UNC CHARLOTTE
College of Computing and Informatics
Department of Software and Information Systems

Poster: Guide Me to Exploit: Assisted ROP Exploit Generation for ActionScript Virtual Machine

Fadi Yilmaz, Meera Sridhar, and Wontae Choi

Learning from Authoritative Security Experiment Results (LASER) 2020
Part II
December 8, 2020, Online

presented by
Fadi Yilmaz

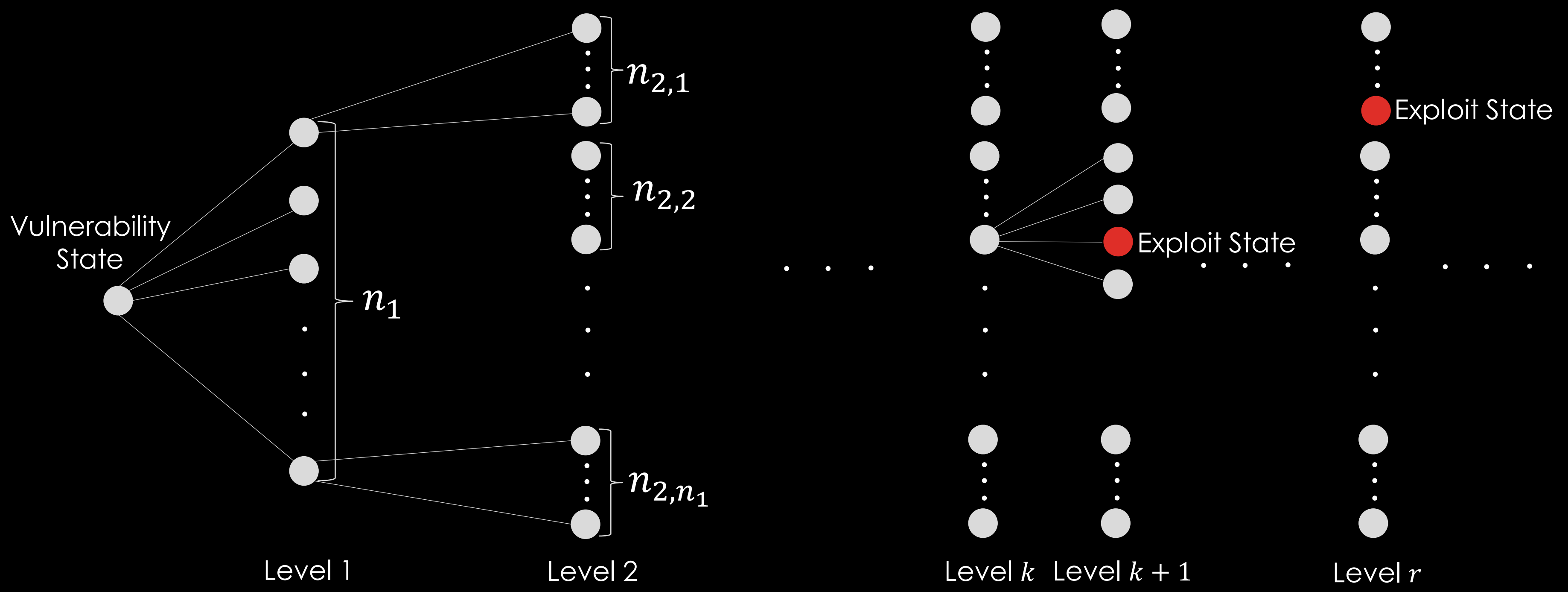
Work supported in part by National Science Foundation Grant No. 1566321

Motivation of Automated Exploit Generation (AEG)

- Monitoring the execution of exploit scripts is crucial
  - Underlying weaknesses of target applications
  - Unorthodox methods to exploit vulnerabilities

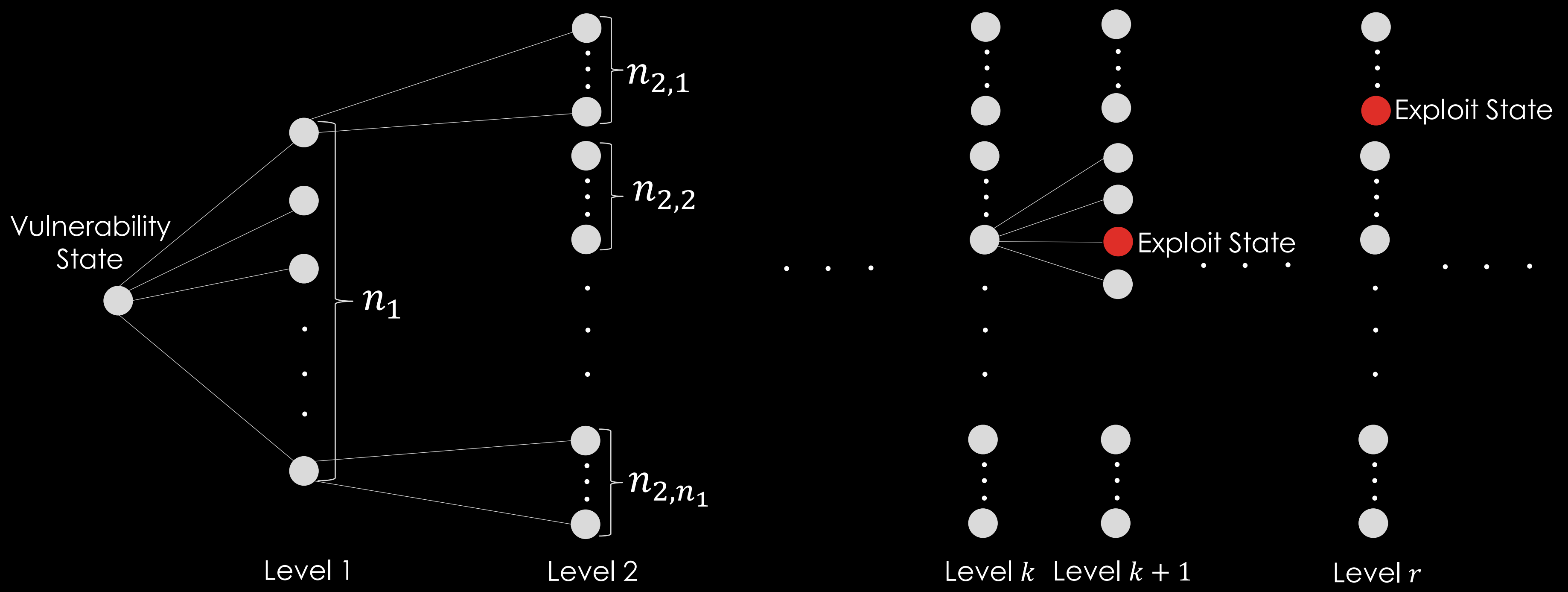
AEG

- Determining the *exploitability* [Younis et al. SQJ'16]
- Explores all possible execution paths [Avgerinos et al. NDSS'11]



