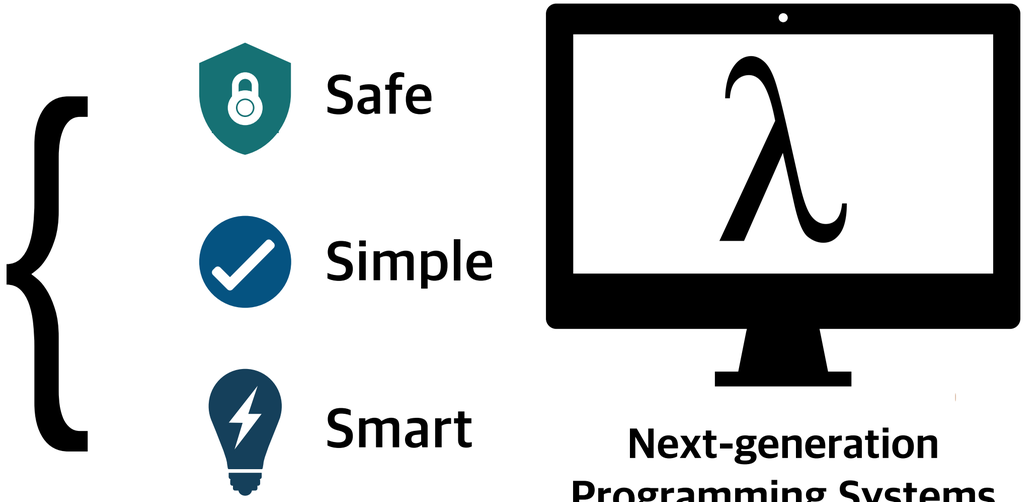
KAIST 전산학부 허기홍

컴퓨터 시스템 소사이어티 동계학술대회 2021

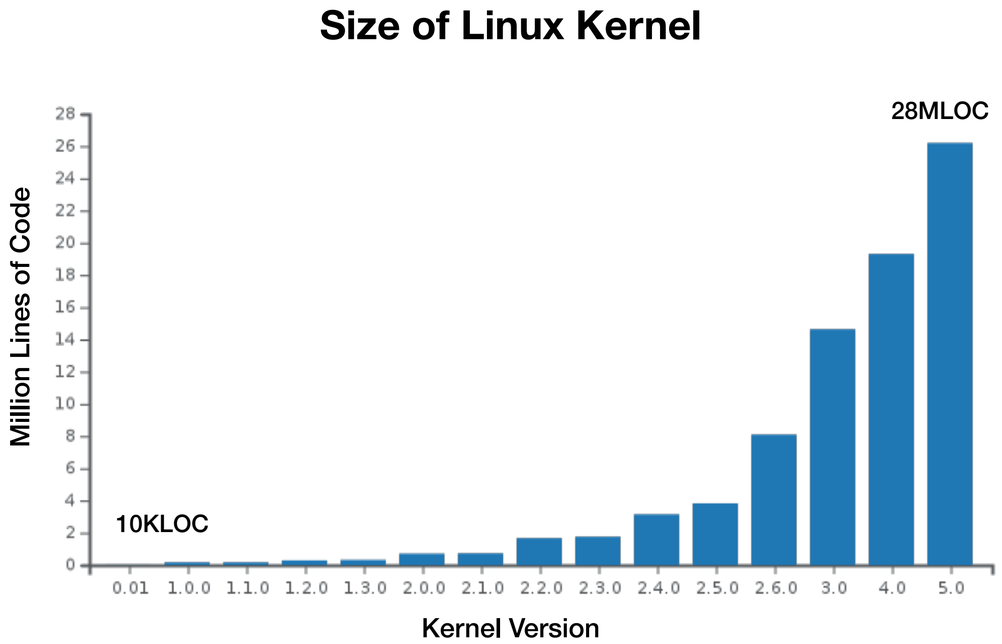
발표자 소개

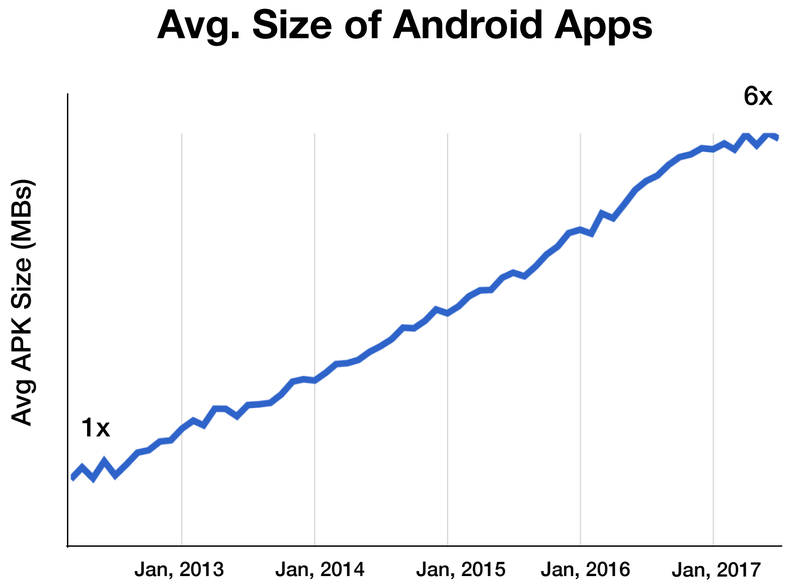
- 소속: KAIST 전산학부 / 정보보호대학원
- 전공: 프로그래밍언어, 프로그램 분석, SW 보안





