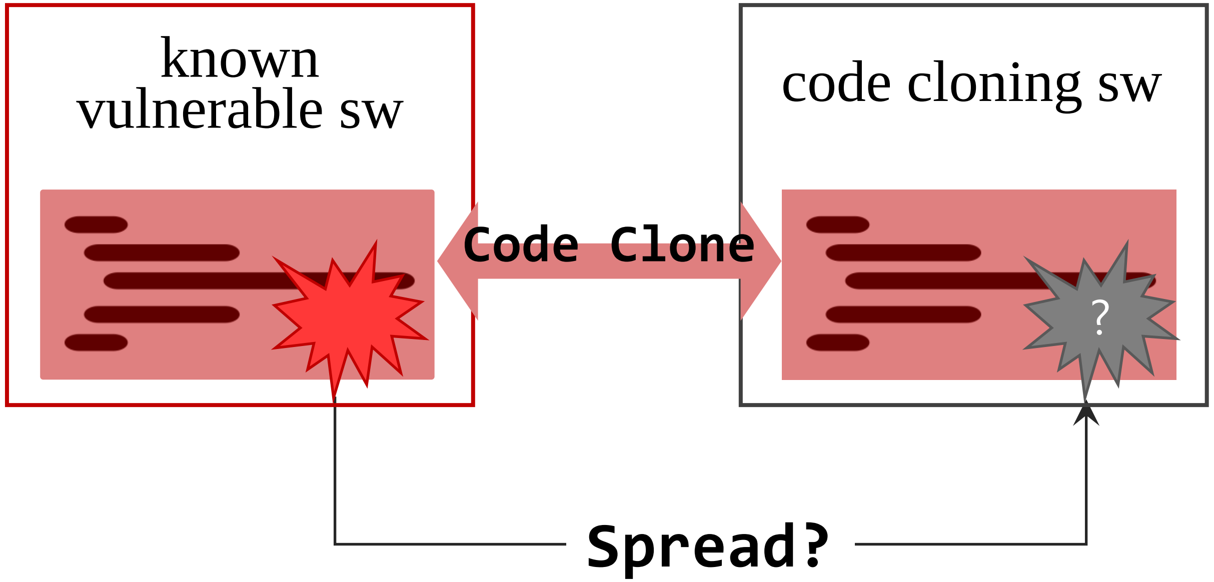
OctoPoCs: Automatic Verification of Propagated Vulnerable Code Using Reformed Proofs of Concept

IEEE/IFIP DSN 2021

Seongkyeong Kwon, Seunghoon Woo, Gangmo Seong, Heejo Lee *

Korea University, Korea

Code Clone and Vulnerability Propagation



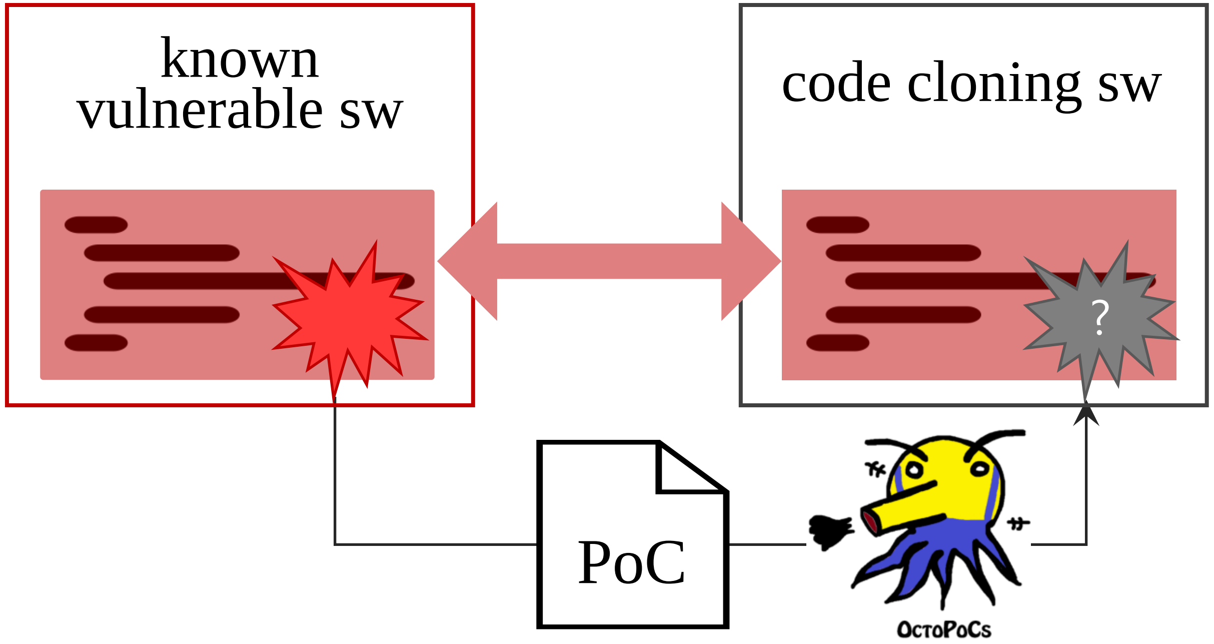
We should determine whether the vulnerable code can be triggered

