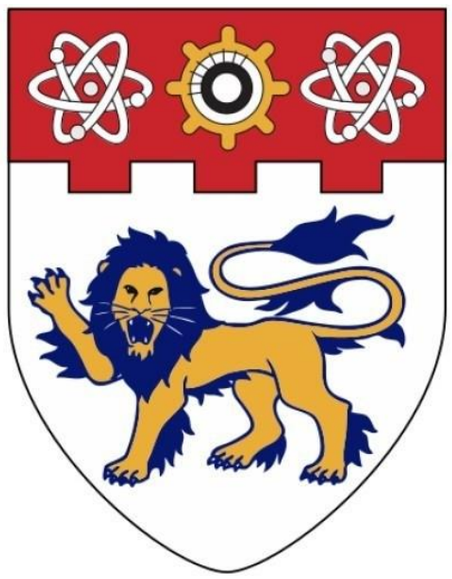
MemLock: Memory Usage Guided Fuzzing

Cheng Wen, Haijun Wang, Yuekang Li, Shengchao Qin, Yang Liu, Zhiwu Xu,
Hongxu Chen, Xiaofei Xie, Geguang Pu and Ting Liu



Coverage-Based Grey-Box Fuzzing

- Software vulnerabilities can be critical, leading to big economic losses or even losses of lives.
- Fuzzing has been an effective and practical technique in detecting various software vulnerabilities
- Coverage-based grey-box fuzzing in particular
  - e.g. AFL-based fuzzing techniques helped find thousands of bugs
- However, current coverage-based greybox fuzzers may have blind spots in detecting vulnerabilities caused by uncontrolled memory consumption

Uncontrolled memory consumption

- A kind of critical software security weaknesses
  - The software does not properly control the allocation of a limited resource thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources (from CWE-400).

- Typical uncontrolled memory consumption vulnerabilities:
  - CWE-674: Uncontrolled Recursion
  - CWE-789: Uncontrolled Memory Allocation
  - CWE-401: Memory Leak

Coverage-based fuzzing may be difficult to detect these vulnerabilities

- These bugs can only be triggered with a sufficiently large *size/depth*.

```
// uncontrolled recursion
void recur (int depth) {
    int a[10];
    return recur(depth-1);
}

int main (void) {
    int depth = get_user_input();
    recur(depth);
    return 0;
}
```

```
//uncontrolled memory allocation
void main (void) {
    int size = get_user_input();
    char *p = malloc(size);
    use(p);
    free(p);
}
```

```
//memory leak with large leakage
void func (void) {
    char *p = malloc(size * sizeof(char))
    use(p);
}
```

- However, such inputs (of a sufficiently large *size/depth*) may not be generated as they will unlikely increase the coverage info.

A real-world vulnerability on uncontrolled recursion CVE-2018-17985 (found by this work)

```
1  struct demangle_component *
2  cplus_demangle_type (struct d_info *di) {
3
4    // "peek" is a single character extracted from the input directly
5    char peek = d_peek_char (di);
6
7    switch (peek){
8      ...
9      case 'P':
10       ret = d_make_comp (di,
11         DEMANGLE_COMPONENT_POINTER,
12         cplus_demangle_type (di), NULL);
13       break;
14     case 'C':
15     ...
16   }
17   ...
18 }
```

Recursive function

AFL Testcase	Recursive Depth	Status
p	1	Interesting
pp	2	Interesting
pppp	4	Interesting
pppp...ppp	128	Interesting
pppp...ppppp	260	**Discard**
pppp...pppppp	...	**Discard**
pppp...ppppppp	**35000+**	**Run out of stack memory**

A real-world vulnerability on uncontrolled memory allocation CVE-2018-4866

```
1  class EXIV2API DataBuf {
2  public:
3    // Constructor with an initial buffer size
4    explicit DataBuf(long size): pData(new byte[size]), size(size) {}
5    ...
6    byte* pData; // Pointer to the buffer
7    size_t size; // The current size of the buffer
8  };
9
10 void Jp2Image::readMetadata() {
11   while (io_->read((byte*)&subBox, sizeof(subBox)) ==
     ↪  sizeof(subBox) && subBox.length ) {
12     subBox.length = getLong((byte*)&subBox.length, bigEndi
13     DataBuf data(subBox.length); // Allocation without che
14     ...
15     io_->seek(position - sizeof(box) + box.length, BasicI
16   }
17 }
```