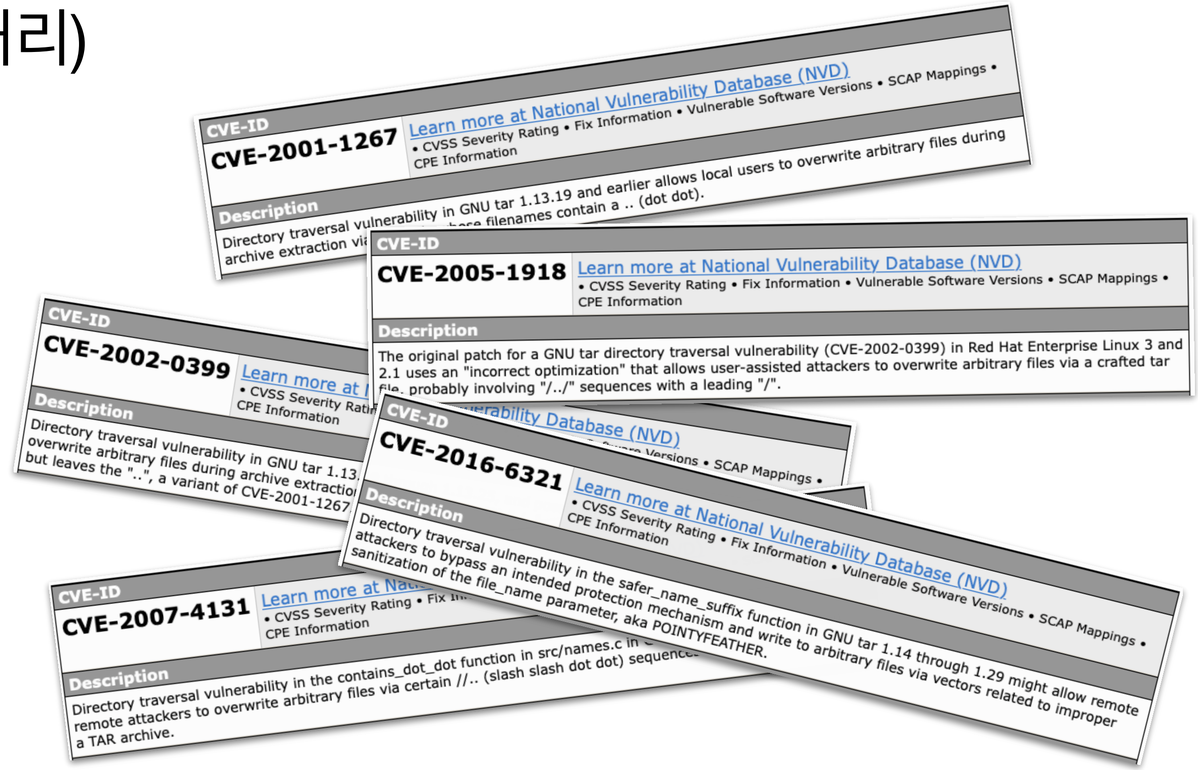
나날이 증가하는 소프트웨어의 복잡도



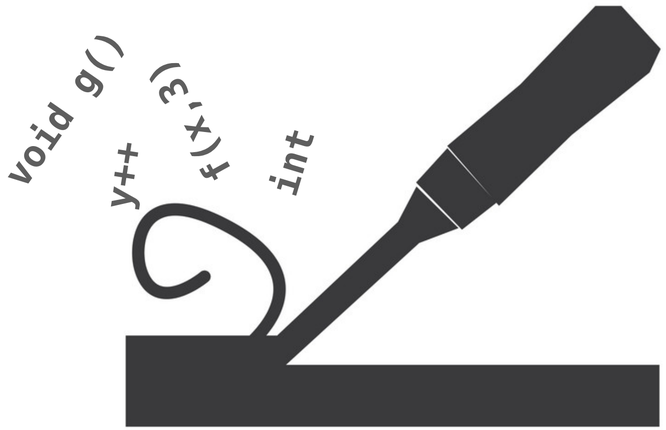


소프트웨어 거품

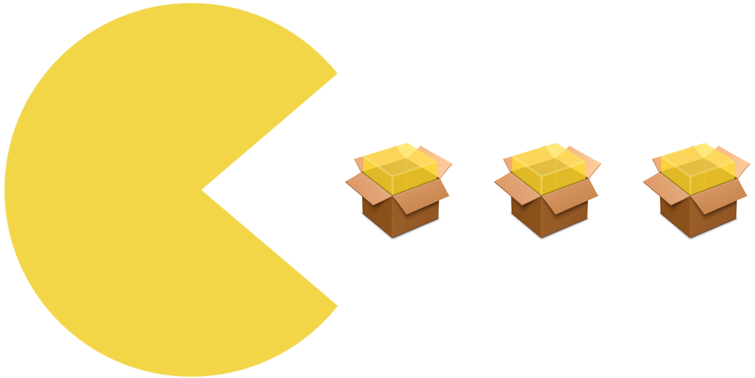
- 과도하게 부풀어 오른 소프트웨어 거품의 원인:
  - 일반적이고 재사용 가능한 코드 (예: 라이브러리)
  - 다양한 사용자의 요구 (예: 환경, 기능)
- 결과:
  - 유지보수 어려움 (예: “dependency hell”)
  - 성능 저하 (예: instruction cache miss)
  - 보안 문제 (예: 미처 생각지 못한…)



소프트웨어 거품 자동 제거 시스템



Chisel: code-level SW debloating system [CCS'18]