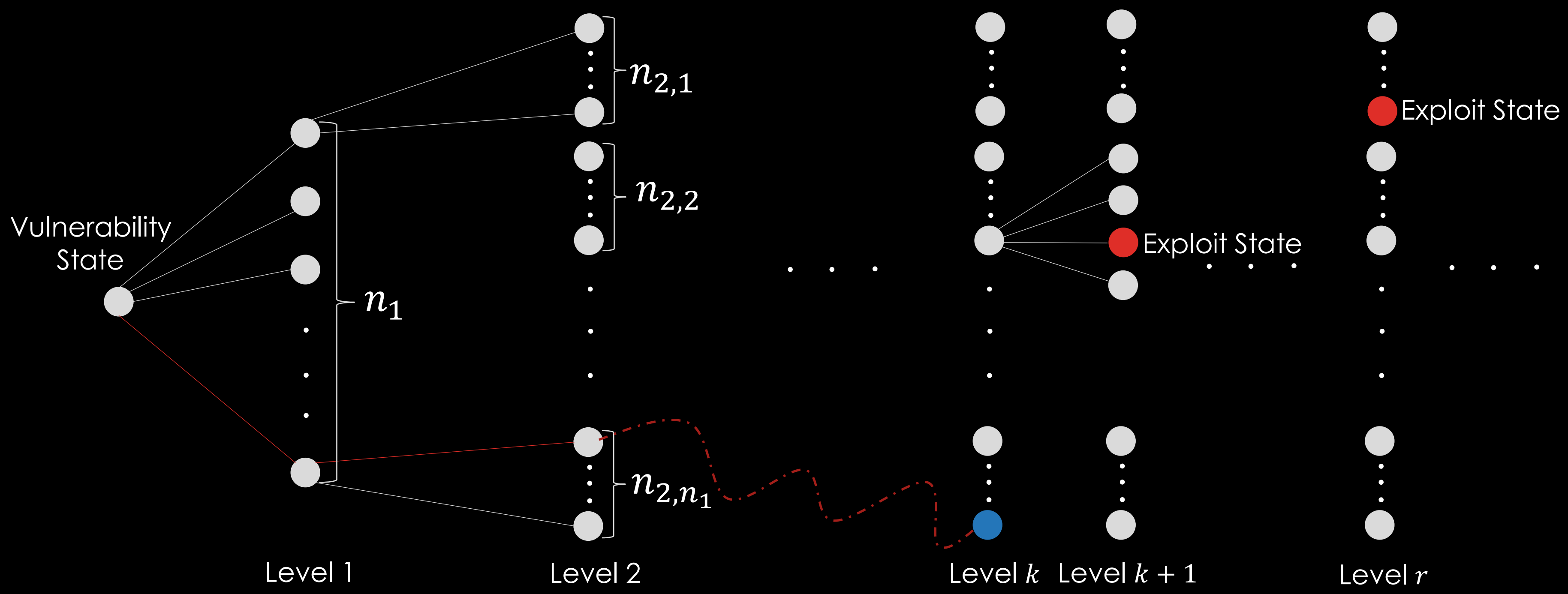
AEG Components

- Fuzzer [Miller et al. ACM'90, Jayaraman et al. NFM'09, Rawat et al. NDSS'17]
  - Explores only one execution path in one run