Verify the security of cloned code

- Existing method to solve the security problem of code clone
  : *vulnerable code clone detection technique*
  -> cannot determine whether the vulnerable code can actually be triggered

- Existing method to check whether there is any vulnerability in a software
  : *fuzzing, AEG techniques*
  -> *"verify"* the specific vulnerable code ≠ *"discover"* possible vulnerability

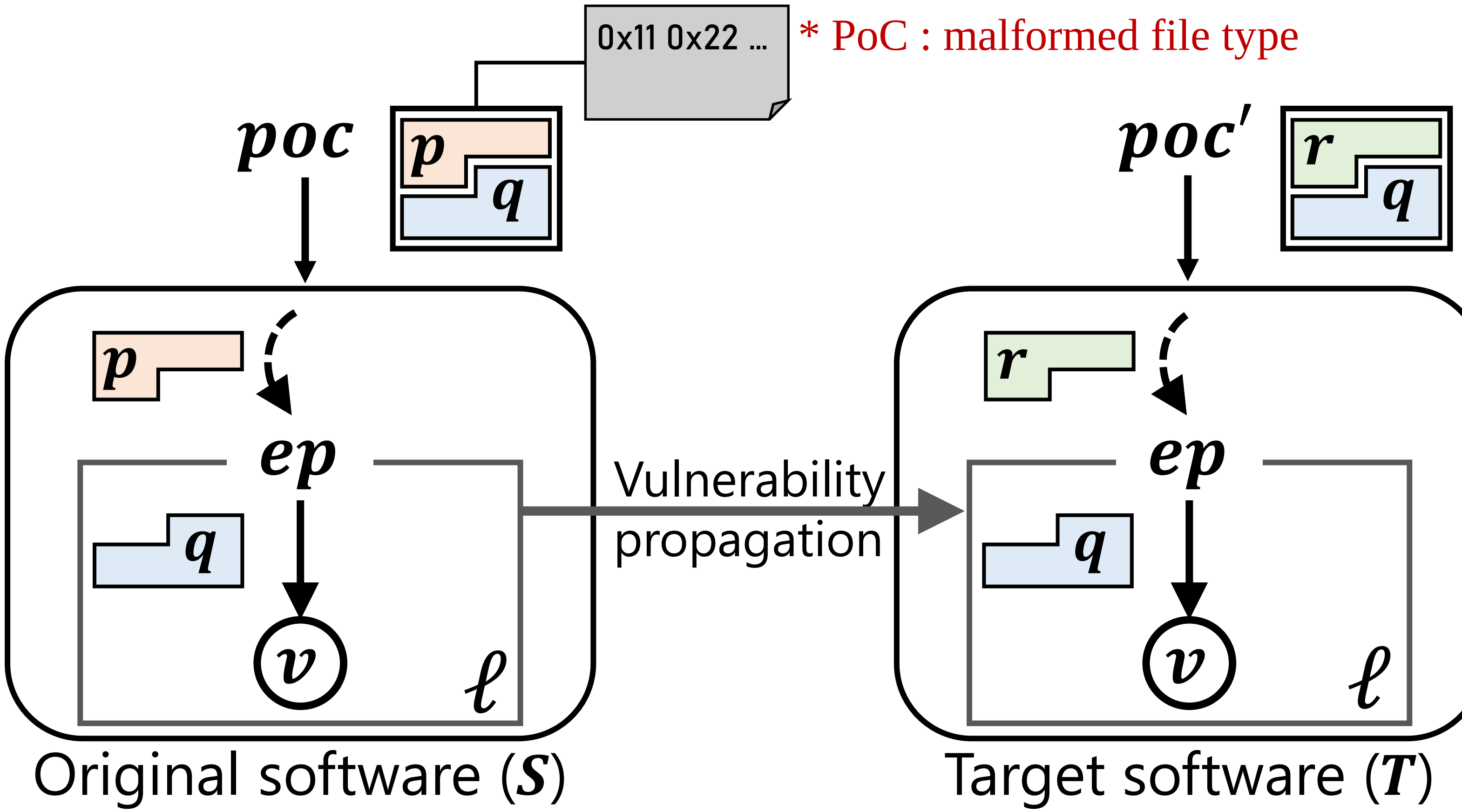
Effective verification of vulnerable clone