memory allocation

subBox is extracted from the user inputs

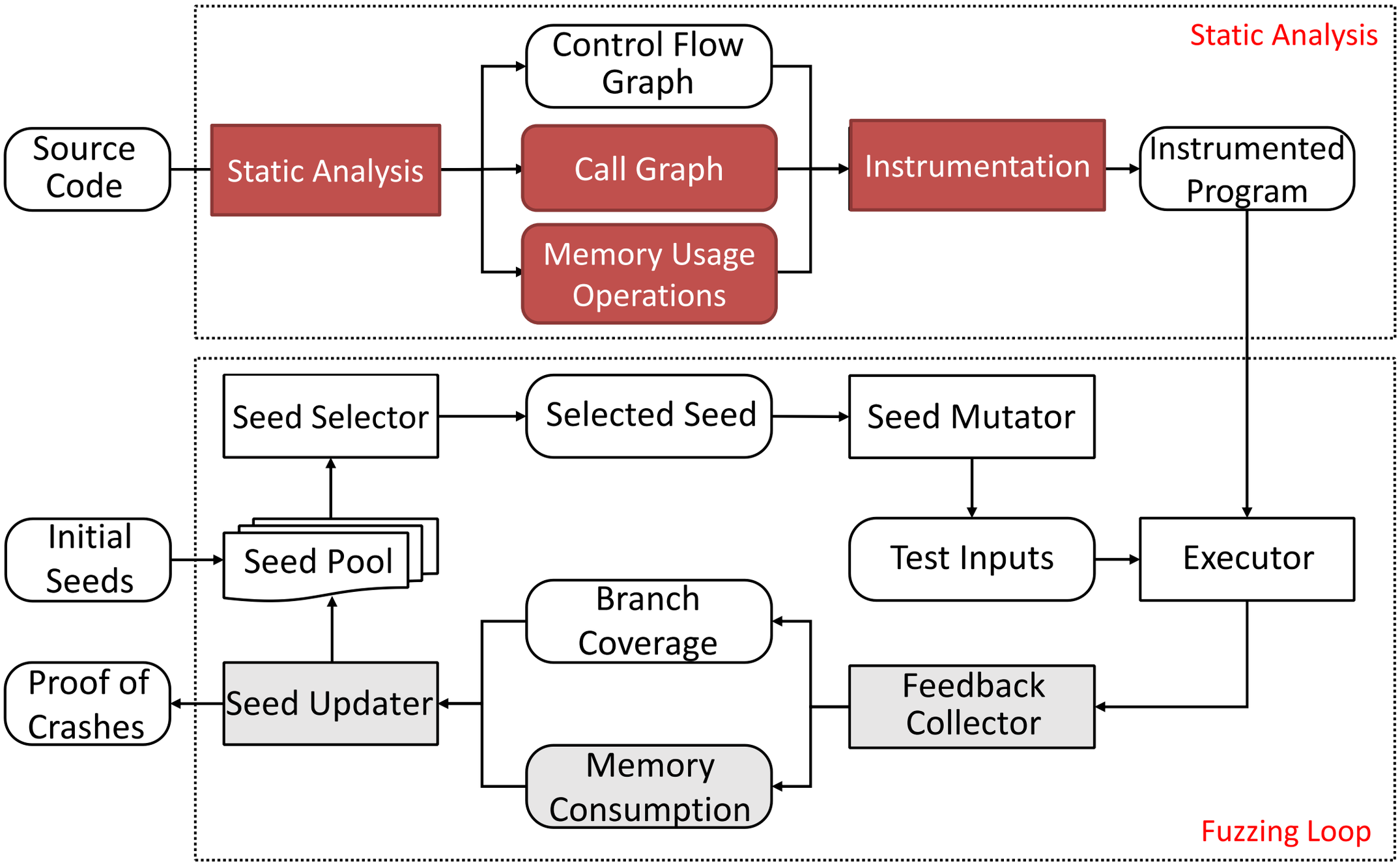
allocate memory without checking

AFL Testcase (size)	Status
2048	Interesting
8192	**Discard**
61440	**Discard**
2097152	**Discard**
53687092	**Discard**
…	**Discard**
4294967296+	**Allocation fails**

Our Proposal ***MemLock***: Memory Usage Guided Fuzzing

- Use light-weight program **instrumentation** to collect the memory consumption information.
- Guide the fuzzing process with new-dimensional **feedback** guidance information--memory consumption.
- Through a novel **seed updating scheme** to retain the most interesting input for each path.