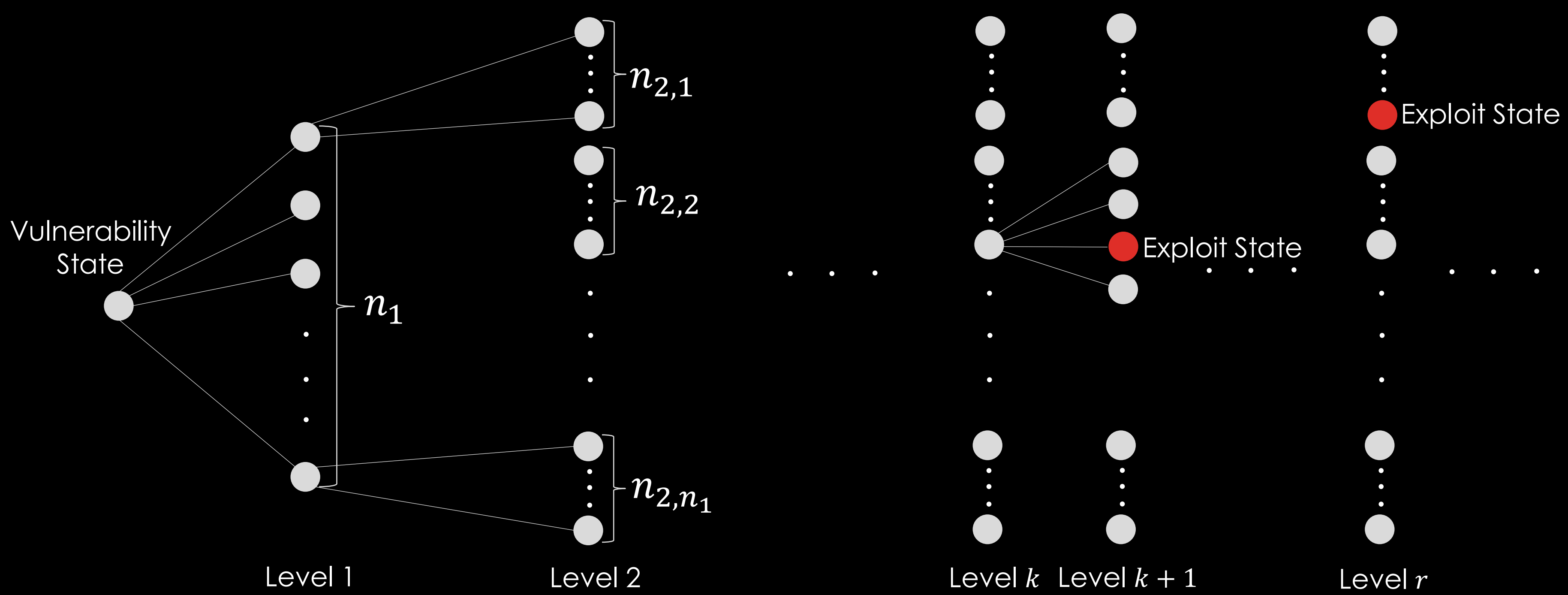
AEG Components

- Fuzzer
  - Explores only one execution path in one run



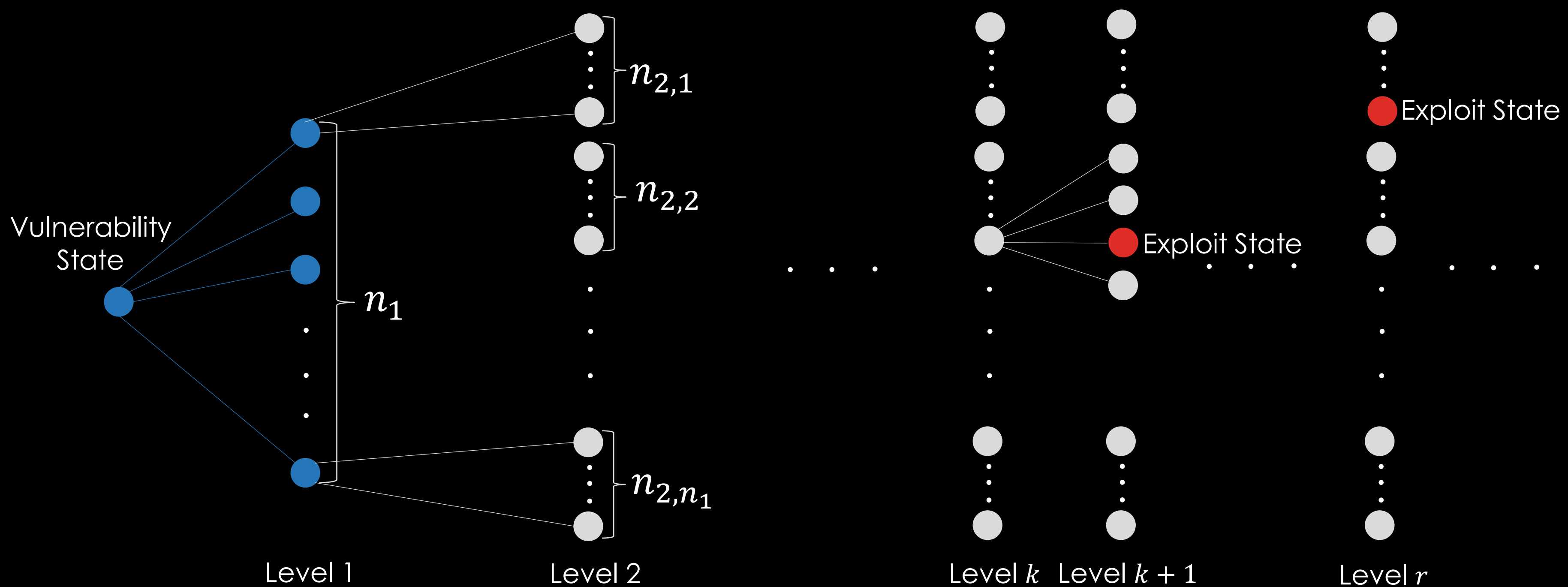
AEG Components

- Symbolic Execution [King et al. ACM'76]
  - Explores all execution paths symbolically in one run



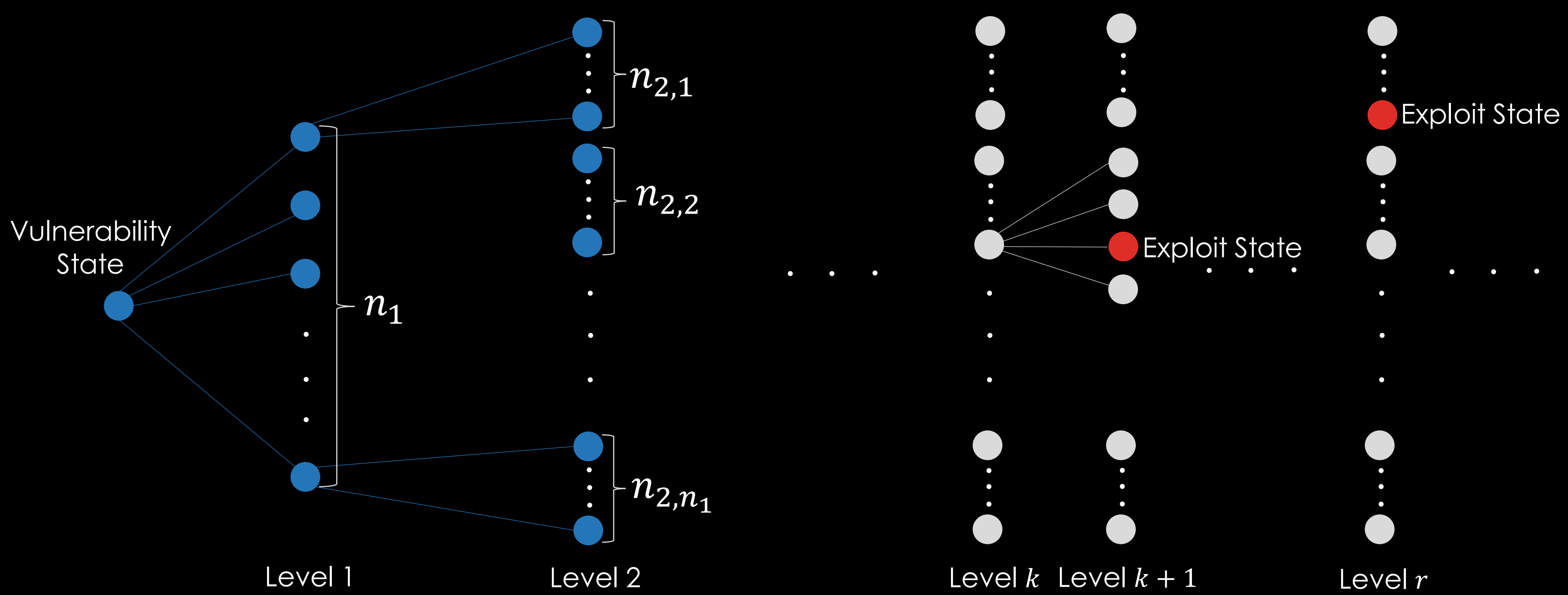
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run