Use the proof of concept to verify whether
the shared vulnerable source code is triggerable in other software!

Structure of the Vulnerability

- $v \subset \ell \subset \mathrm{S}$
- $\ell \subset \mathrm{T}$
- $poc = p \oplus q$
- $poc' = r \oplus q$
- ep : entry point of ℓ

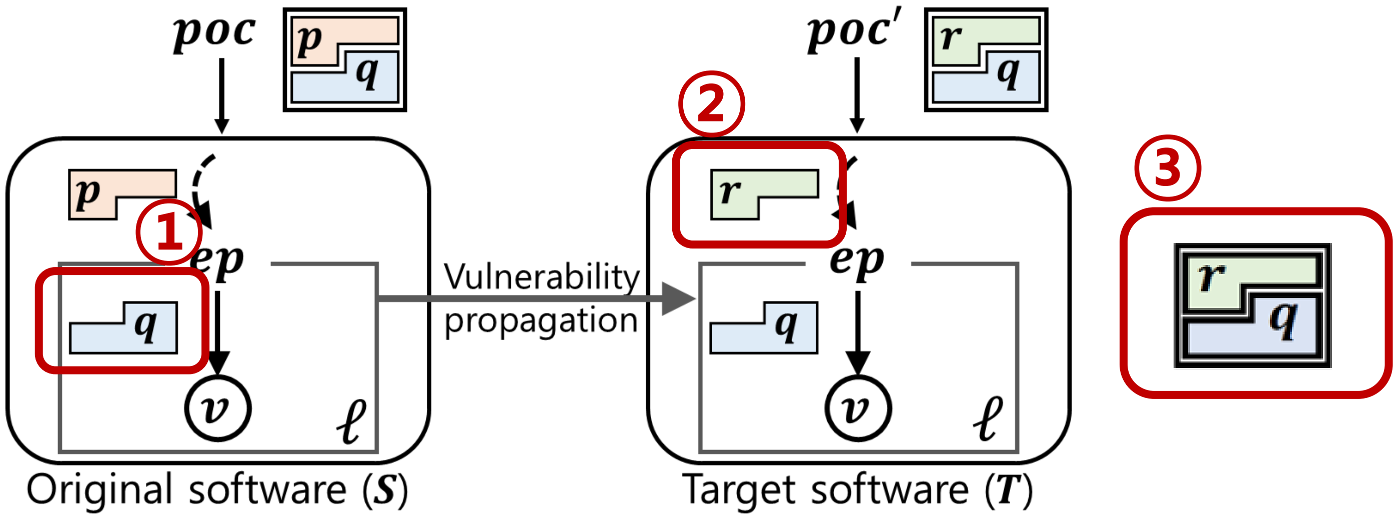


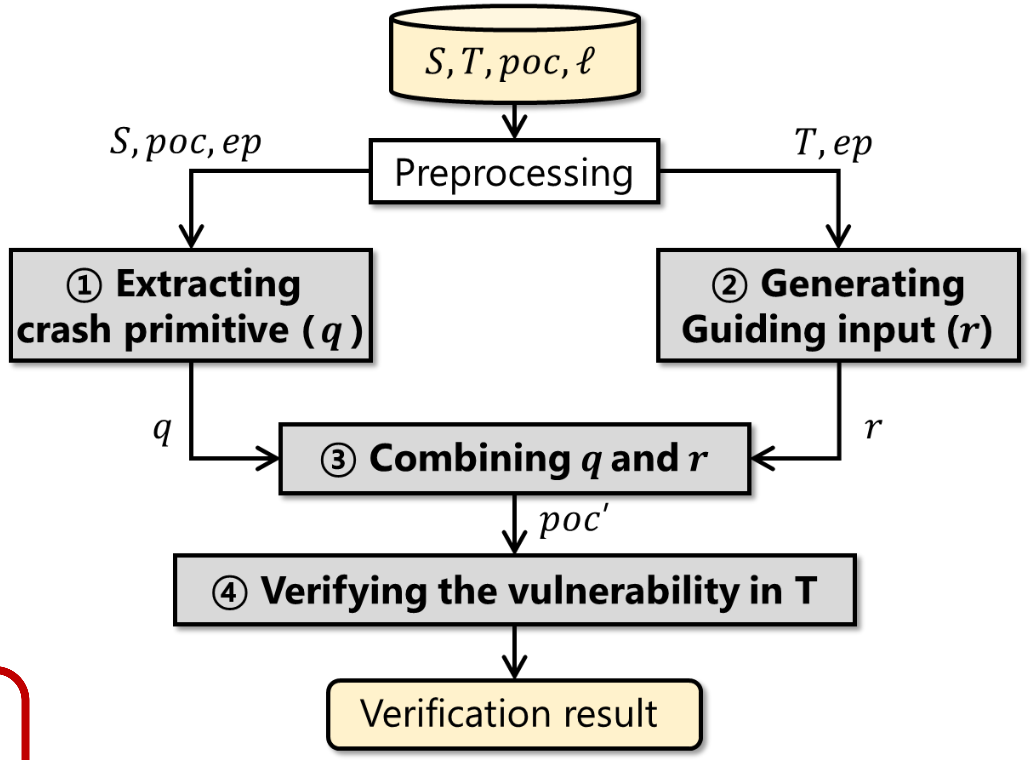
Process Overview

① Extracting crash primitive → q

② Generating guiding input → r

③ Combining q and r → poc`

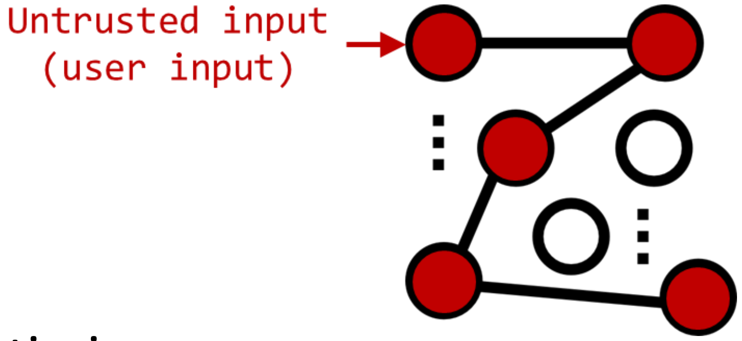




Extracting Crash Primitive in Input File

- **Crash Primitive : the reusable part, a set of bytes used in ℓ**
- Taint Analysis : tracking the flow of untrusted input
  - Check controllable memory area and register with input value