MemLock Overview



Instrumentation

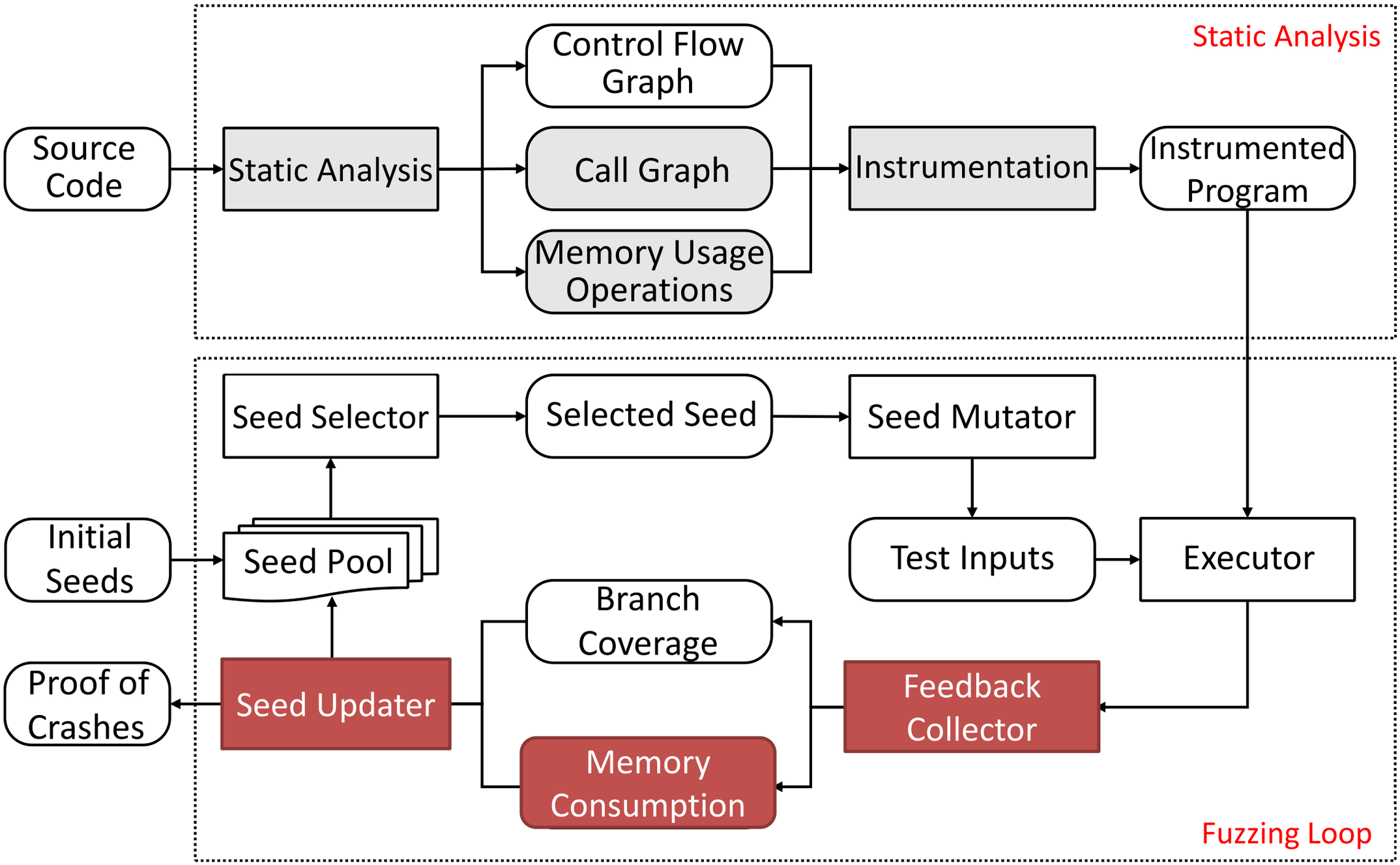
- **Instrument the program based on the static analysis of**
  - Control Flow Graph (CFG)

    add instrumentation into every edge of the CFG to obtain the coverage information.
  - Call Graph

    add instrumentation into both the entry and the exit of the function call to obtain the recursive depth.
  - Memory Usage Operations

    add instrumentation into the memory allocation/deallocation functions and obtain its parameters and return value.

