Software Security Analysis Methods

System Programing Laboratory - SPLab
Yerevan State University

Sevak Sargsyan
sevaksargsyan@ispras.ru

SPLab Team

Founded by academician ***Victor Ivannikov*** in 2009**.**

Members (15)

1. Yerevan State University
2. National Polytechnic University
3. Russian-Armenian University
4. Moscow State University, Yerevan branch

Summer courses for new members selection (2, 3, 4 grade):

1. Compilers: Design and Implementation
2. Software Security
3. Advanced C++ and Algorithms

SPLab Projects

1. Compiler optimizations
2. Code obfuscation
3. Code clone detection
4. ***Code static analysis***
5. ***Code dynamic analysis***

Why software security is critical?

1. Privacy (*credit card*, *photos*, *private documents/information*, *etc.*)
2. Safety (*airplanes*, *trains*, *self driving cars*, *satellites*, *smart homes*, *etc.*)
3. Cyber attacks/wars
4. Cyber spying

Why there are bugs?

1. Unqualified software engineers
2. High complexity
3. Untested software (hard deadlines)
4. ***Defects in compilers***

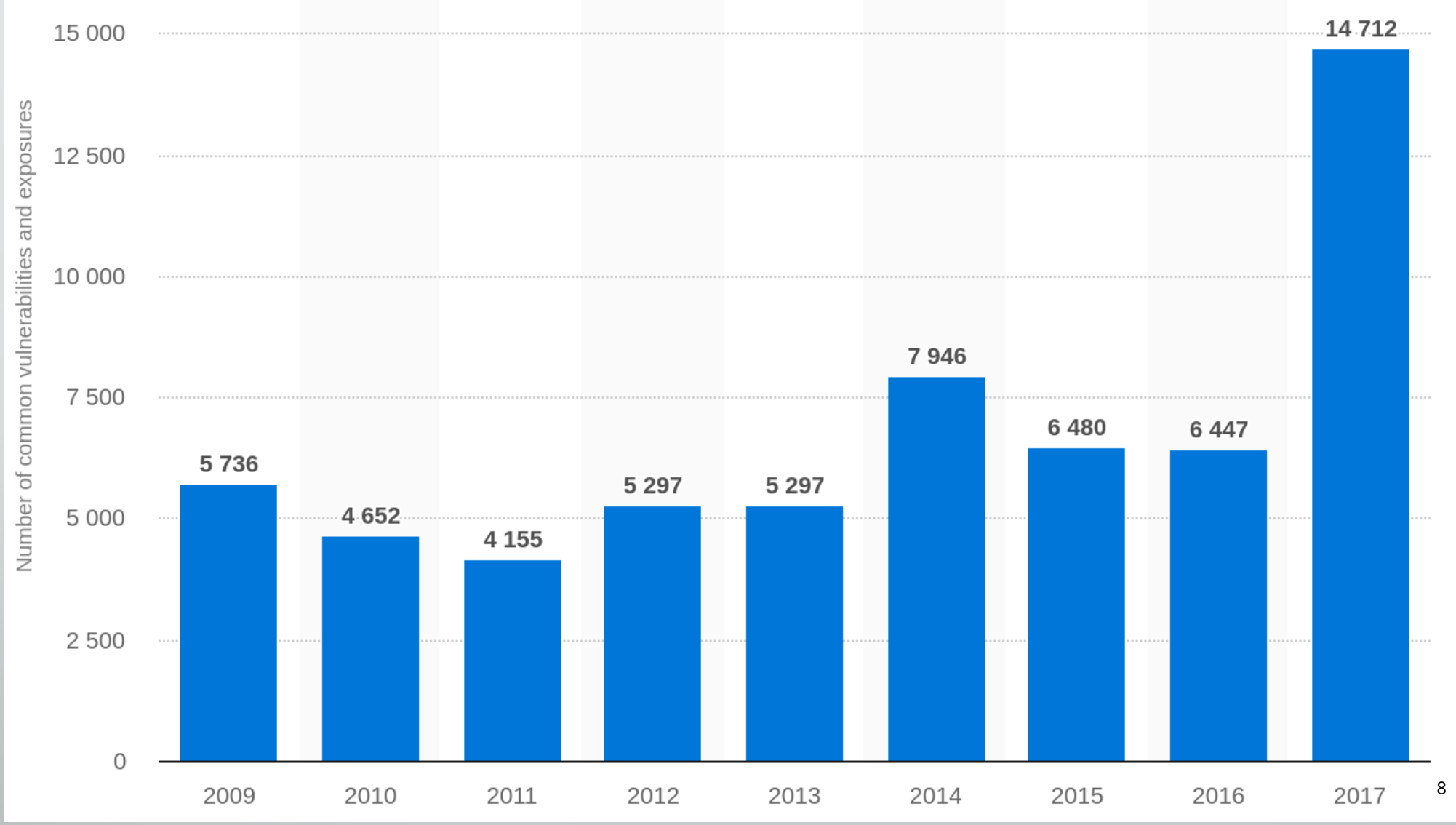
Software bugs ratio

1. Industry Average: 15 - 50 defects per 1000 lines of code
2. Microsoft Applications: 10 - 20 defects per 1000 lines of code
3. Cleanroom development (Harlan Mills): ~3 defects per 1000 lines of code

Widely known bugs that were exploited

1. ***OpenSSL Heartbleed,*** allows stealing protected by SSL/TSL information
2. ***Ransomware Attacks (WannaCry)***, encrypt files and demand payment, ~400.000 machines were infected