  → *finding which bytes are used in ℓ*

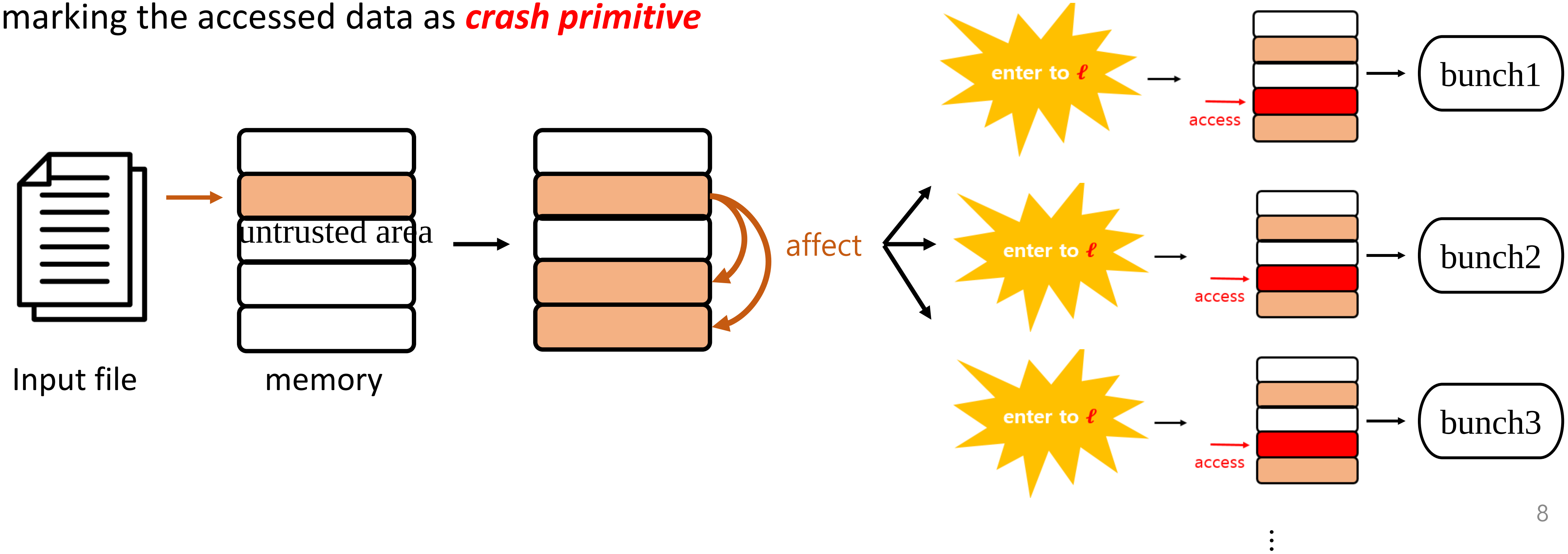


- We should consider the execution context! ex. byte usage timing

  → ***context-aware taint analysis***

Extracting Crash Primitive in Input File

1. Monitoring memory area where file data is uploaded – ***untrusted area***
2. Tracking if *read operation* occurs in untrusted area
3. Marking all memory address and registers affected by untrusted value - with *file offset*
4. After entering ℓ, if processor access to untrusted area, marking the accessed data as ***crash primitive***



Generating Guiding Input

- **Guiding Input : bytes that guide the execution flow to ℓ**
  - satisfy several conditions for the path to ℓ

- Symbolic Execution : software analysis technique that use symbolic value to execute a program

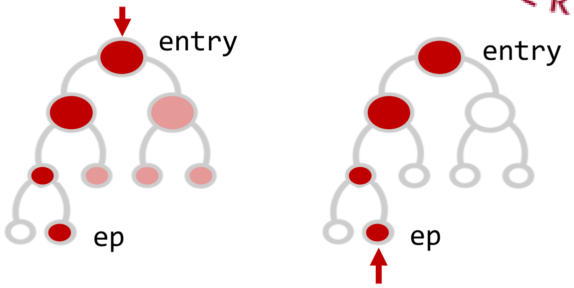
  → *get constraints of path to ℓ and solve*

- To avoid the path explosion problem, take advantage of knowing destination

  → ***backward path finding, directed symbolic execution***

Generating Guiding Input

- Backward Path Finding
  1. Generate CFG(Control Flow Graph) : to know the path to reach ℓ
  2. Find paths *from ℓ to entry point* to reduce computing resources



<BlockID 0x414535 0x414535.child=[0x414548, 0x41455B]>
<BlockID 0x414548 0x414548.child=[0x41467a, 0x4142bd]>
<BlockID 0x41455B 0x41455B.child=[0x414690]>
…

→

<BlockID 0x414535 0x414535.child=[0x414548, 0x41455B]>
<BlockID 0x414548 0x414548.child=[0x41467a, 0x4142bd]>
<BlockID 0x41455B 0x41455B.child=[0x414690]>
…