MemLock Overview



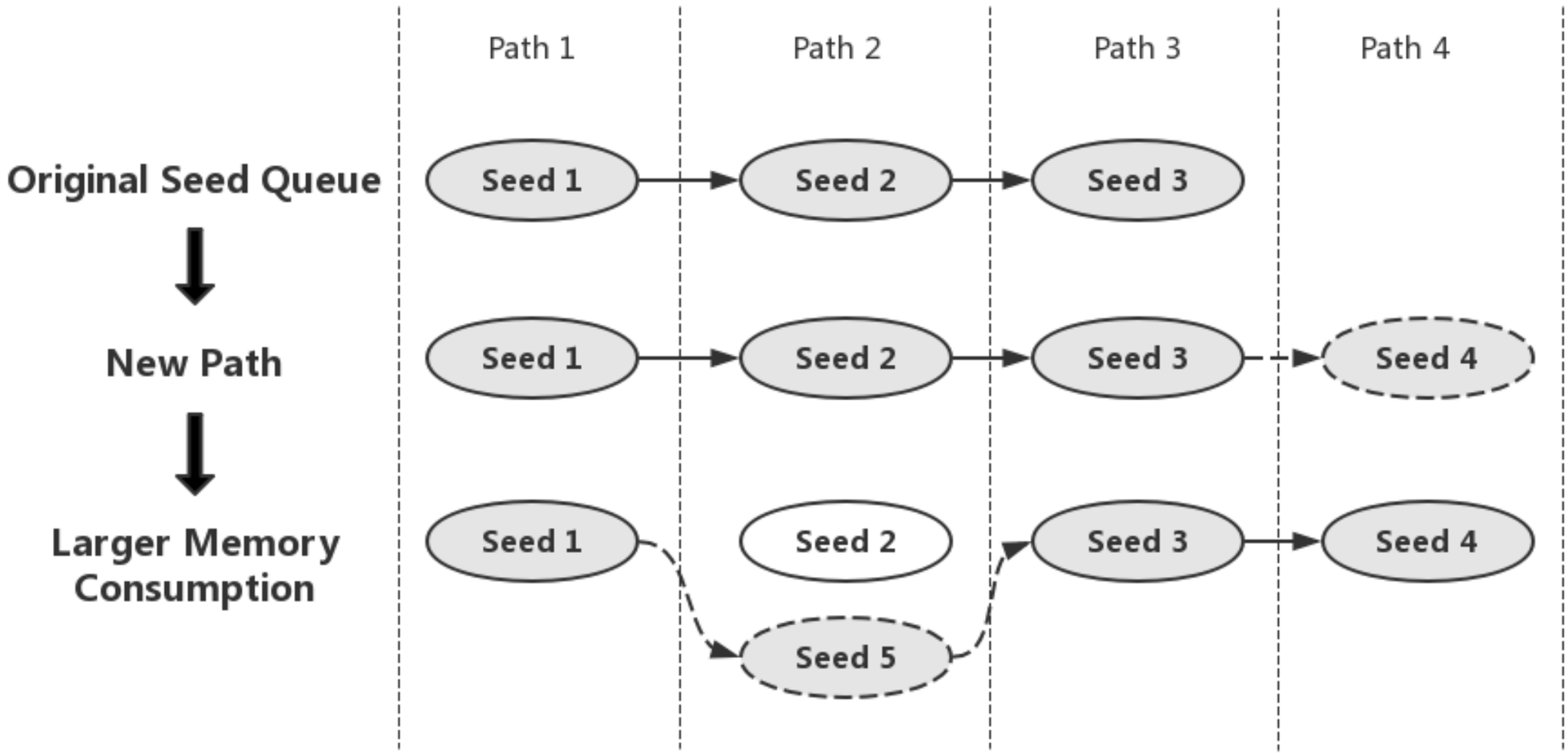
Feedback

- **Fuzzing is Guided by two-dimensional information**
  - Coverage

    guides the exploration of different program paths.
  - Memory consumption

    guides the search for those program paths that exhibit more memory consumption.

Seed Updater



Seed updating scheme

CVE-2018-17985

- **Code Snippet from *cp-demangle.c* in *Binutils v2.31***

```
1  struct demangle_component *
2  cplus_demangle_type (struct d_info *di) {
3
4    // "peek" is a single character extracted from the input directly
5    char peek = d_peek_char (di);
6
7    switch (peek){
8      ...
9      case 'P':
10       ret = d_make_comp (di,
11         DEMANGLE_COMPONENT_POINTER,
12         cplus_demangle_type (di), NULL);
13       break;
14     case 'C':
15     ...
16   }
17   ...
18 }
```

Recursive function

Memlock Testcase	Recursive Depth	Status
p	1	Interesting
pp	2	updating
pppp	4	updating
pppp...ppp	128	updating
pppp...ppppp	260	updating
pppp...pppppp	...	updating
pppp...ppppppp	**35000+**	**trigger the bug**

CVE-2018-4866

- **Code Snippet from *jp2image.cpp* in *Exiv2 v0.26***

```
1  class EXIV2API DataBuf {
2  public:
3    // Constructor with an initial buffer size
4    explicit DataBuf(long size): pData(new byte[size]), size(size) {}
5    ...
6    byte* pData; // Pointer to the buffer
7    size_t size; // The current size of the buffer
8  };
9
10 void Jp2Image::readMetadata() {
11   while (io_->read((byte*)&subBox, sizeof(subBox)) ==
     ↪  sizeof(subBox) && subBox.length ) {
12     subBox.length = getLong((byte*)&subBox.length, bigEnd
13     DataBuf data(subBox.length); // Allocation without che
14     ...
15     io_->seek(position - sizeof(box) + box.length, BasicI
16   }
17 }
```

memory allocation

subBox is extracted from the user inputs

allocate memory without checking

AFL Testcase (size)	Status
2048	Interesting
8192	updating
61440	updating
2097152	updating
53687092	updating
…	updating
4294967296+	**trigger the bug**