3. ***Win32k Elevation***, gain privileged control of machine
4. ***Windows 10 WiFi Sense Contact Sharing***, get access to your contacts without authorization
5. ***Microsoft Windows Journal Vulnerability***, allows remote code execution

Vulnerabilities statistics



Security Development Lifecycle

Security development lifecycle:

1. ***Static analysis***
2. ***Dynamic analysis***

Microsoft:

1. ***FxCop*** (Static analysis)
2. ***SAGE*** (Fuzzing + Symbolic execution)
3. ***Neural Fuzzing***
4. **……**

Google:

1. ***OSS-Fuzz***
2. ***BinNavi*** (Static analysis platform)
3. **…….**

Apple:

1. ***libFuzz*** (Functions fuzzing)
2. ***LLVM Sanitize*** (Compile time instrumentation, run time detection)
3. **…….....**

Tools used for software analysis

Static analysis tools:

1. ***Coverity*** (set of checkers on internal representation)
2. ***FindBugs*** (set of checkers on internal representation)
3. ***Svace*** (set of checkers on internal representation)
4. ***Binnavi*** (binary analysis platform)
5. **…….**

Dynamic analysis tools:

1. ***GDB*** (debugger)
2. ***LLVM Sanitizer*** (compiler)
3. ***AFL***, ***Syzcaller*** (fuzzing tool)
4. ***Valgrind*** (binary translation)
5. ***Anxiety*** (dynamic symbolic execution)
6. **…….**

SPLab - Code Static Analysis

1. Source code clones detection
2. Binary code clone detection (viruses detection, etc.)
3. Buffer overflows detection
4. Format string detection (C/C++ printf)
5. Use after free detection (C/C++ new/delete)
6. Old/buggy software components/library detection
7. ……