PacMan: package-level SW debloating system

기존 방법: GNU tar 사례

General-purpose tar

- Linux default package
- 97 cmd line options
- 45,778 LOC
- 13,227 statements
- CVE-2016-6321

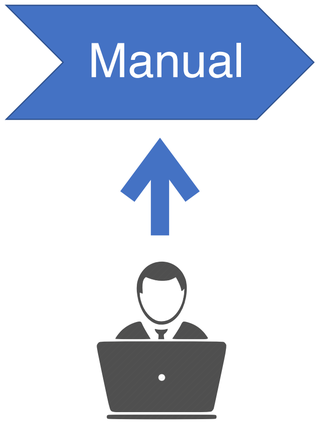
Customized tar

- BusyBox utility package
- 8 cmd line options
- 3,287 LOC
- 403 statements
- No known CVEs

기존 방법: GNU tar 사례

General-purpose tar

- Linux default package
- 97 cmd line options
- 45,778 LOC
- 13,227 statements
- CVE-2016-6321



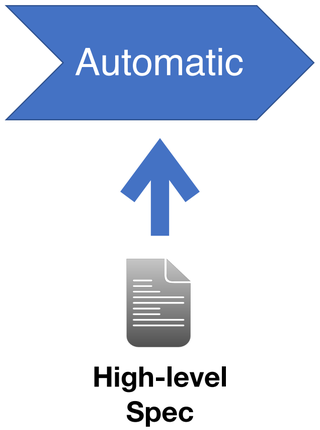
Customized tar

- BusyBox utility package
- 8 cmd line options
- 3,287 LOC
- 403 statements
- No known CVEs

Chisel 의 목표

General-purpose tar

- Linux default package
- 97 cmd line options
- 45,778 LOC
- 13,227 statements
- CVE-2016-6321



Customized tar

- BusyBox utility package
- 8 cmd line options
- ~~3,287~~ **1,646** LOC
- ~~403~~ **518** statements
- No known CVEs

예: tar-1.14

```
int absolute
int ignore_
struct tar_sta

char *safer_name_suffix          file_name, int link_target) {
    int prefix_len;
    char *p;

    if (absolute_names) {
        p = file_name;
    } else {
       /* CVE-2016-6321 */
       /* Incorrect sanitiza      if “file_name” contains ".." */
       ...
    }
    ...
    return p
}

void extract_archive() {
    char *file_name = safer_name_suffix(stat_info.file_name, 0);
    /* Overwrite “file_name” if exists */
    ...
}

void list_archive() { ... }
```

Global variable declarations removed

Code containing CVE removed

Overwriting functionalities removed

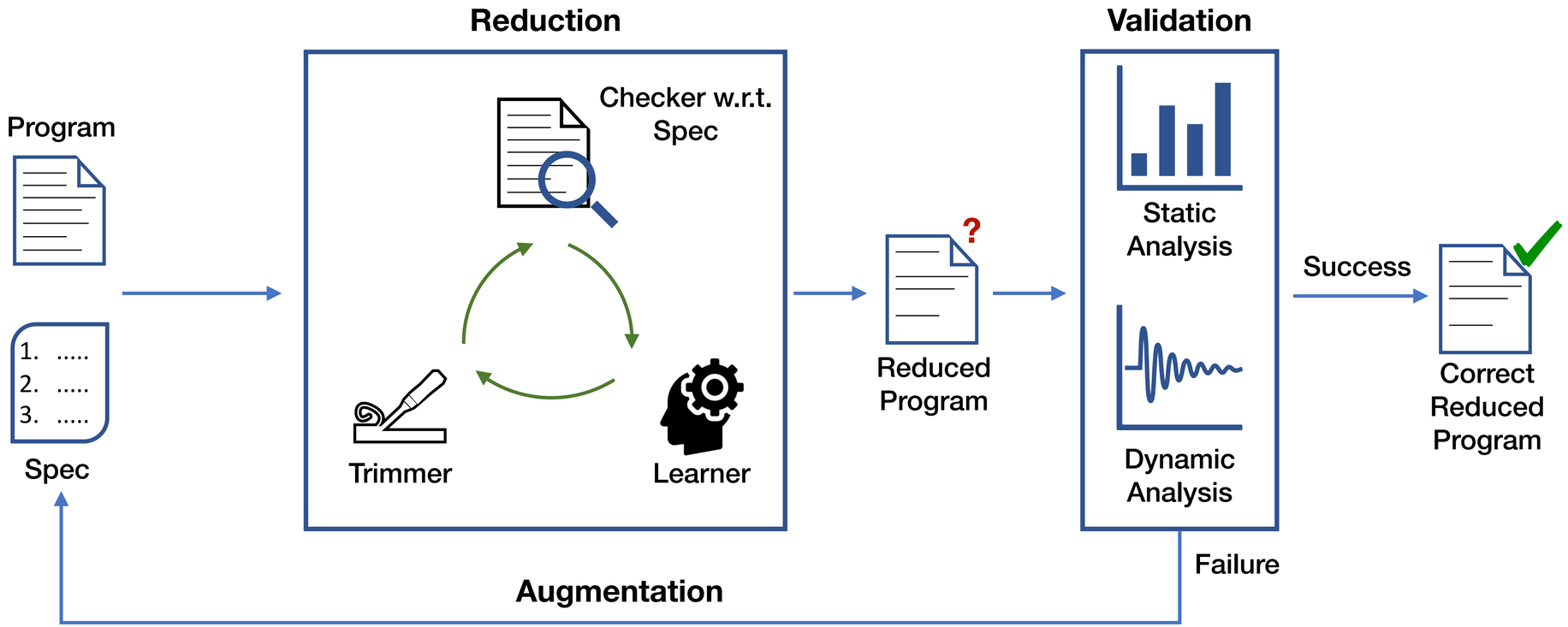
```
void read_and(void *(do_something)(void)) {
    enum read_header status;
    while (...) {
        status = read_header();
        switch (status) {
        case HEADER_SUCCESS: (*do_something)(); continue;
        case HEADER_ZERO_BLOCK:
          if (ignore_zeros_option) continue;
          else break;
        ...
        default:
        }
    }
    ...
}

/* Supports all options: -x, -t, -P, -i, ... */
int main(int argc, char **argv) {
    int optchar;
    while (optchar = getopt_long(argc, argv) != -1) {
        switch(optchar) {
        case 'x': read_and(&extract_archive); break;
        case 't': read_and(&list_archive); break;
        case 'P': absolute_names = 1; break;
        case 'i': ignore_zeros_option = 1; break;
        ...
        }
    }
    ...
}
```