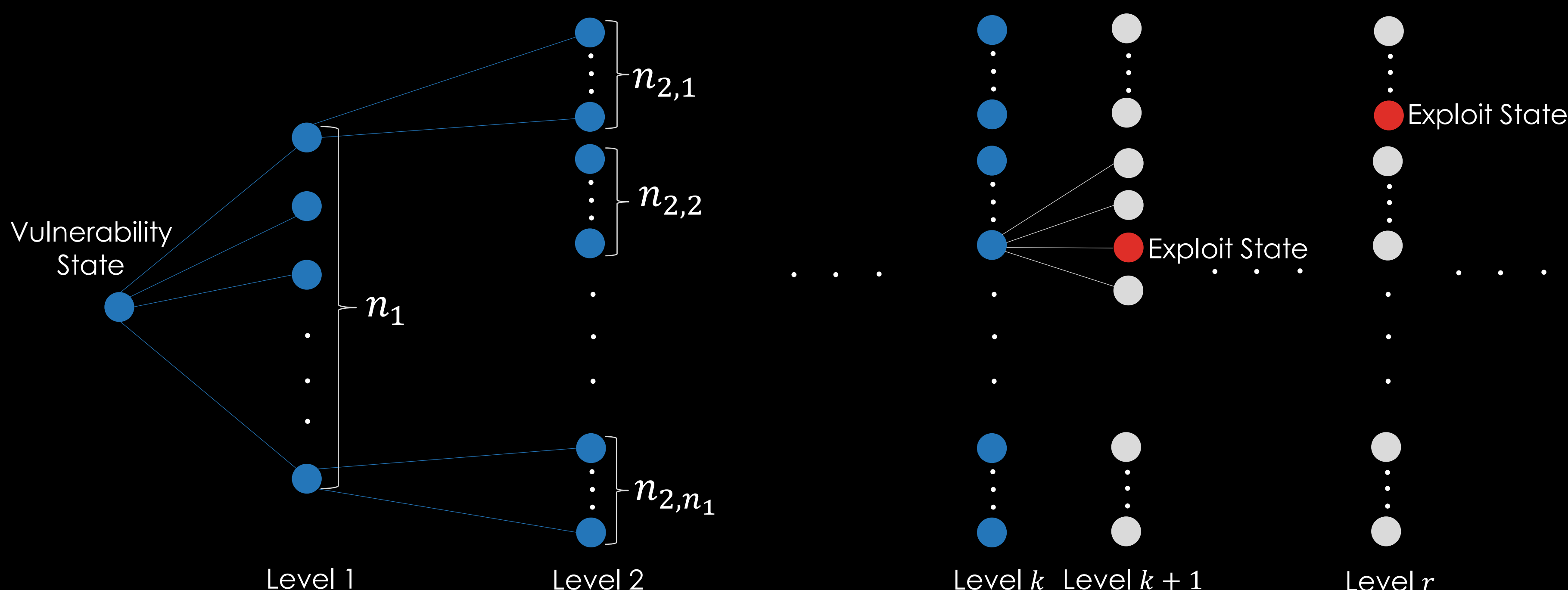
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run



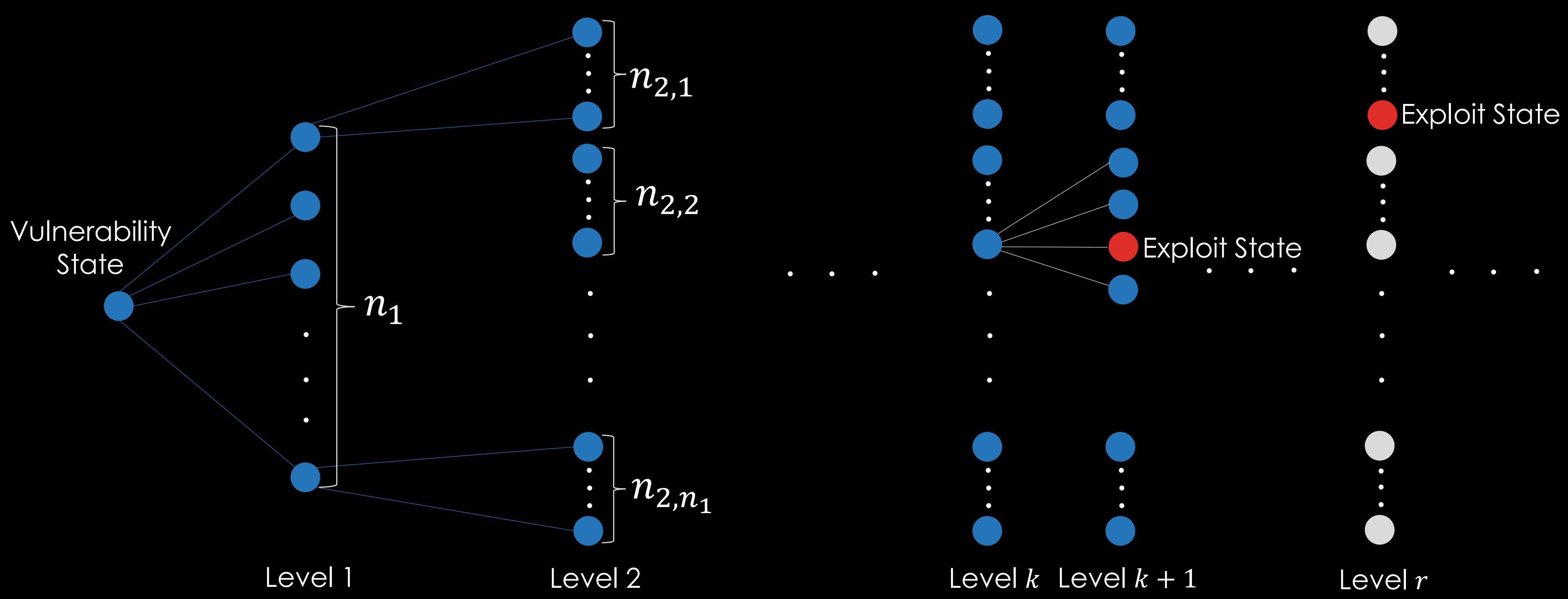
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run



AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run



AEG Components

+pros

Fast, easy to build

Fuzzer

Complex grammar rules for executables

Infinitesimal chance

-cons

+pros

Explores all execution paths in one run

Symbolic Execution

The path-explosion problem

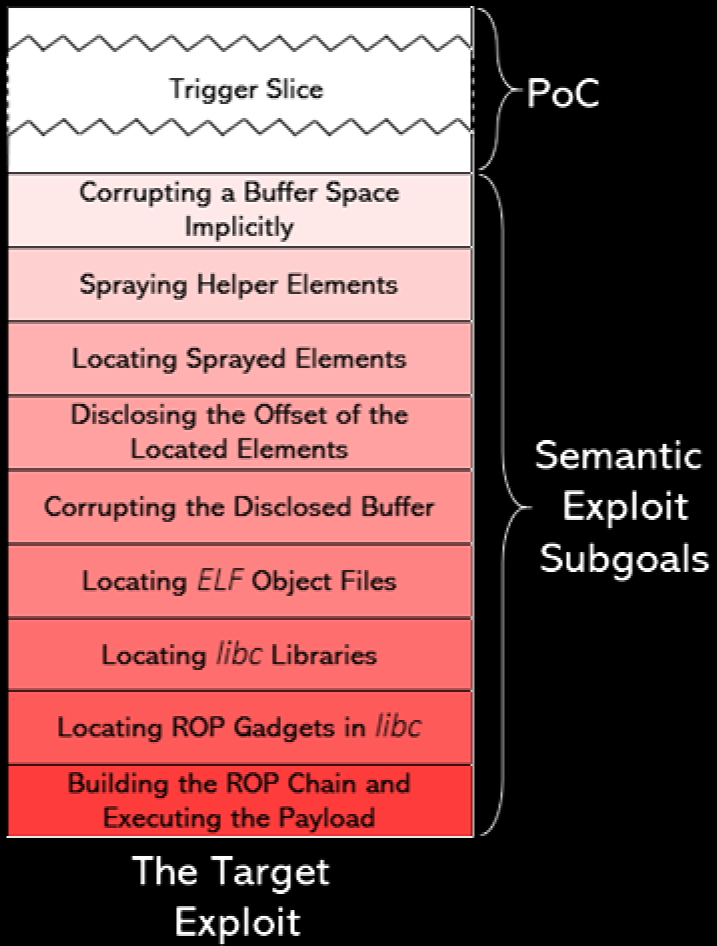
-cons

GuidExp : A Prototype Semi-Automatic AEG Tool

- The first *guided* (semi-automatic) exploit generation tool for the AVM implementations
- Does not rely on a fuzzer or a symbolic execution tool

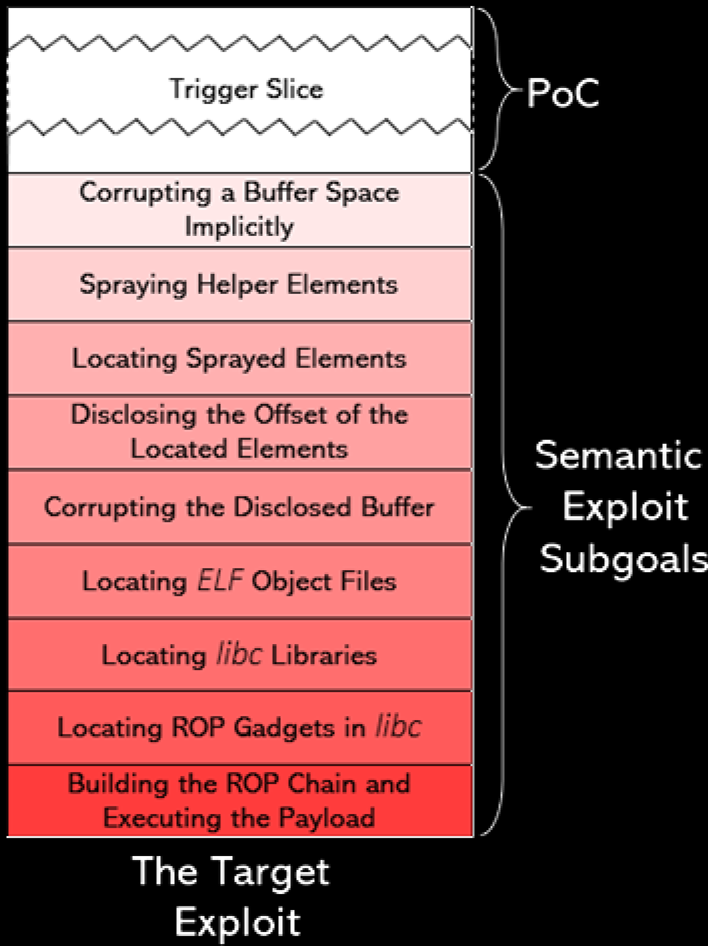
Intuition Behind Target Exploit Generation

- Structure of our target exploit
- Exploit pattern



Exploit Subgoals

- A search space
  - Set of instructions
- An invariant
  - The test



Experimental Results -I

- The difference is due to starting/closing of the Flash Player
  - It takes 85ms on average, equivalent to 89% of the time

Exploit Subgoal	Number of Generated Candidate Slices	Number of Executed Candidate Slices	Percentage of Executed Candidate Slices	Synthesizing Time (s)
Corrupting a Buffer Space Implicitly	2,396,744	12,229	0.51	9.35
Spraying Helper Elements	19,173,952	73,997	0.38	55.90
Locating Sprayed Elements	37,448	357	0.95	1.72
Disclosing the Offset of the Located Elements	55,345,757	282,392	0.51	138.26
Corrupting the Disclosed Buffer	4,793,488	21,591	0.45	17.03
Locating `ELF` Object Files	19,173,952	81,545	0.42	57.12
Locating `libc` Libraries	55,345,757	278,385	0.50	138.05
Locating Executable Segment	76,695,808	379,587	0.49	199.78
Locating Gadgets and Building the ROP Chain	435,848,049	1,648,451	0.37	240.92
				Total: 858.13 (14m 18.13s)
Corrupting a Buffer Space Implicitly	2,396,744	29,167	1.21	605.58
Spraying Helper Elements	19,173,952	210,225	1.09	3,895.64
Locating Sprayed Elements	37,448	769	2.05	12.76
Disclosing the Offset of the Located Elements	55,345,757	508,339	0.91	6,845.86
Corrupting the Disclosed Buffer	4,793,488	41,342	0.86	963.86
Locating `ELF` Object Files	19,173,952	201,852	1.05	3,364.89
Locating `libc` Libraries	55,345,757	459,336	0.82	6,276.25
Locating Executable Segment	76,695,808	706,031	0.92	9,546.07
Locating Gadgets and Building the ROP Chain	435,848,049	2,954,400	0.67	11,512.47
				Total: 43,023.38 (11h 57m 03.38s)

