An IR-based Fuzzing Approach for Finding Context-Aware Bugs in API-based Systems

Wen Xu

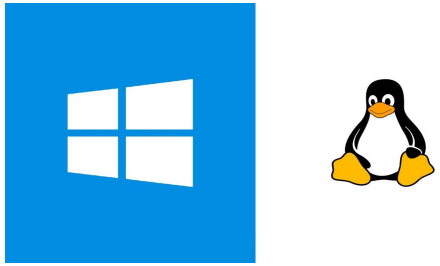
Georgia Institute of Technology

04.27.2021

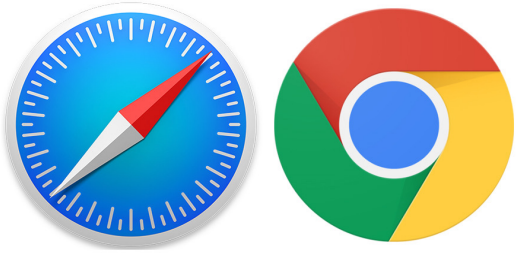
API Fuzzing Revisited

API-based Systems

- **Security-critical** software used through **programming interfaces**
- *Input:* a set of **API calls** to operate objects



Operating systems
Input **system calls**
Operate **kernel objects**



Web browsers
Input **HTML (Web APIs)**
Operate **DOM objects**

Numerous APIs

Specifications

API Fuzzing

- Testing an API-based system with *randomly generated API calls*
- Generated API calls should follow the API specifications
  - **Syntax:** the structure / representation of an API
  - **Semantics:** the (contextual) input and consequence of an API