Unnecessary functionalities removed

Unsupported options removed

Chisel 시스템


Reduction
Validation
Program
Checker w.r.t.
Spec
1. .....
2. .....
3. .....
Spec
Trimmer
Learner
Reduced
Program
?
Static
Analysis
Dynamic
Analysis
Success
Correct
Reduced
Program
Failure
Augmentation

명세

```
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}

# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
desired || exit 1
undesired || exit 1
```

명세

```
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}

# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

1. The program is compilable.

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
desired || exit 1
undesired || exit 1
```

명세

```
#!/bin/bash

function compile {
  clang -o tar.debloat tar-1.14.c
  return $?
}
```

```
# tests for the desired functionalities
function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

2. The program produces the same results with the desired functionalities. (e.g., using regression test suites)

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}

compile || exit 1
desired || exit 1
undesired || exit 1
```

명세

```
#!/bin/bash

function desired {
  # 1. archiving multiple files
  touch foo bar
  ./tar.debloat cf foo.tar foo bar
  rm foo bar
  ./tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # 2. extracting from stdin
  touch foo
  ./tar.debloat cf foo.tar foo
  rm foo
  cat foo.tar | ./tar.debloat x
  test -f foo || exit 1

  # other tests
  …
  return 0
}
```

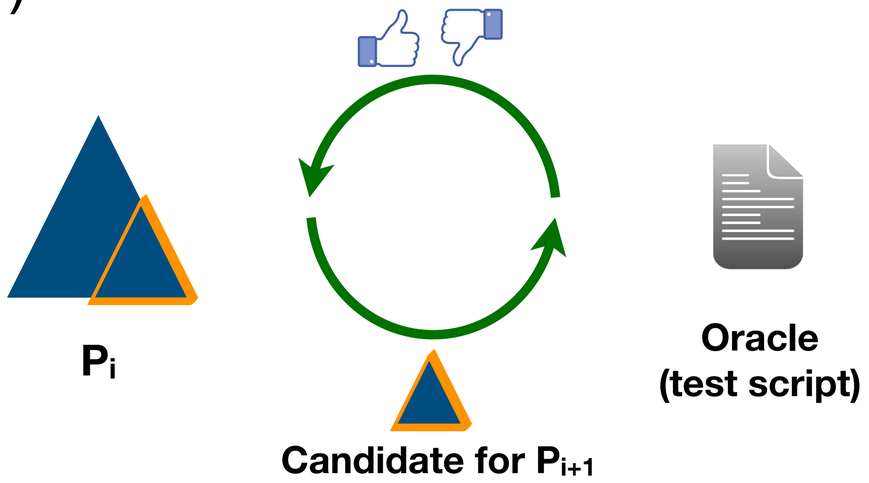
3. The program does not crash with the undesired functionalities. (e.g., using Clang sanitizers)

```
# tests for the undesired functionalities
function undesired {
  for test_script in `ls other_tests/*.sh`
  do
    { sh -x -e $test_script; } >& log
    grep 'Segmentation fault' log && exit 1
  done
  return 0
}
```

```
compile || exit 1
desired || exit 1
undesired || exit 1
```