Experiments

- **MemLock is built on top of AFL**
- **Baseline fuzzers:**
  - AFL [AFL 2.52b]
  - AFLfast [Böhme2017]
  - PerfFuzz [Lemieux2018]
  - FairFuzz [Lemieux2018]
  - Angora [Chen2018]
  - QSYM [Yun2018]
- **Evaluation dataset:** 14 widely-used real-world programs
- **Runtime:** 24hours, 5 times

Unique Crashes Evaluation

Program	Bug Type	MemLock	AFL		AFLfast		PerfFuzz		FairFuzz		Angora		QSYM	
		#crashes	#crashes	$\widehat{A}_{12}$	#crashes	$\widehat{A}_{12}$	#crashes	$\widehat{A}_{12}$	#crashes	$\widehat{A}_{12}$	#crashes	$\widehat{A}_{12}$	#crashes	$\widehat{A}_{12}$
mjs	UR	114	36	1.00	31	1.00	88	0.96	12	1.00	0	1.00	30	1.00
cxxfilt	UR	448	373	1.00	304	1.00	401	0.88	39	1.00	0	1.00	327	1.00
nm	UR	127	12	1.00	21	1.00	17	1.00	0	1.00	0	1.00	20	1.00
nasm	UR	132	6	1.00	4	1.00	40	1.00	0	1.00	0	1.00	4	1.00
flex	UR	61	0	1.00	0	1.00	0	1.00	0	1.00	0	1.00	0	1.00
yaml-cpp	UR	4	0	1.00	1	1.00	3	0.56	0	1.00	0	1.00	0	1.00
libsass	UR	23	6	1.00	4	1.00	23	0.53	11	0.88	26	0.25	7	1.00
yara	UR	156	34	1.00	33	1.00	65	0.94	13	1.00	0	1.00	31	1.00
readelf	UA	273	104	1.00	110	1.00	54	1.00	181	0.88	0	1.00	114	1.00
exiv2	UA	10	11	0.14	11	0.20	6	0.90	15	0.00	13	0.16	8	0.52
openjpeg	UA	16	8	0.80	5	1.00	0	1.00	7	0.46	0	1.00	5	0.80
bento4	UA	*5*	2	1.00	2	0.98	2	1.00	1	1.00	189	0.00	1	1.00
	ML	154	78	1.00	72	1.00	61	1.00	125	1.00	290	0.00	74	1.00
libming	UA	18	20	0.40	18	0.60	17	0.62	20	0.20	3	1.00	16	0.80
	ML	264	336	0.20	324	0.00	324	0.00	371	0.00	87	1.00	354	0.00
jasper	UA	3	2	0.84	3	0.56	0	1.00	3	0.56	2	1.00	2	0.92
	ML	210	243	0.08	235	0.08	35	1.00	216	0.40	820	0.00	212	0.46
Total (improv.)		**2009**	1262 (+59.2%)		1178 (+70.5%)		1136 (+76.9%)		1014 (+98.1%)		1430 (+40.5%)		1205 (+66.7%)	

**UR means the uncontrolled-recursion bug, UA means the uncontrolled-memory-allocation bug, and ML means the memory leak.*