Experiment

- Generating exploit scripts for different vulnerabilities with the closed-source debugger

Selected Vulnerabilities	Synthesizing Time	Flash Player Version
CVE-2015-5119	**11h 57m 03.38s**	**v11.2.202.262**
CVE-2013-0634	12h 09m 14.50s	v11.2.202.262
CVE-2014-0502	12h 54m 15.19s	v11.2.202.262
CVE-2014-0515	12h 51m 26.67s	v11.2.202.262
CVE-2014-0556	12h 08m 35.29s	v11.2.202.262
CVE-2015-0311	11h 56m 19.10s	v11.2.202.262
CVE-2015-0313	12h 20m 47.98s	v11.2.202.442
CVE-2015-0359	11h 05m 05.61s	v11.2.202.262
CVE-2015-3090	12h 01m 33.16s	v11.2.202.262
CVE-2015-3105	13h 25m 46.80s	v11.2.202.262
CVE-2015-5122	12h 07m 02.59s	v11.2.202.262

Experimental Results -II

- Generating exploit scripts for different vulnerabilities with the closed-source debugger

Selected Vulnerabilities	Synthesizing Time	Flash Player Version
CVE-2015-5119	**11h 57m 03.38s**	**v11.2.202.262**
CVE-2013-0634	12h 09m 14.50s	v11.2.202.262
CVE-2014-0502	12h 54m 15.19s	v11.2.202.262
CVE-2014-0515	12h 51m 26.67s	v11.2.202.262
CVE-2014-0556	12h 08m 35.29s	v11.2.202.262
CVE-2015-0311	11h 56m 19.10s	v11.2.202.262
CVE-2015-0313	12h 20m 47.98s	v11.2.202.442
CVE-2015-0359	11h 05m 05.61s	v11.2.202.262
CVE-2015-3090	12h 01m 33.16s	v11.2.202.262
CVE-2015-3105	13h 25m 46.80s	v11.2.202.262
CVE-2015-5122	12h 07m 02.59s	v11.2.202.262

Experimental Setup

- Focus
  - General numbers
  - Applicability of the tool
- Goal
  - To demonstrate that our tool is actually useful

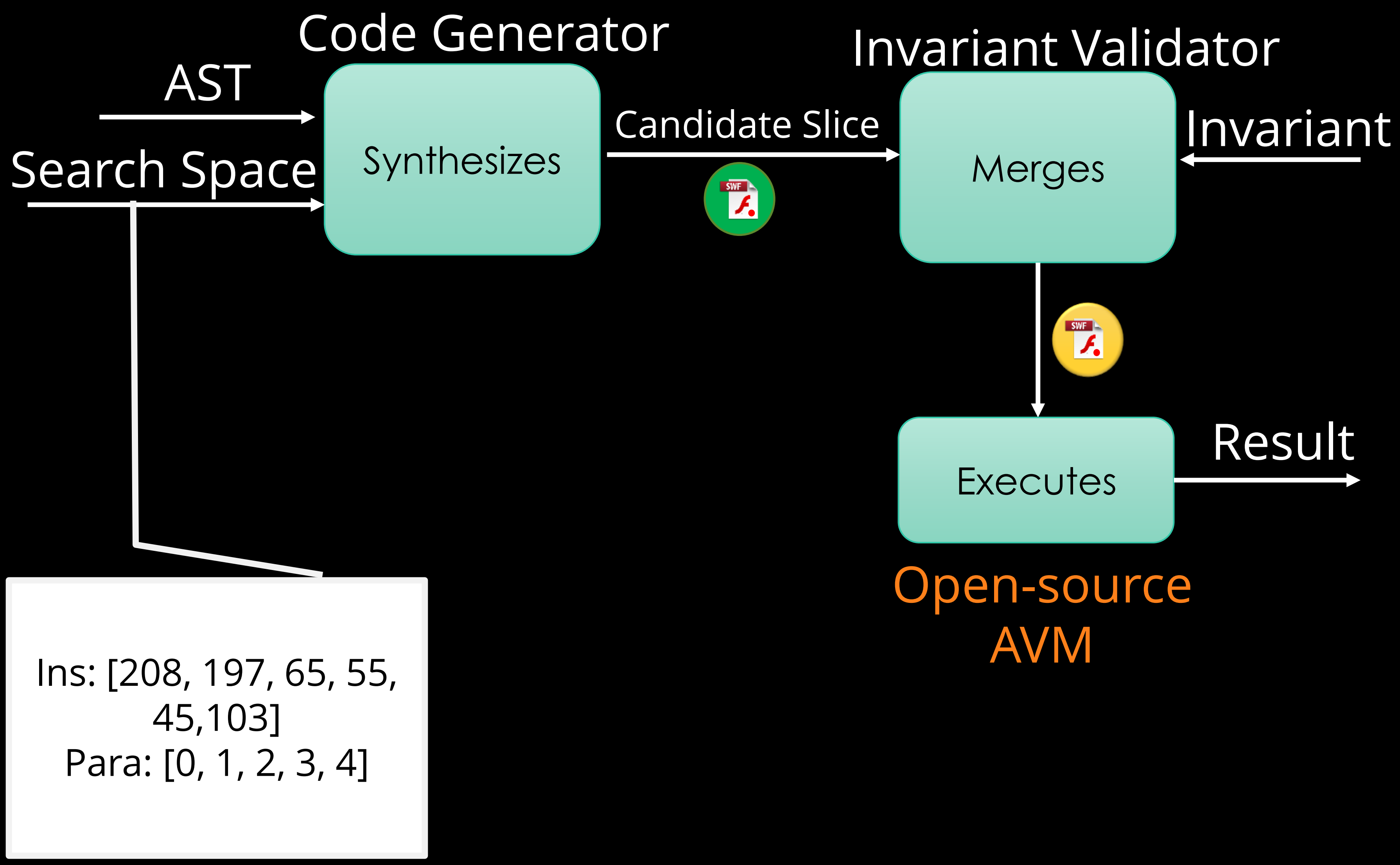
Experimental Setup – Cont.