Delta Debugging 을 이용한 거품 제거

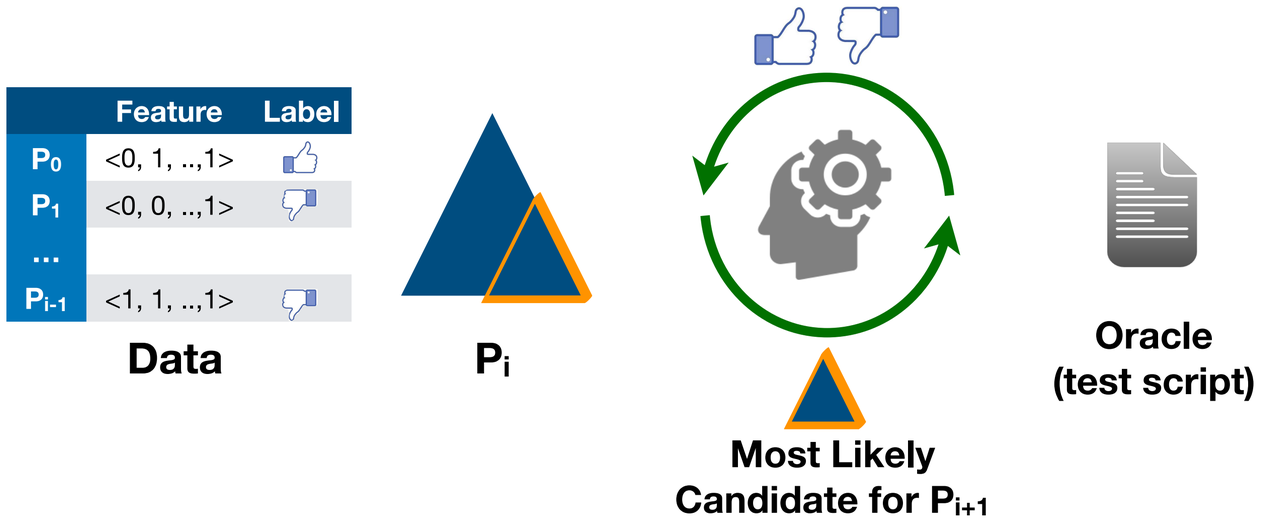
- Oracle $\mathcal{O}$ takes a program and returns **PASS** or **FAIL**
- Given a program ***P***, find a **1-minimal** ***P**** such that $\mathcal{O}$(***P****) = **PASS**
- **1-minimal:** removing any single element of ***P**** does not pass $\mathcal{O}$
- Time complexity: O(|***P***|^2)



*Zeller and Hildebrandt, simplifying and isolating failure-inducing input, TSE, 2002

학습을 이용한 가속

- **Learn a policy** for DD using reinforcement learning (RL)
- **Guide the search** based on the prediction of the learned policy
- Still guarantee **1-minimality** and **O(|*P*|²)** time complexity



실험

코드 크기

#Statement

Program	Original	Chisel	Hand-written
bzip-1.05	6,316	1,575	2,342
chown-8.2	3,422	186	141
date-8.21	4,100	913	107
grep-2.19	10,816	1,071	355
gzip-1.2.4	4,069	1,042	1,058
mkdir-5.2.1	1,746	142	94
rm-8.4			
sort-8.16			
tar-1.14			
uniq-8.16	1,923	192	51
Total	55,848	6,111	4,729

Reachable code by static analysis

Chisel reduced 89%

Comparable to hand-written versions

실험
안전성

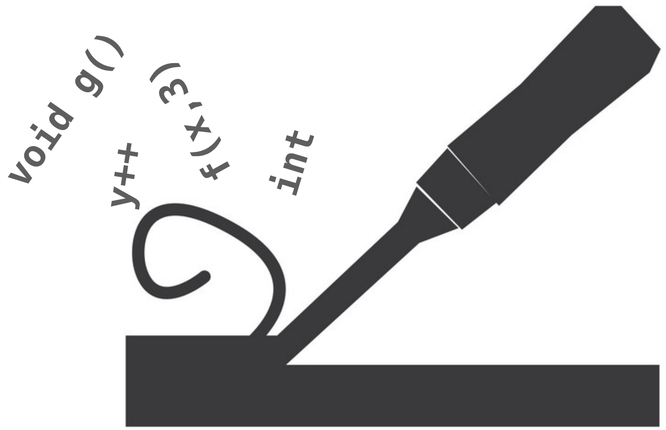
Remove 4 and 2 CVEs in undesired and desired functionalities.
4 CVEs are not easily fixable by reduction (e.g., race condition).

Program	CVE	#ROP Gadgets Original	#ROP Gadgets Reduced		#Alarms Original	#Alarms Reduced	
bzip-1.05	✘	662	298	(55%)	1,991	33	(98%)
chown-8.2	✔	534	162	(70%)	47	1	(98%)
date-8.21	✔	479	233	(51%)	201	23	(89%)
grep-2.19	✔	1,065	411	(61%)	619	31	(95%)
gzip-1.2.4	✔	456	340	(25%)	326	128	(61%)
mkdir-5.2.1	✘	229	124	(46%)	43	2	(95%)
rm-8.4	✘	565	95	(83%)	48	0	(100%)
sort-8.16	✔						
tar-1.14	✔						
uniq-8.16	✘	349	99	(69%)	60	1	(98%)
Total		6,752	2,285	(66%)	5,298	243	(95%)

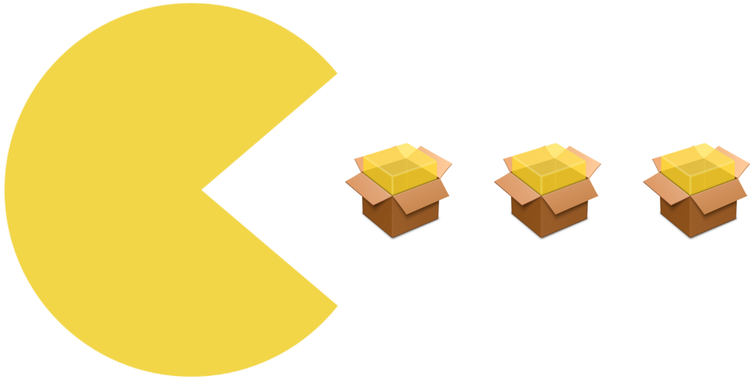
Reduced potential attack surface

Make it feasible for manual alarm inspection

소프트웨어 거품 자동 제거 시스템



Chisel: code-level SW debloating system [CCS'18]



PacMan: package-level SW debloating system

패키지 수준에서 소프트웨어 거품

- 패키지 기반 SW 개발
  - 다른 패키지를 이용해서 SW를 (패키지로) 개발
  - 패키지 매니저 (Apt, Homebrew, NPM, PIP, etc) 를 이용
- **거품**: 모든 가능성을 고려한 의존 관계
  - 예: Chromium 57.0 를 Apt 로 설치할 시 의존 패키지 298 개 설치 (직접 의존: 39개)
- **의문**: 과연 이 중 얼마나 많은 패키지가 실제 사용되는가?