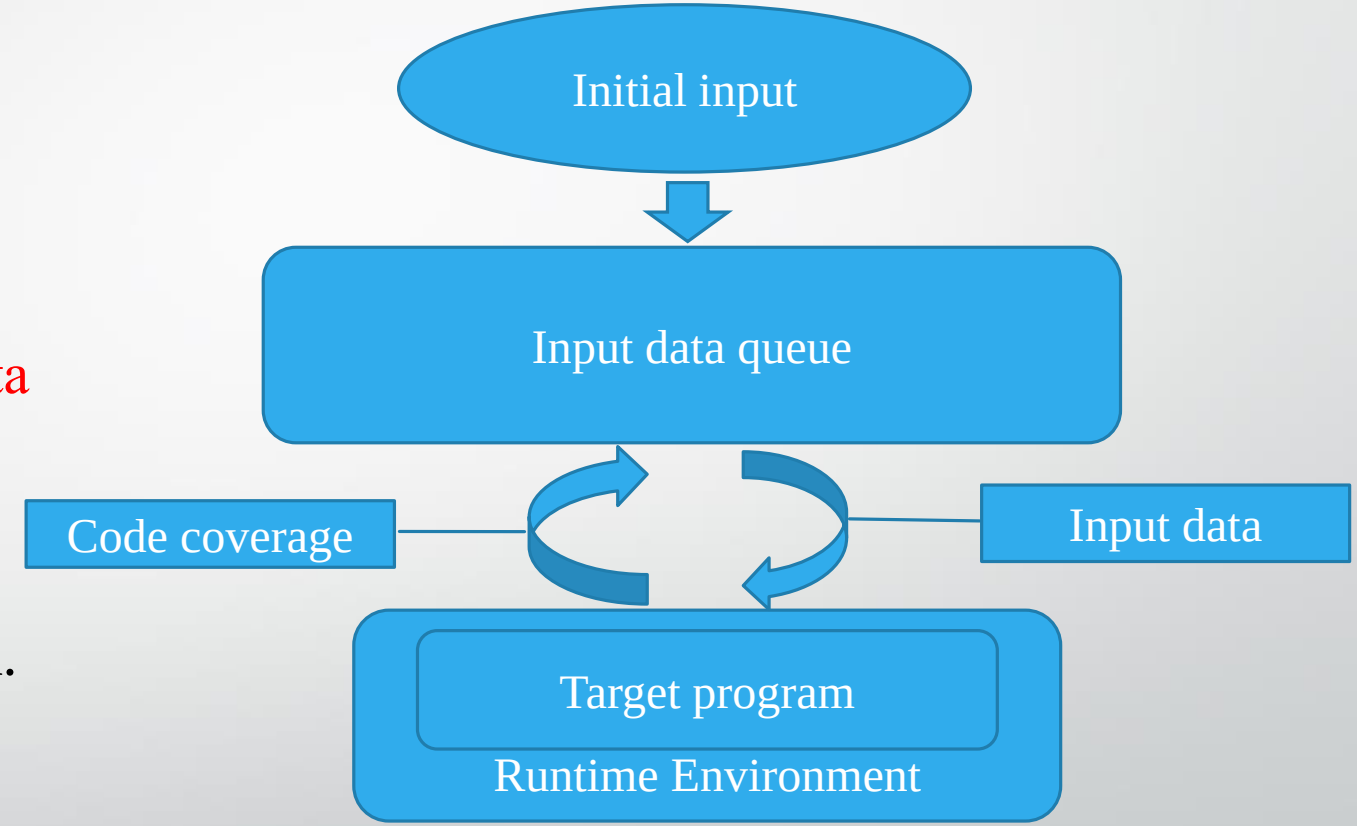
SPLab - Code Dynamic Analysis

1. BNF grammar fuzzing
2. Directed fuzzing
3. Network fuzzing
4. API calls fuzzing
5. ……..

Fuzzing

1. Dumb fuzzing
2. Smart fuzzing
3. Fuzzing can’t generate targeted data

Fuzzing tool example: AFL is smart fuzzing tool based on genetic algorithm.