- Two set of experiments
  - Executing candidate slices with open-source AVM
  - Executing candidate slices with closed-source AVM

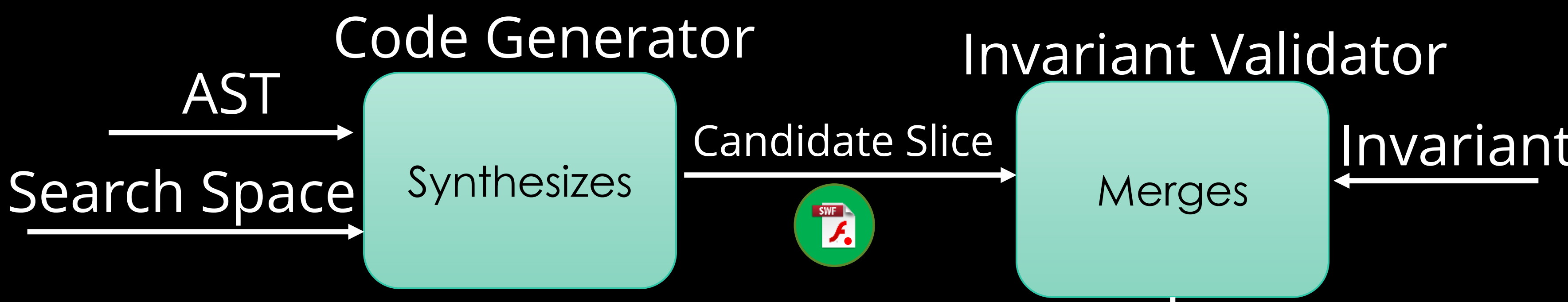
Experimental Setup – Cont.

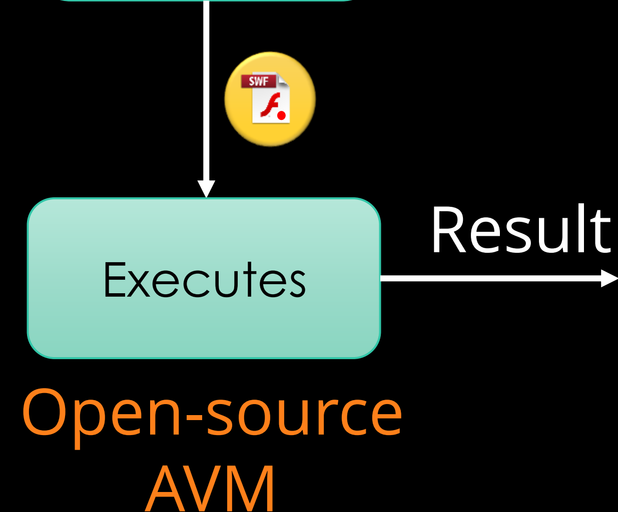
- Used artifacts
  - The golden example
    - The only vulnerability in the open-source AVM
    - To explain difference between targeting open-source and closed-source AVM
  - Eleven vulnerabilities collected for a closed-source AVM
    - Includes the golden example

Execution Flow Recap



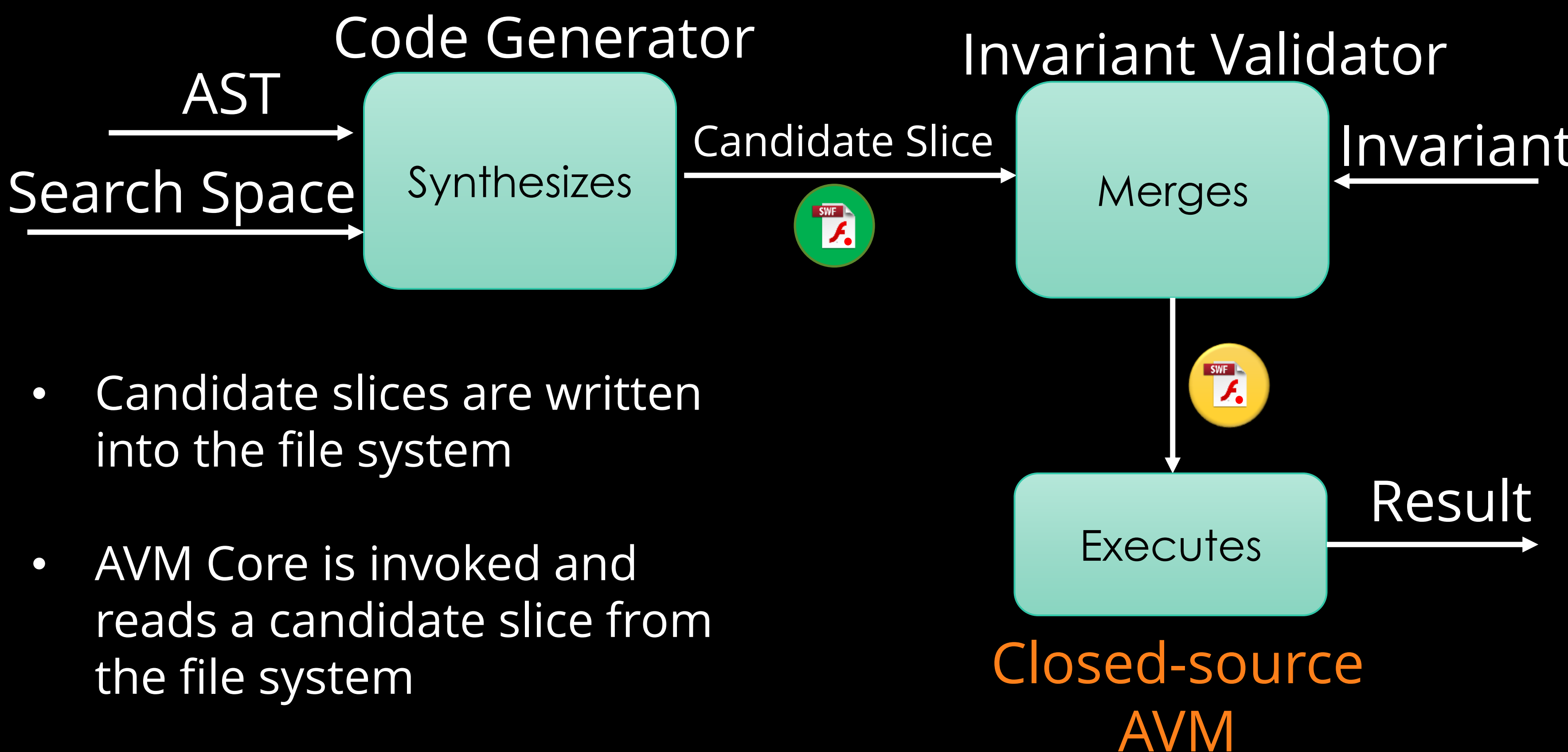
Execution Flow Recap – Cont.