보안 문제

- 너무 많은 패키지 설치 ⇒ 더 많은 위험에 노출
  - 예) VLC: 374 CVE / 321개 의존 패키지
- 유명 패키지 대상 공격 ⇒ 저비용/고효율 공격
  - 예) Python 가짜 악성 패키지
- 문제 원인 파악 어려움 ⇒ 대응 지연
  - 예) VLC — libebml 문제 해결까지 한 달

Two malicious Python libraries caught stealing SSH and GPG keys

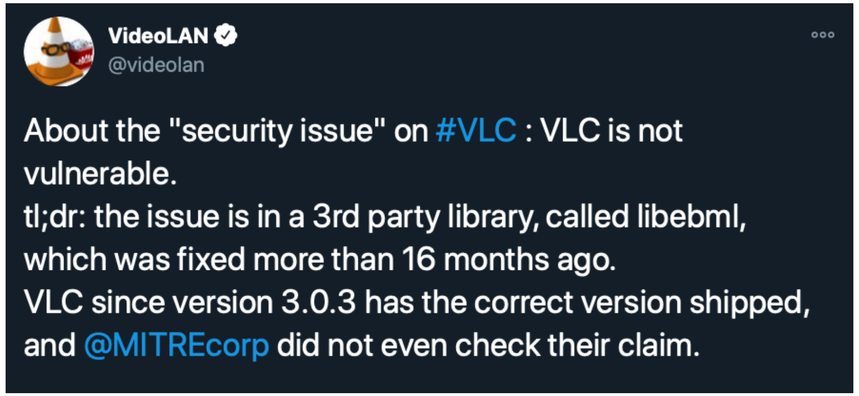
One library was available for only two days, but the second was live for nearly a year.

Catalin Cimpanu • December 4, 2019 -- 00:52 GMT (08:52 SGT)

The Python security team removed two trojanized Python libraries from PyPI (Python Package Index) that were caught stealing SSH and GPG keys from the projects of infected developers.

The two libraries were created by the same developer and mimicked other more popular libraries -- using a technique called typosquatting to register similarly-looking names.

The first is "python3-dateutil," which imitated the popular "dateutil" library. The second is "jellyfish" (the first L is an I), which mimicked the "jellyfish" library.



VideoLAN ✓
@videolan

About the "security issue" on #VLC : VLC is not vulnerable.
tl;dr: the issue is in a 3rd party library, called libebml, which was fixed more than 16 months ago.
VLC since version 3.0.3 has the correct version shipped, and @MITREcorp did not even check their claim.

PacMan 의 목표

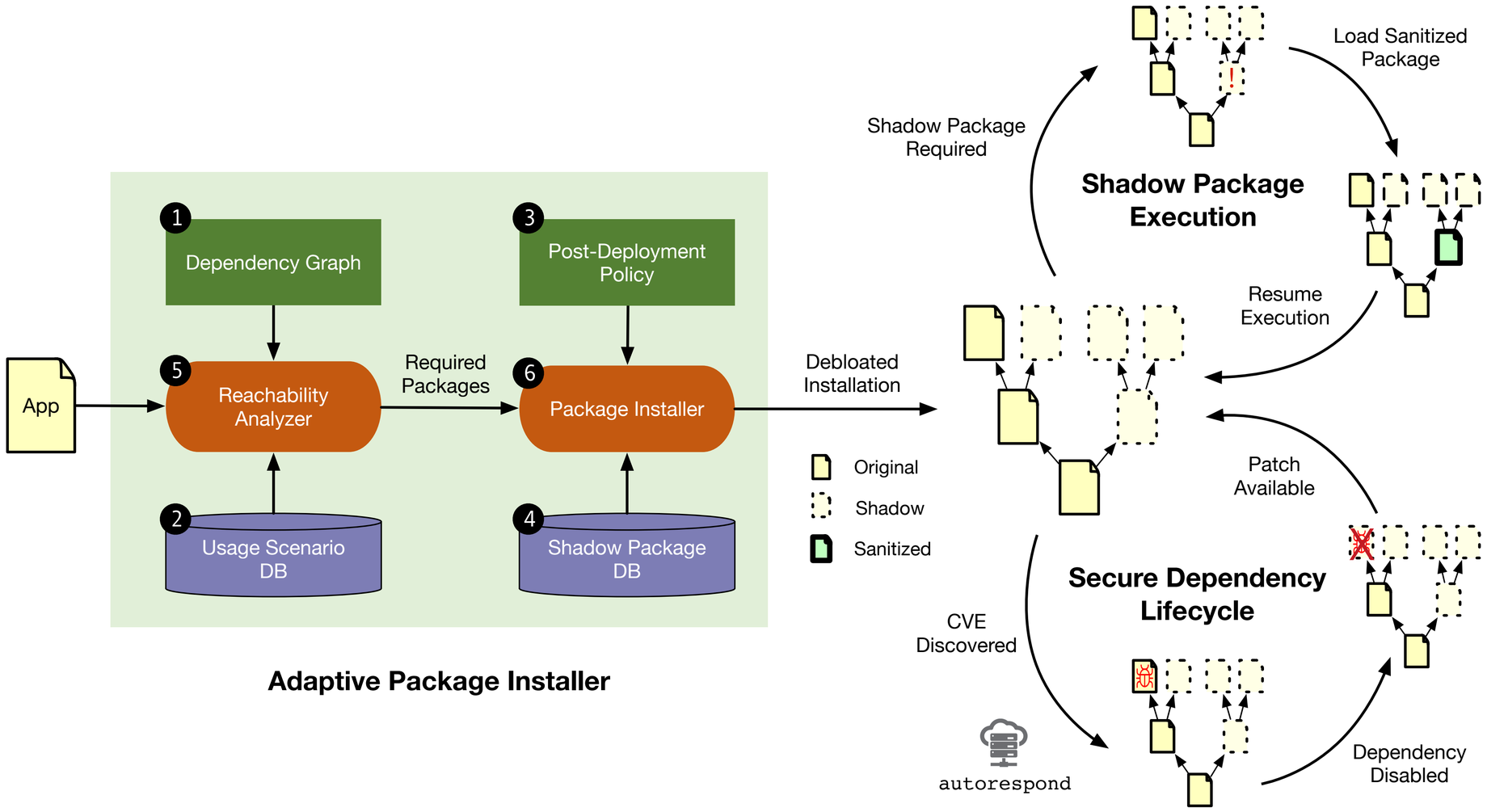
- **최소** 패키지 설치
  - 불필요한 의존 관계 제거
- 패키지 설치 **사용자화**
  - 사용 시나리오에 최적화된 패키지 설치
- **안전한** 의존성 생명 주기
  - 의존 패키지에서 지속적으로 발견, 수정되는 보안 문제에 즉시 대응
  - 사용성을 해치지 않고 손쉽게