Crashes Growth

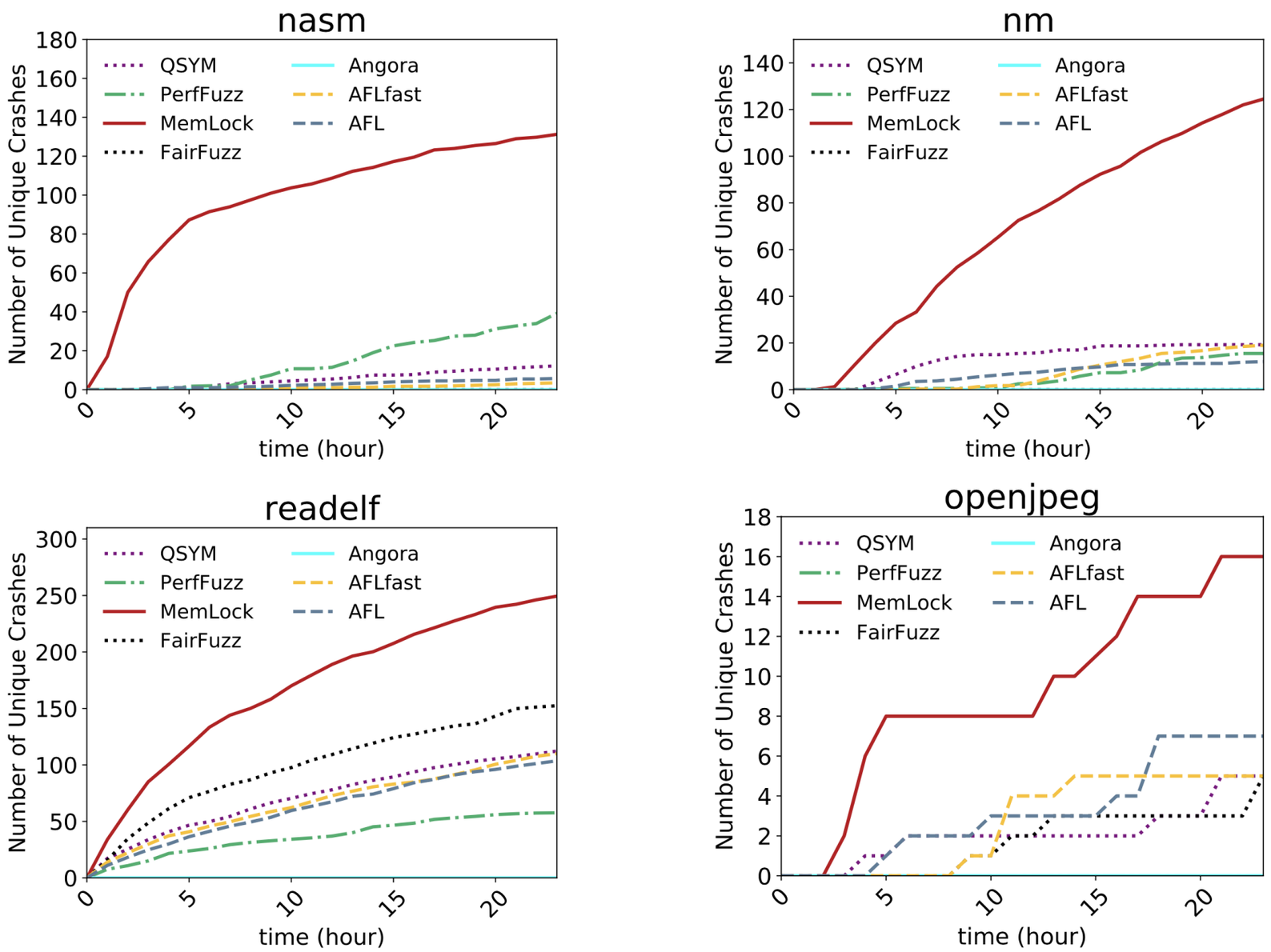


Fig. The growth trend of unique crashes

Time-to-Bugs

Program	Vulnerability	MemLock	AFL		AFLfast		PerfFuzz		FairFuzz		Angora		QSYM	
		Time(h)	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$
mjs	issue#58	0.5	0.3	0.25	0.4	0.25	0.2	0.13	0.4	0.25	T/O	1.00	0.3	0.22
	Issue#106	13.7	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
cxxfilt	CVE-2018-9138	0.3	7.2	1.00	10.1	1.00	0.5	0.81	T/O	1.00	T/O	1.00	3.3	1.00
	CVE-2018-9996	T/O	16.5	0.00	T/O	0.50	T/O	0.50	T/O	0.50	T/O	0.50	T/O	0.50
	CVE-2018-17985	0.2	1.1	1.00	4.5	1.00	0.2	0.63	1.9	1.00	T/O	1.00	1.4	1.00
	CVE-2018-18484	0.2	1	1.00	4.5	1.00	0.2	0.63	8	1.00	T/O	1.00	1.4	1.00
	CVE-2018-18700	0.2	1.2	1.00	4.5	1.00	0.3	0.75	12.6	0.88	T/O	1.00	1.4	1.00
nm	CVE-2018-12641	2.6	19.1	1.00	12.6	1.00	12.2	0.88	T/O	1.00	T/O	1.00	12.8	0.88
	CVE-2018-17985	10.4	18.2	0.81	11.9	0.56	T/O	1.00	T/O	1.00	T/O	1.00	13.3	0.63
	CVE-2018-18484	9.9	16.4	0.84	17.1	0.84	T/O	1.00	T/O	1.00	T/O	1.00	14	0.75
	CVE-2018-18700	9.6	14.9	0.63	17.8	0.88	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
	CVE-2018-18701	13.9	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
	CVE-2019-9070	18.4	15.6	0.56	13.9	0.44	T/O	1.00	T/O	1.00	T/O	1.00	15.8	0.56
	CVE-2019-9071	12.4	T/O	0.88	14	0.69	T/O	0.88	T/O	0.88	T/O	1.00	T/O	0.88
nasm	CVE-2019-6290	0.9	T/O	1.00	19	1.00	9	1.00	T/O	1.00	T/O	1.00	17.6	0.00
	CVE-2019-6291	1.5	9	0.94	14	1.00	8.7	1.00	T/O	1.00	T/O	1.00	7.5	0.92
flex	CVE-2019-6293	5.4	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
yaml-cpp	CVE-2019-6292	0.4	T/O	1.00	18.4	1.00	0.9	0.81	T/O	1.00	T/O	1.00	T/O	1.00
	CVE-2018-20573	6.1	T/O	0.88	T/O	0.84	12.4	0.84	T/O	0.84	T/O	1.00	T/O	0.84