Context-aware API Fuzzing

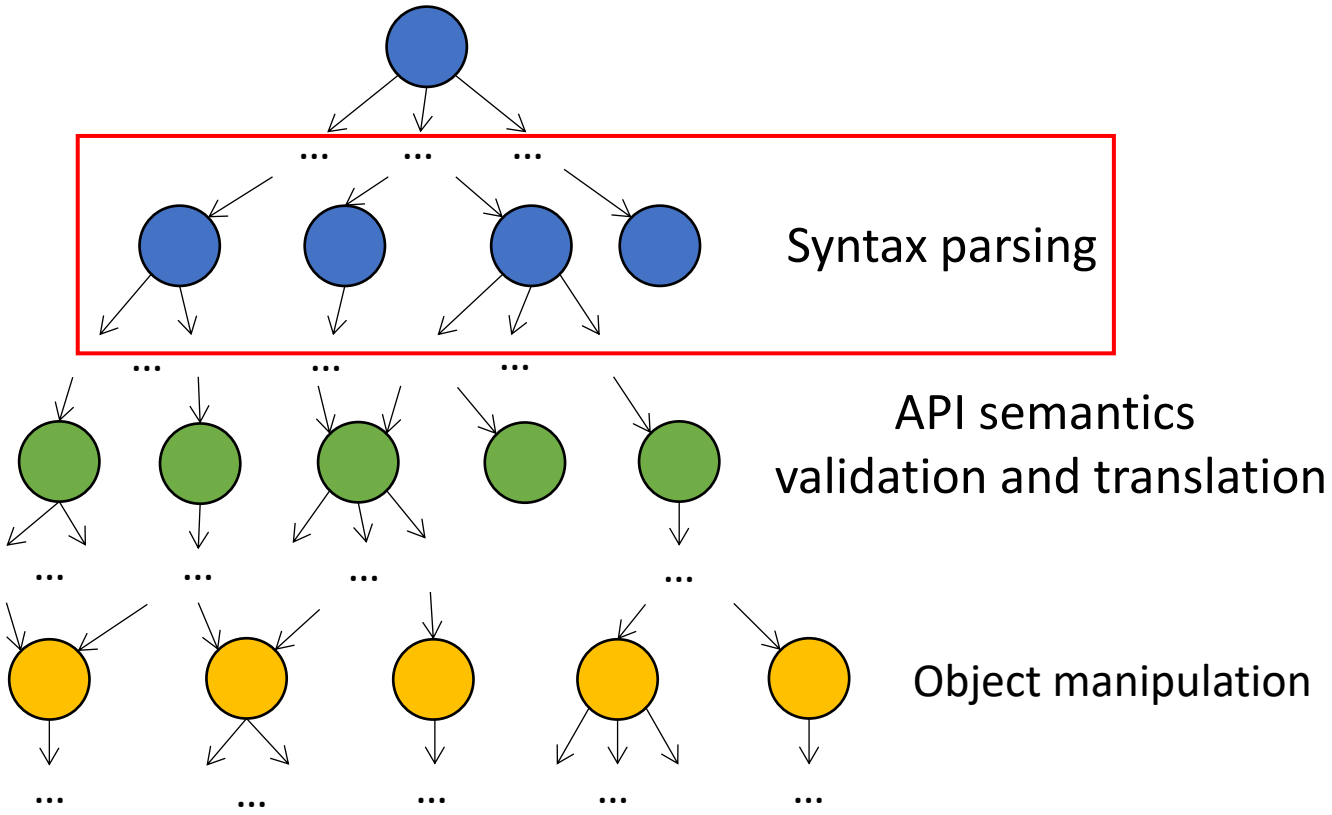


Fig. A typical API-based system

Context-aware API Fuzzing

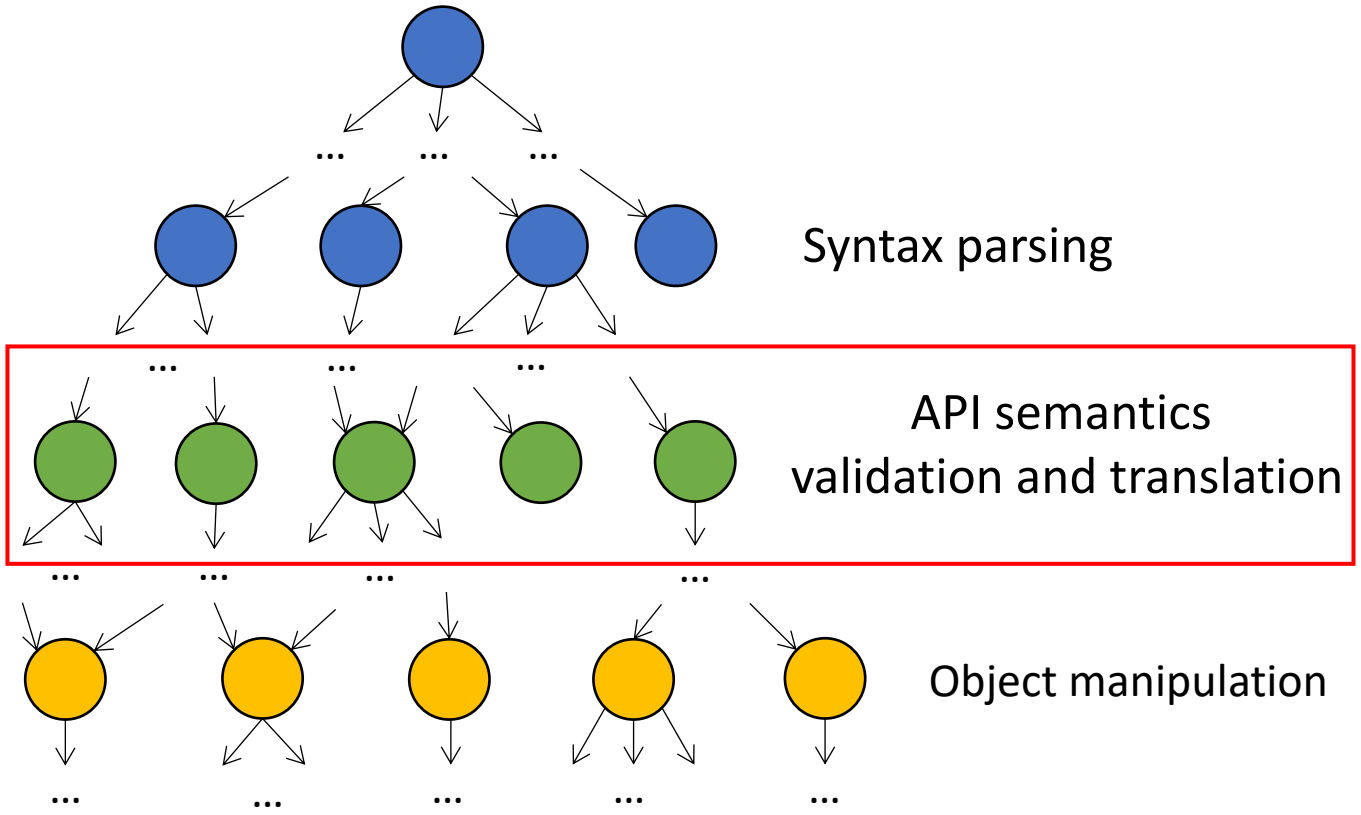


Fig. A typical API-based system

Reference (lifetime)