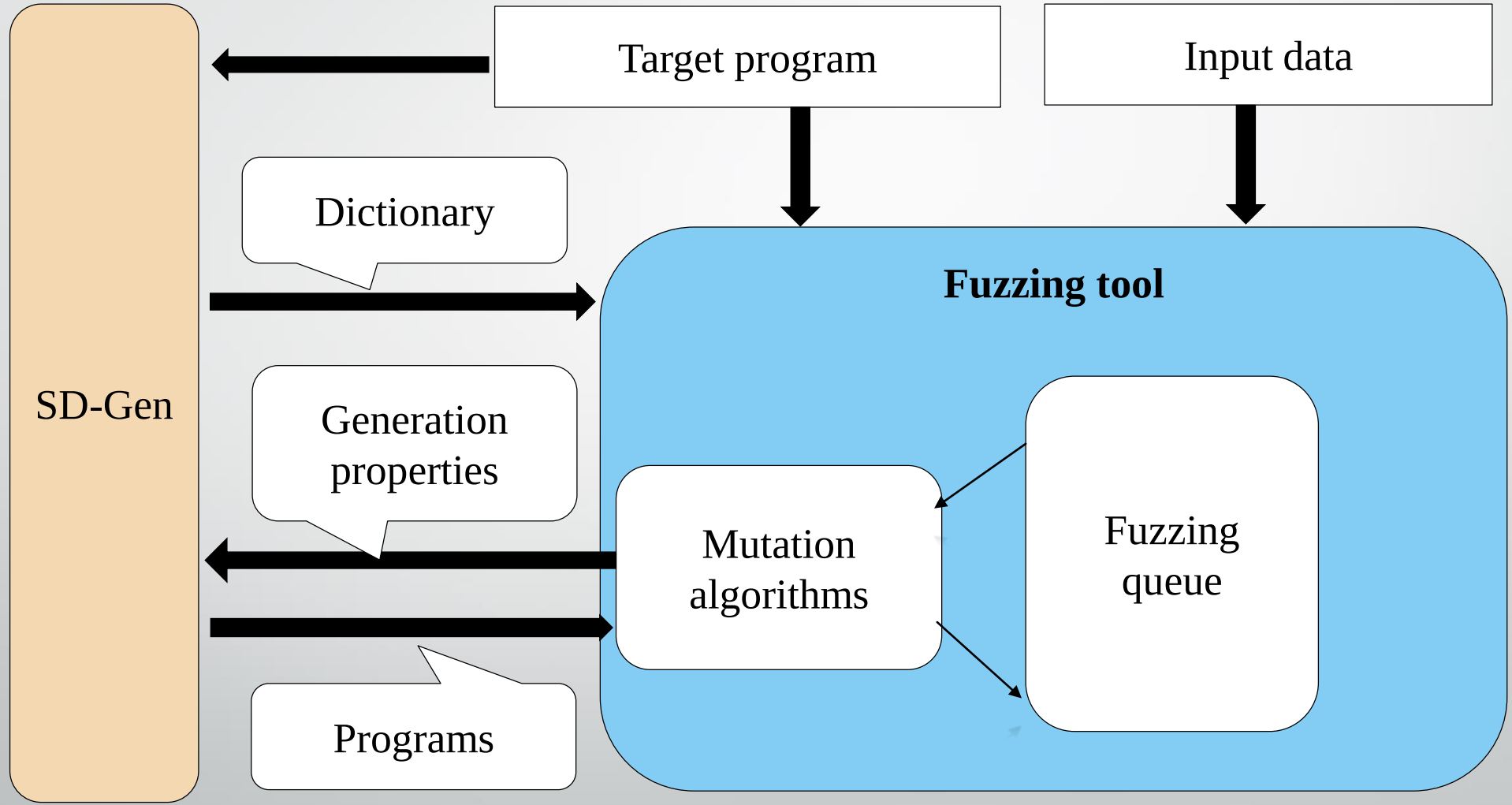


BNF Grammar Fuzzing