Time-to-Bugs

Program	Vulnerability	MemLock	AFL		AFLfast		PerfFuzz		FairFuzz		Angora		QSYM	
		Time(h)	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$	Time(h)	$\widehat{A}_{12}$
libsass	CVE-2018-19837	1.6	13.3	0.88	10.5	0.88	1.8	0.63	8.5	0.88	T/O	1.00	5	0.81
	CVE-2018-20821	0.1	5.7	1.00	6.5	1.00	0.1	0.50	9.5	1.00	T/O	1.00	7.4	1.00
	CVE-2018-20822	15.6	14.3	0.50	19.5	0.56	14.6	0.47	11.3	0.56	0.92	0.00	10.5	0.44
yara	CVE-2017-9438	0.2	0.9	1.00	4.3	1.00	0.6	0.91	5.3	1.00	T/O	1.00	0.8	1.00
readelf	CVE-2017-15996	0.2	0.3	0.86	0.2	0.68	0.5	0.92	0.3	0.68	T/O	1.00	0.3	0.96
exiv2	CVE-2018-4868	0.1	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50
bento4	CVE-2018-20186	0.4	0.4	0.50	0.4	0.50	0.4	0.50	0.4	0.50	0.1	0.00	0.4	0.50
	CVE-2019-7698	14.6	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	0.5	0.00	T/O	1.00
libming	CVE-2019-7581	0.6	0.8	0.68	1.4	0.80	2	0.88	0.4	0.36	T/O	1.00	1.6	0.80
	CVE-2019-7582	0.1	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50	0.1	0.50
	issue#155	1.4	1	0.30	1.3	0.36	1.4	0.40	1.2	0.42	T/O	1.00	1.6	0.64
openjpeg	CVE-2019-6988	7.8	15.1	0.86	11.1	0.84	T/O	1.00	T/O	1.00	T/O	1.00	15.3	0.81
	CVE-2017-12982	4.5	11.4	0.72	10	0.60	T/O	1.00	11.9	0.64	T/O	1.00	10	0.50
jasper	CVE-2016-8886	4.1	17	0.88	22.3	1.00	T/O	1.00	10.3	0.52	T/O	1.00	18.2	0.88
	issue#207	1.7	2.2	0.62	3.6	0.68	T/O	1.00	2.2	0.68	15.9	1.00	4	0.64
Average Time Usage(Improv.)		**5.4**	11.6 (2.15X)		11.6 (2.15X)		11.9 (2.20X)		14.5 (2.69X)		20.3 (3.76X)		11.2 (2.07X)	
Unique Vulnerabilities(Improv.)		**33**	26 (+26.9%)		28 (+17.9%)		20 (+65.0%)		17 (+94.1%)		6 (+450.0%)		25 (+37.0%)	

**T/O means the fuzzer can't find this vulnerability throughout 24 hours across 5 repetitions. When we calculate the average time usage, we replace T/O with 24 hours.*

Memory Leakage (Bytes)

Program	Type	Tool	leakge (Bytes)	Improve.	p-value	$\hat{A}_{12}$
bento4	memory leak	MemLock	**52,709,574**	-	-	-
		AFL	151,862	+34609%	0.0061	1.00
		AFLfast	1,233,255	+4174%	0.0061	1.00
		PerfFuzz	105,984	+49633%	0.0061	1.00
		FairFuzz	1,910,466	+2659%	0.0061	1.00
		Angora	141,512	+37147%	0.0060	1.00
		QSYM	15,784,847	+234%	0.0061	1.00
libming	memory leak	MemLock	**176,320,785**	-	-	-
		AFL	4,869,594	+3521%	0.0061	1.00
		AFLfast	2,535,212	+6855%	0.0061	1.00
		PerfFuzz	47,044,964	+257%	0.0061	1.00
		FairFuzz	828,742	+21176%	0.0061	1.00
		Angora	4,698	+3753163%	0.0060	1.00
		QSYM	1,219,093	+14363%	0.0061	1.00
jsaper	memory leak	MemLock	**2,372,844,732**	-	-	-
		AFL	56,018,839	+4136%	0.0061	1.00
		AFLfast	48,403,244	+4802%	0.0061	1.00
		PerfFuzz	6,229,898	+37988%	0.0061	1.00
		FairFuzz	56,788,235	+4096%	0.0061	1.00
		Angora	191,907,941	+1136%	0.0105	0.98
		QSYM	38,244,568	+6104%	0.0061	1.00