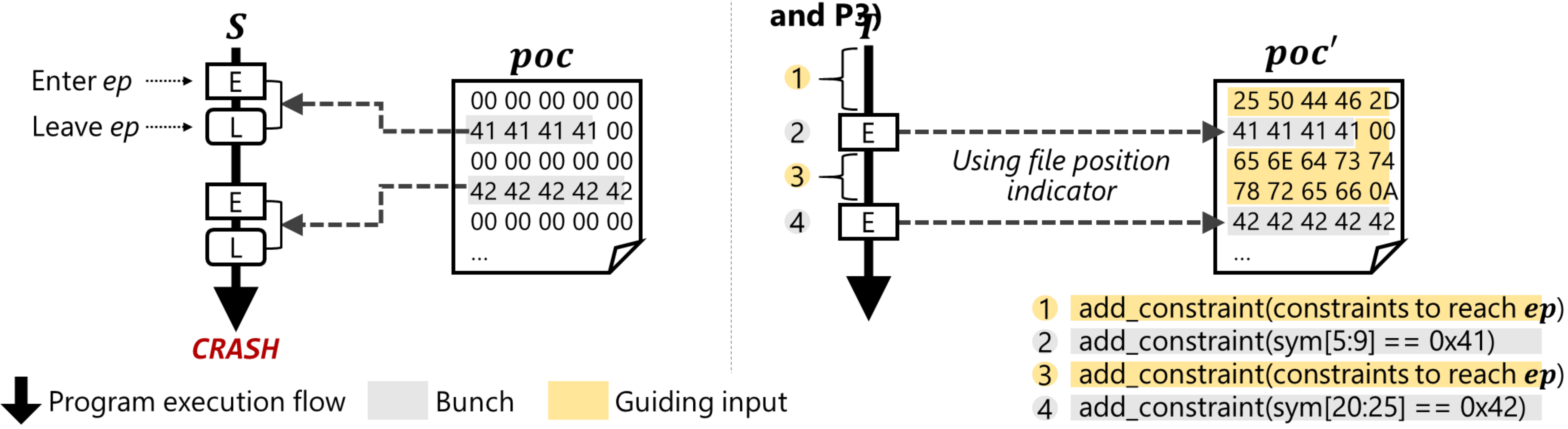
- Directed Symbolic execution
  3. Make symbolic file and upload to memory
  4. Execute with the symbolic file along the path

     - active state, loop state, loop-dead state, program-dead state
  5. After executing, solve the constraints

Combining

Extracting crash primitive phase (P1)

Generating guiding input & combining phases (P2 and P3)



Evaluation

Type	Idx	*S* Name	*S* Version	*T* Name	*T* Version	Vulnerability ID	Vulnerability Type†	*poc′*	Verification
Type-I	1	JPEG-compressor	N/A	libgdx	1.9.10	CVE-2017-0700	No-CWE	O	O
	2	JPEG-compressor	N/A	zxing	@0a32109	CVE-2017-0700	No-CWE	O	O
	3	pdftops (Poppler)	0.59	pdftops (Xpdf)	4.02	CVE-2017-18267	CWE-835	O	O
	4	avconv	12.3	ffmpeg	1.0	CVE-2018-11102	CWE-119	O	O
	5	tjbench (libjpeg-turbo)	2.0.1	tjbench (mozjpeg)	@0xbbb7550	CVE-2018-20330	CWE-190	O	O
	6	pdfalto	0.2	pdfinfo (Xpdf)	4.0.0	CVE-2019-9878	CWE-119	O	O
Type-II	7	ghostscript	9.26	opj_dump	2.1.1	ghostscript-BZ697463	No-CWE	O	O
	8	opj_dump	2.1.1	MuPDF	1.9	ghostscript-BZ697463	No-CWE	O	O
	9	gif2png	2.5.8	gif2png (artificial)	N/A	CVE-2011-2896	CWE-119	O	O
Type-III	10	tiffsplit	4.0.6	opj_compress	2.3.1	CVE-2016-10095	CWE-119	X	O
	11	tiffsplit	4.0.6	libsdl2	2.0.12	CVE-2016-10095	CWE-119	X	O
	12	tiffsplit	4.0.6	libgdiplus	6.0.5	CVE-2016-10095	CWE-119	X	O
	13	ghostscript	9.26	opj_dump	2.2.0	ghostscript-BZ697463	No-CWE	X	O
	14	pdfalto	0.2	pdftops (Xpdf)	4.1.1	CVE-2019-9878	CWE-119	X	O
Failure	15	pdf2htmlEX	0.14.6	pdfinfo (Poppler)	0.41.0	CVE-2018-21009	CWE-190	X	X

† CWE-119: buffer overflow, CWE-190: integer overflow, CWE-835: infinite loop

Evaluation

TABLE V: Elapsed time for verifying the propagated vulnerability in T in AFLFast, AFLGo, and OCTOPOCS.

S	*T*	AFLFast*	AFLGo*	OCTOPOCS
		Elapsed time (s)		
ghostscript	opj_dump	N/A	N/A	9.67
opj_dump	MuPDF	N/A	Error†	75.4
gif2png	gif2png (arti.)	201	N/A	558.46

∗: running 20 h, †: cannot executed due to the tool error.

Conclusion

- OCTOPOCS
  - verifying whether a vulnerability in propagated vulnerable code can still be triggered by using the reformed PoC

- context-aware analysis, directed symbolic execution
  - effectively reform PoC

- Limitations : loop-dead state, malformed file type

Thank you for your attention
Questions?

bible_kwon@korea.ac.kr