- Candidate slices are written into the file system
- AVM Core is invoked and reads a candidate slice from the file system
- The result is written into a file system and read from the file system

Execution Flow Recap – Cont.



- Candidate slices are written into the file system
- AVM Core is invoked and reads a candidate slice from the file system
- The result is written into a file system and read from the file system

Initial Development

- Measure the initial performance
  - Memory
  - Running time
  - Interaction Cost

- To get something fast and lean enough to be used

Initial Development

- Used Artifact
  - The golden example
  - A single vulnerability from an open-source AVM
    - CVE-2015-5119
      - Details are in our paper
    - Real example
    - Not too complicated
    - Still not too simple

Development Cycle

- Implement a prototype
- Run it on the benchmark
- Evaluate the numbers
- Identify bottlenecks
- Optimize
- Go back to step (1)

Development Cycle -I

- Everything was written into HDD
  - Huge bottleneck
- This is not a part of our algorithm!
- Easy to solve!
  - Ask for an SSD!
    - Not good enough
      - Use VM

Development Cycle -II

- The number of execution paths to explore is too big!
- Solution
  - Adding search space limitation

Development Cycle -III

- The search may last infinite
- Tested various search target prioritization techniques (DFS or BFS or Random)
- Final decision: BFS
  - Level is limited

Development Cycle -IV

- The search still takes too much time
  - Number of candidate slice is more than billions
- Optimize
  - Lots of type errors happened
    - Feedback optimization
  - Stack simulation
  - Tiling

Initial Development – The Bottom Line

- Golden Example
  - Good Part
    - Iterate extremely fast
    - Identify all the small details of algorithms and artifacts
  - Danger
    - Development can be biased
  - Mitigation
    - Use more than one golden example

Actual Evaluation

- What we did
  - Applied our technique to all these examples
  - Showed that everything passes
- What we observed
  - Not biased with golden example
  - Performance of our tool with closed-source VM is not as good as it is with open-source VM

Actual Evaluation – The Bottom Line

- We were lucky that we started with a vulnerability in an open-source VM
  - With a closed-source VM, our initial development process could be infeasible

- Generalizing different configurations can be challenging

Manual Intervention

- Evaluating manual intervention is not in our focus
- Our focus is to move from “unable” to “able”
- This is the future work!

Artifacts Borrowed from the Community

- Synthesizes a ROP exploit for given AVM vulnerabilities
- AVM vulnerabilities
  - Exploit databases
    - exploit-db.com
    - Google's Project Zero*
  - Tech Reports
  - We synthesized different exploits

*https://bugs.chromium.org/

Artifacts Borrowed from the Community

- Synthesizes a ROP exploit for given AVM vulnerabilities
- ROP chain
  - ROPgadget*
  - Locates and build the ROP chain
  - Execute ‘int 0x80’
- We copied the idea

*Jonathan Salwan - https://github.com/JonathanSalwan/ROPgadget

Intermediate Results

- Development Cycle
  - Many iterations
  - Many results

- Gradually getting faster tool
  - Start with months
  - Down to 15 minutes

Intermediate Results - Optimizations

- Multi-threading
  - Three threads
  - From months to weeks
- Stack Simulation
  - Almost hundred times faster
  - From weeks to hours

Intermediate Results - Optimizations

- Run-time Errors
  - From hours to minutes
  - There are thousands of different run-time error messages*
  - Not all of them is raised
  - Not all of them is useful

*https://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/runtimeErrors.html/

What can be learned from your methodology and your experience using your methodology?

Any Failed Attempts

- Not really
  - Aimed to implement more powerful system
  - More optimization techniques

Did you attempt to replicate or reproduce results of earlier research as part of your work?

Future Works

- Need to measure
  - How much human interaction is required
  - How much human expertise is required
    - Can a newbie use the tool?
  - How much effort does our tool save for a seasoned developer

Future Works

- User-study
- Two dimensions of expertise
  - Exploits
  - ActionScript language
- Three level of expertise
  - Newbie
  - Intermediate
  - Seasoned

Thank You

Fadi Yilmaz
UNC Charlotte
fyilmaz@uncc.edu

Meera Sridhar
UNC Charlotte
msridhar@uncc.edu

Wontae Choi
wtchoi.kr@gmail.com

Key References

[Younis et al. SQJ'16] Awad Younis, Yashwant K Malaiya, and Indrajit Ray. 2016. Assessing vulnerability exploitability risk using software properties. Software Quality Journal 24, 1 (2016), 159–202.

[Avgerinos et al. NDSS'11] Thanassis Avgerinos, Sang Kil Cha, Brent Lim Tze Hao, and David Brumley. 2011. AEG: Automatic Exploit Generation. In Proceedings of The Network and Distributed System Security Symposium (NDSS).

[Miller et al. ACM'90] Barton P Miller, Louis Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. Commun. ACM 33, 12 (1990), 32–44.

[Jayaraman et al. NFM'09] Karthick Jayaraman, David Harvison, and Adam Kiezun Vijay Ganesh. 2009. jFuzz: A concolic whitebox fuzzer for Java. In Proceedings of the First NASA Formal Methods Symposium (NFM).

[Rawat et al. NDSS'17] Sanjay Rawat, Vivek Jain, Ashish Kumar, Lucian Cojocar, Cristiano Giuffrida, and Herbert Bos. 2017. VUzzer: Application-aware Evolutionary Fuzzing. In Proceedings of the Network and Distributed System Security Symposium (NDSS), Vol. 17. 1–14.