정적 분석 (static analysis)

사용 시나리오 DB +
동적 분석 (dynamic analysis)

그림자 패키지

PacMan 시스템



Adaptive Package Installer

그림자 패키지

- 손쉬운 설치/제거를 위해 원본 패키지의 틀만 유지한 채 내용은 모두 삭제한 패키지
  - 틀: ABI (application binary interface)
- 다양한 정책 구현 가능: 그림자 패키지가 실행되면 미리 정해진 정책에 따라 행동
  - Strict mode: 즉시 실행 종료.
  - Decay mode: 즉시 원본 설치 후 계속 진행. 수행 종료 후 일정 기간 안쓰면 그림자로.
  - Permissive mode: 즉시 원본 설치 후 계속 진행.
- 정책에 따라 원본 혹은 특수 처리된 버전 설치 (e.g., sanitized)

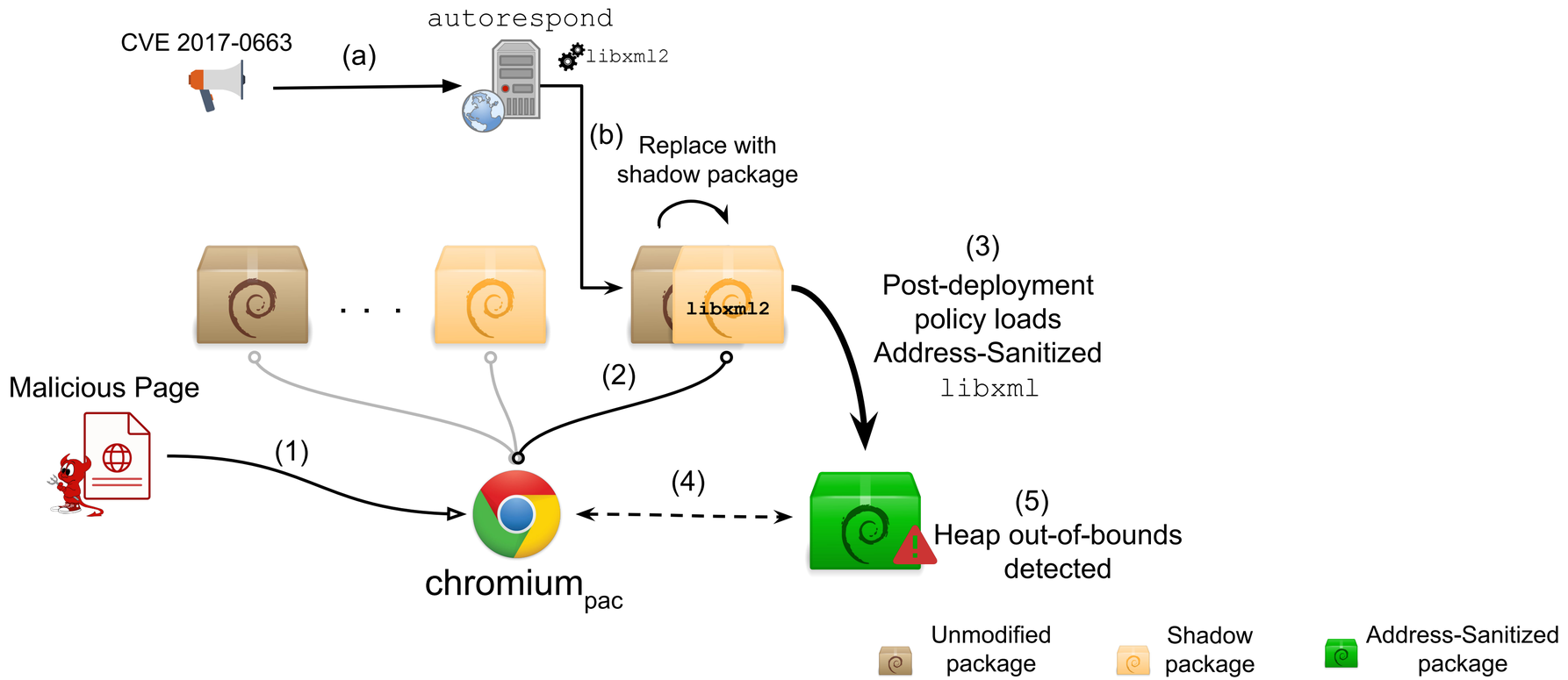
실험

- 사용례 DB: Alexa top 500, Google 검색, StackOverflow, 튜토리얼

	Apt			Reduction by PacMan		
Benchmark	**Deps**	**CVEs**	**Gadgets (K)**	**Dep**	**CVEs**	**Gadgets (K)**
bc	14	30	21	11 (79%)	13 (43%)	16 (76%)
gawk	15	29	26	12 (80%)	16 (55%)	22 (84%)