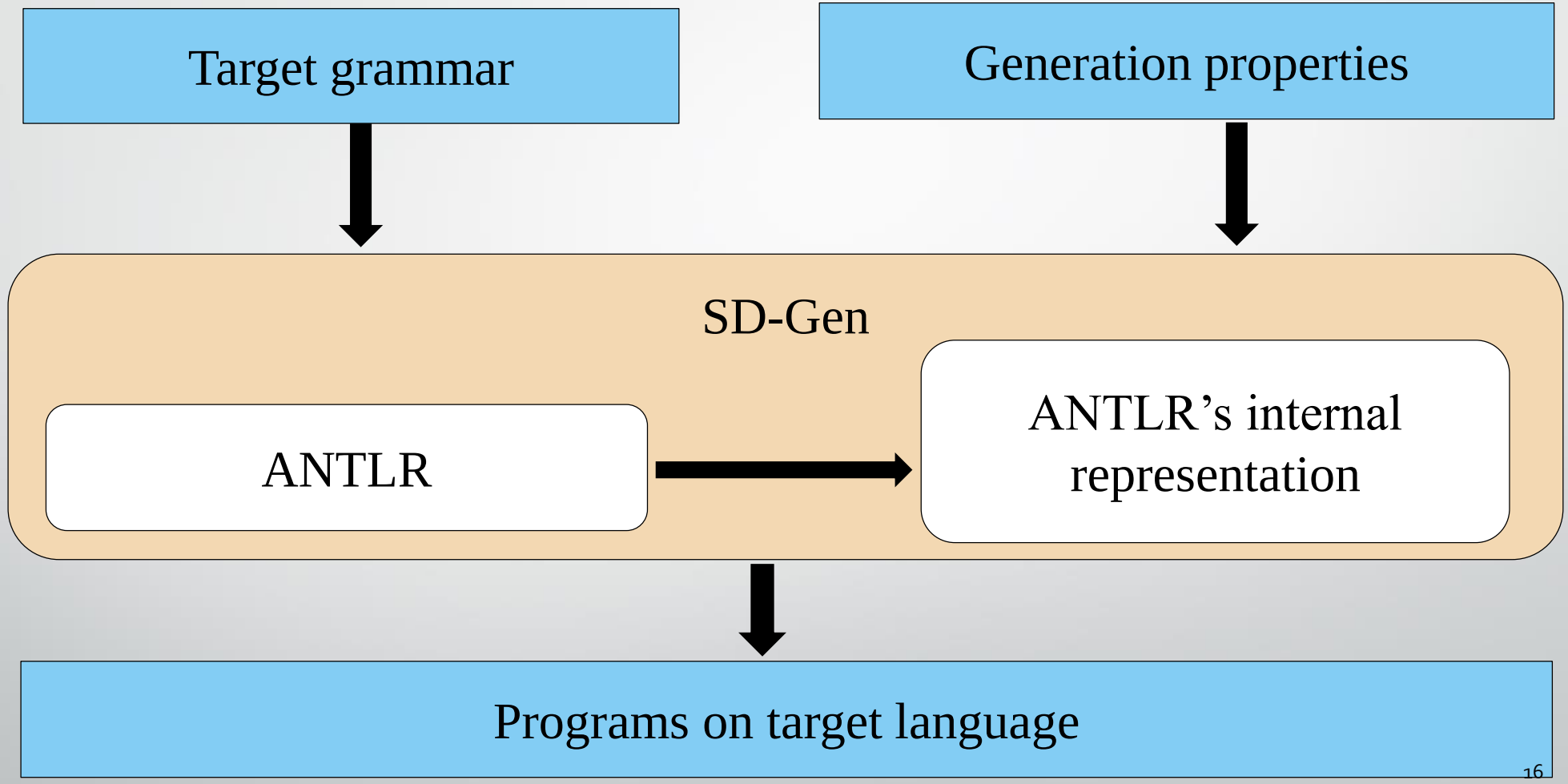
Effective fuzzing of:

1. Compilers
2. Interpreters
3. Parsers
4. Translators

BNF Grammar Fuzzing Architecture

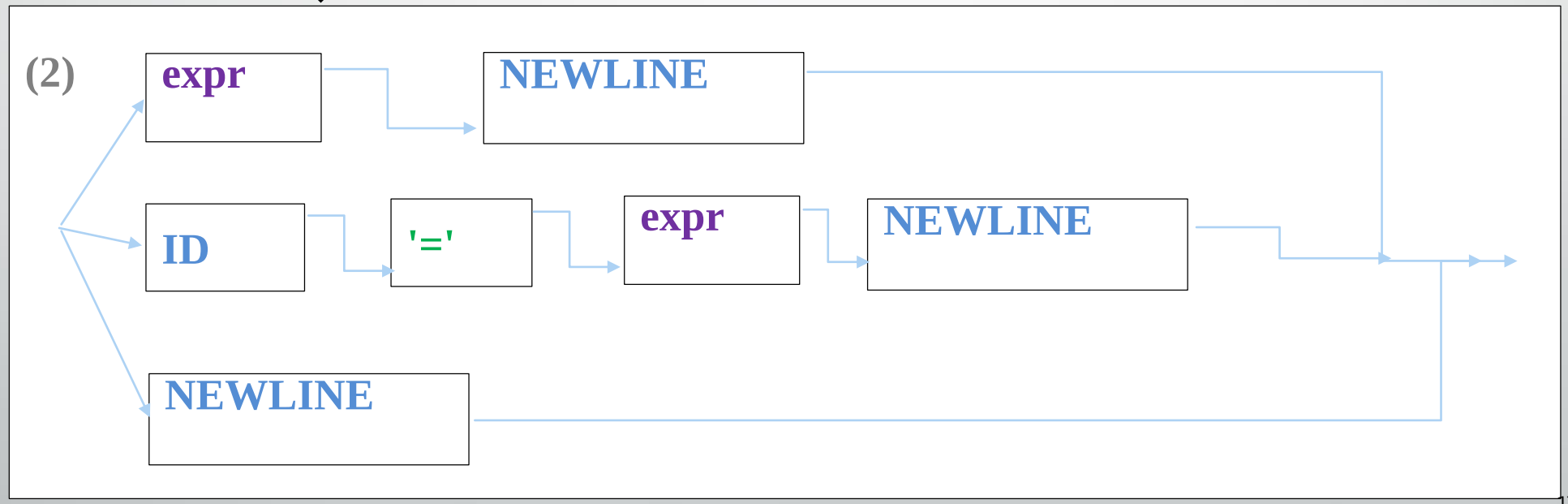


BNF Grammar Fuzzing: SD-Gen



ANTLR's Internal Representation

```
(1)  stat:   expr NEWLINE
           | ID '=' expr NEWLINE
           | NEWLINE
```



Program Generation Algorithm

1. Randomly select *BNF* rule, which is marked as start, and add it to ***rList***. Assign ***cL*** the number of nodes of the automata corresponding to the selected rule. Go to the step 2.
2. From ***rList*** choose first non-terminal node ***NT*** if exists and go to the step 3. Otherwise, go to the step 6.
3. In the automata corresponding to ***NT*** choose random path from the start node to the end. Random path is constructed based on depth first search algorithm. ***NT*** is replaced with the elements of the detected random path. ***cL*** increased by the number of nodes in the detected path. ***cD*** increased by one.
4. If ***cD*** < ***D*** and ***cL*** < ***L*** then go to the step 2. Otherwise got to step 5.
5. Each non-terminal symbol is replaced with terminal path from the start node to the end (***PFSNE***). In the corresponding automata of non-terminal symbol shortest terminal ***PFSNE*** is detected. If there is no such path, then algorithm recursively replaces non-terminal symbols of shortest ***PFSNE***.
6. Write values of elements ***rList*** to the buffer and returns it.