Table. Total leak bytes

Real-world Vulnerabilities We Found – 26 CVEs

CVE ID	Product	Vulnerability Type
CVE-2020-36375	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36374	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36373	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36372	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36371	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36370	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36369	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36368	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36367	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-36366	*MJS 1.20.1*	Uncontrolled Recursion
CVE-2020-18392	*MJS 1.20.1*	Uncontrolled Recursion
issue#181	*libming 0.4.8*	Uncontrolled Recursion
issue#142	*Exiv2 0.27*	Uncontrolled Memory Allocation
CVE-2019-6293	*Flex 2.6.4*	Uncontrolled Recursion
CVE-2019-6292	*Yaml-cpp v0.6.2*	Uncontrolled Recursion
CVE-2019-6291	*NASM 2.14.03*	Uncontrolled Recursion
CVE-2019-6290	*NASM 2.14.03*	Uncontrolled Recursion
CVE-2018-18701	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-18700	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-18484	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-17985	*Binutils v2.31*	Uncontrolled Recursion
CVE-2019-7704	*Binaryen 1.38.22*	Uncontrolled Memory Allocation
CVE-2019-7698	*Bento4 v1.5.1-624*	Uncontrolled Memory Allocation
CVE-2019-7148	*Elfutils 0.175*	Uncontrolled Memory Allocation
CVE-2018-20652	*Tinyexr 0.9.5*	Uncontrolled Memory Allocation
CVE-2018-18483	*Binutils v2.31*	Uncontrolled Memory Allocation
CVE-2018-20657	*Binutils 2.31*	Memory Leak
CVE-2018-20002	*Binutils v2.31*	Memory Leak

MemLock: Memory Usage Guided Fuzzing

Coverage-Based Grey-Box Fuzzing

- Software vulnerabilities can be critical, leading to big economic losses or even losses of lives.
- Fuzzing has been an effective and practical technique in detecting various software vulnerabilities
- Coverage-based grey-box fuzzing in particular
  - e.g. AFL-based fuzzing techniques helped find thousands of bugs
- However, current coverage-based greybox fuzzers may have blind spots in detecting vulnerabilities caused by uncontrolled memory consumption

Coverage-based fuzzing may be difficult to detect these vulnerabilities

- These bugs can only be triggered with a sufficiently large *size/depth*.

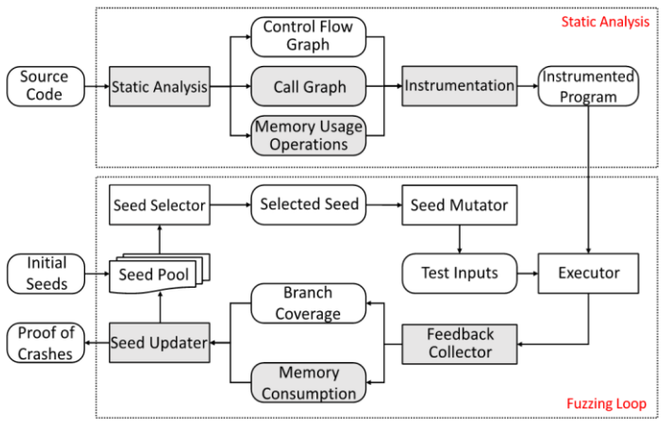
```
// uncontrolled recursion
void recur (int depth) {
    int a[10];
    return recur(depth-1);
}

int main (void) {
    int depth = get_user_input();
    recur(depth);
    return 0;
}
```

```
//uncontrolled memory allocation
void main (void) {
    int size = get_user_input();
    char *p = malloc(size);
    use(p);
    free(p);
}
```

```
//memory leak with large leakage
void func (void) {
    char *p = malloc(size * sizeof(char))
    use(p);
}
```

- However, such inputs (of a sufficiently large *size/depth*) may not be generated as they will unlikely increase the coverage info.

MemLock Overview



Real-world Vulnerabilities We Found – 15 CVEs

CVE ID	Product	Vulnerability Type
CVE-2019-6293	*Flex 2.6.4*	Uncontrolled Recursion
CVE-2019-6292	*Yaml-cpp v0.6.2*	Uncontrolled Recursion
CVE-2019-6291	*NASM 2.14.03*	Uncontrolled Recursion
CVE-2019-6290	*NASM 2.14.03*	Uncontrolled Recursion
CVE-2018-18701	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-18700	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-18484	*Binutils v2.31*	Uncontrolled Recursion
CVE-2018-17985	*Binutils v2.31*	Uncontrolled Recursion
CVE-2019-7704	*Binaryen 1.38.22*	Uncontrolled Memory Allocation
CVE-2019-7698	*Bento4 v1.5.1-624*	Uncontrolled Memory Allocation
CVE-2019-7148	*Elfutils 0.175*	Uncontrolled Memory Allocation
CVE-2018-20652	*Tinyexr 0.9.5*	Uncontrolled Memory Allocation
CVE-2018-18483	*Binutils v2.31*	Uncontrolled Memory Allocation
CVE-2018-20657	*Binutils 2.31*	Memory Leak
CVE-2018-20002	*Binutils v2.31*	Memory Leak

`git clone https://github.com/ICSE2020-MemLock/MemLock.git`