wget	39	50	143	24 (62%)	26 (52%)	108 (76%)
curl	50	68	168	30 (60%)	26 (38%)	80 (48%)
git	56	75	164	31 (55%)	27 (37%)	115 (70%)
xpdf	92	154	263	64 (70%)	84 (51%)	225 (85%)
firefox	187	182	717	122 (65%)	103 (57%)	541 (75%)
chromium	152	513	338	90 (59%)	150 (59%)	184 (54%)
gimp	250	289	901	171 (68%)	88 (34%)	665 (74%)
vlc	321	374	1361	188 (59%)	164 (44%)	608 (45%)
Average				66%	46%	69%

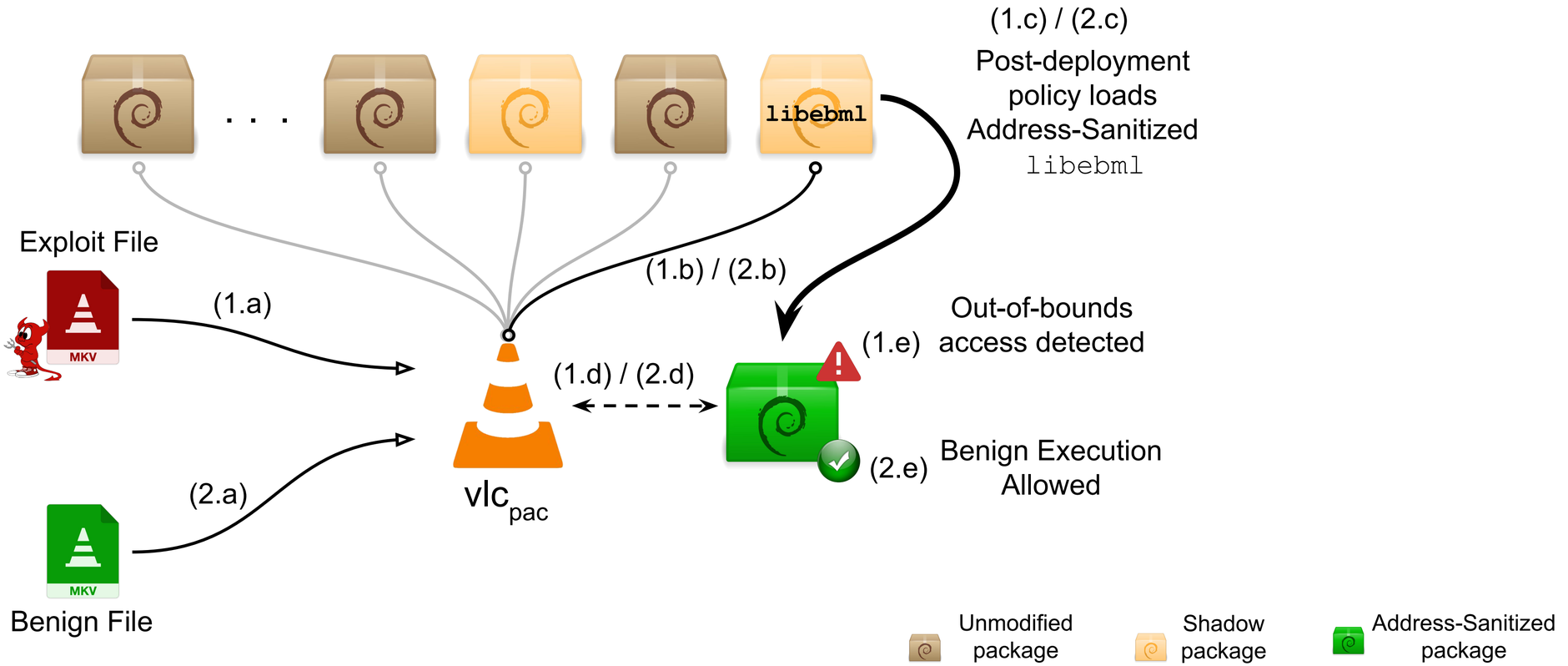
적용 사례 1: Chromium

- 기본 설치: 152 패키지, 513 CVE (APT) vs 62 패키지, 363 CVE (PacMan)



적용 사례 2: VLC

- 기본 설치: 321 패키지, 374 CVE (APT) vs 131 패키지, 210 CVE (PacMan)



결론

- SW 거품: 유지보수, 성능, 보안 문제의 원인
- SW 거품 자동 제거 시스템: 주어진 명세를 만족하는 작고 단단한 SW 로 자동 변환
  - Chisel: 코드 단위 거품 제거
  - PacMan: 패키지 단위 거품 제거