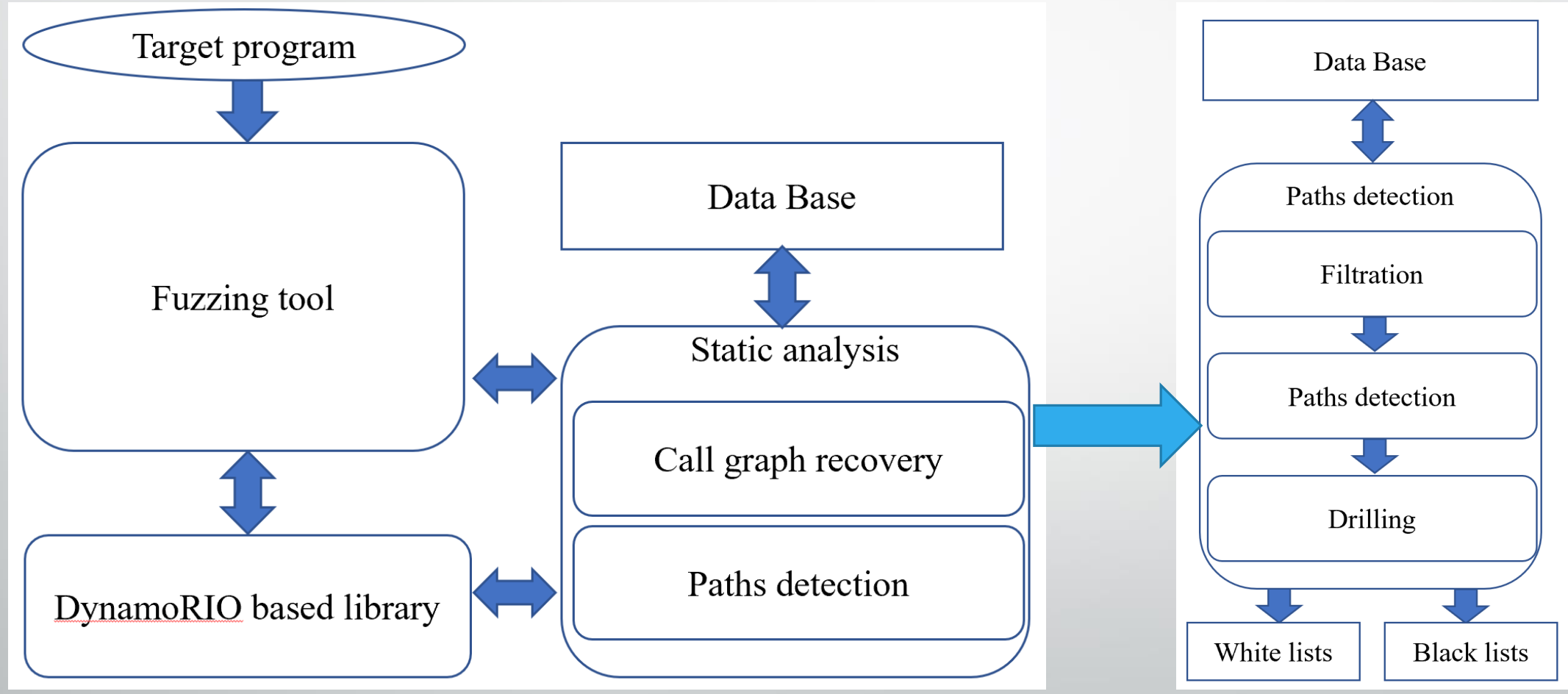
BNF Grammar Fuzzing Results

Test name	All basic blocks	AFL Fuzz	Fuzz + Dictionary	Fuzz + Dictionary + SD-Gen	Coverage
gcc-7.1	41329	6240	**6343**	**6356**	**+116**
g++-7.1	41440	6378	**6382**	**6381**	**+3**
python-2.7	123819	13315	**13399**	**16979**	**+3664**
php-v7.1.7	342988	7497	**7865**	**8003**	**+506**
luca-5.3.4	8266	2007	**2248**	**2515**	**+508**

Directed Fuzzing

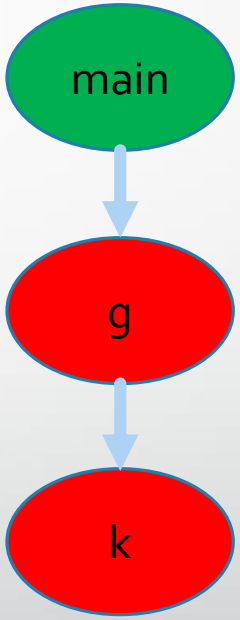
1. Suspicious fragments of code from static analysis
2. Effectively generate inputs to cover specific code fragments
3. Target program executed faster (direct fast)

Directed Fuzzing Architecture



Directed Fuzzing: Drilling

```
……
void k (…., int &a, ….) {
    a = 1;
}
void g (…., int &a, ….) {
    k(a);
}
int main () { // Entry point
    g(k);
    if (k > 0) {
      //Target point
    }
}
……
// Call graph: main()-->g()-->k()
```



Directed Fuzzing Results

Test name	Direct fuzz crashes or hangs	Direct fast fuzz crashes or hangs	AFL crashes or hangs
Personal_Fitness_Manager	2	0	0
Humaninterface	1	0	0
H20Flowlnc	0	1	0
One_Vote	1	0	0
Middleout	1	0	0
Particle_Simulator	0	1	0
Single-Sign-On	2	0	0
Stream_vm2	0	0	2
Multipass3	0	2	0
3D_Image_Toolkit	5	0	0
ECM_TCM_Simulator	2	2	0
XStore	1	0	0
HackMan	1	1	0
SAuth	1	0	1
CGC_Board	2	1	0
Flash_File_System	13	0	43
ASL6parse	1	0	0
Network_Queuing_Simulator	0	0	27

Thank You!