Write to
a **non-closed** fd

Style a **present**
HTML element

Context-aware API Fuzzing

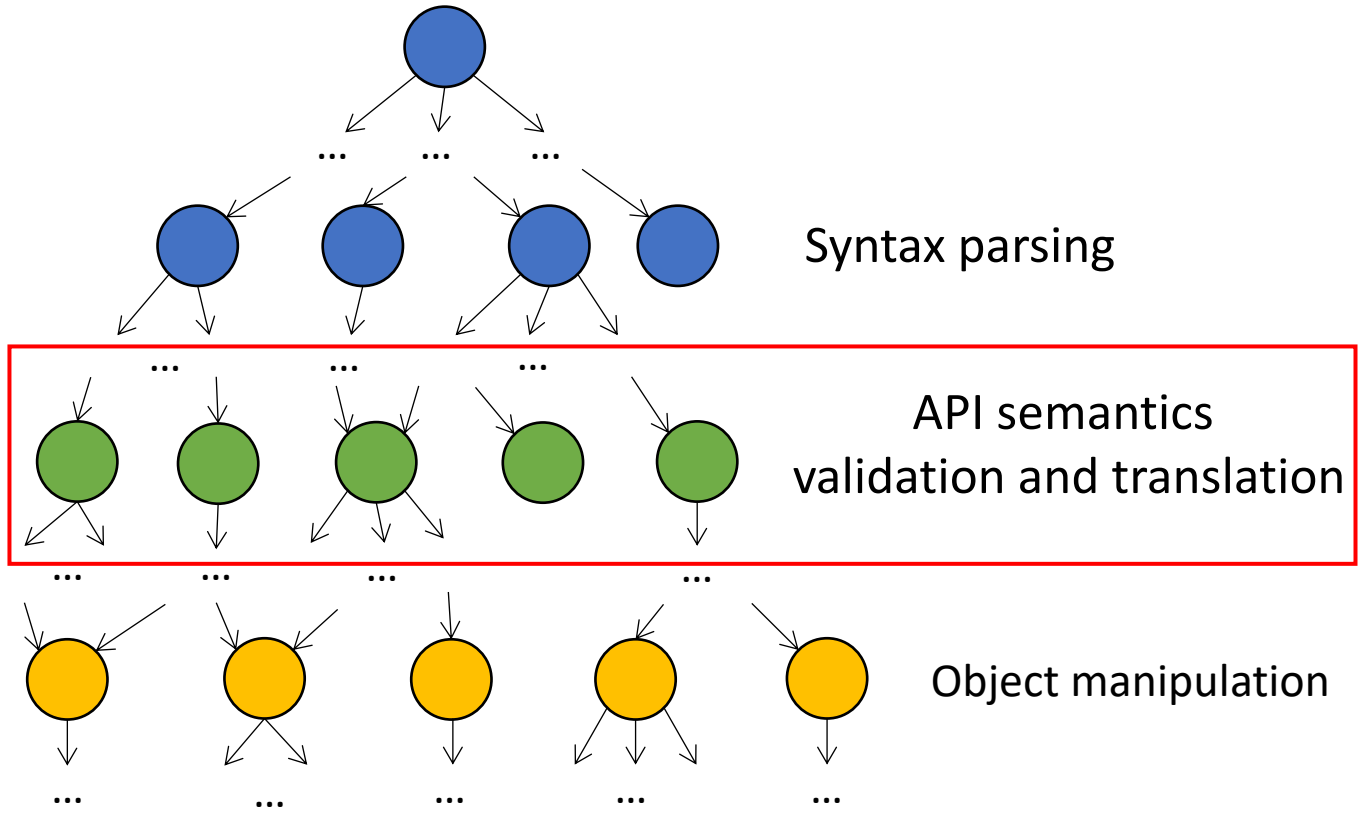


Fig. A typical API-based system

Type

CSS filtering with an SVG <filter>

Context-aware API Fuzzing

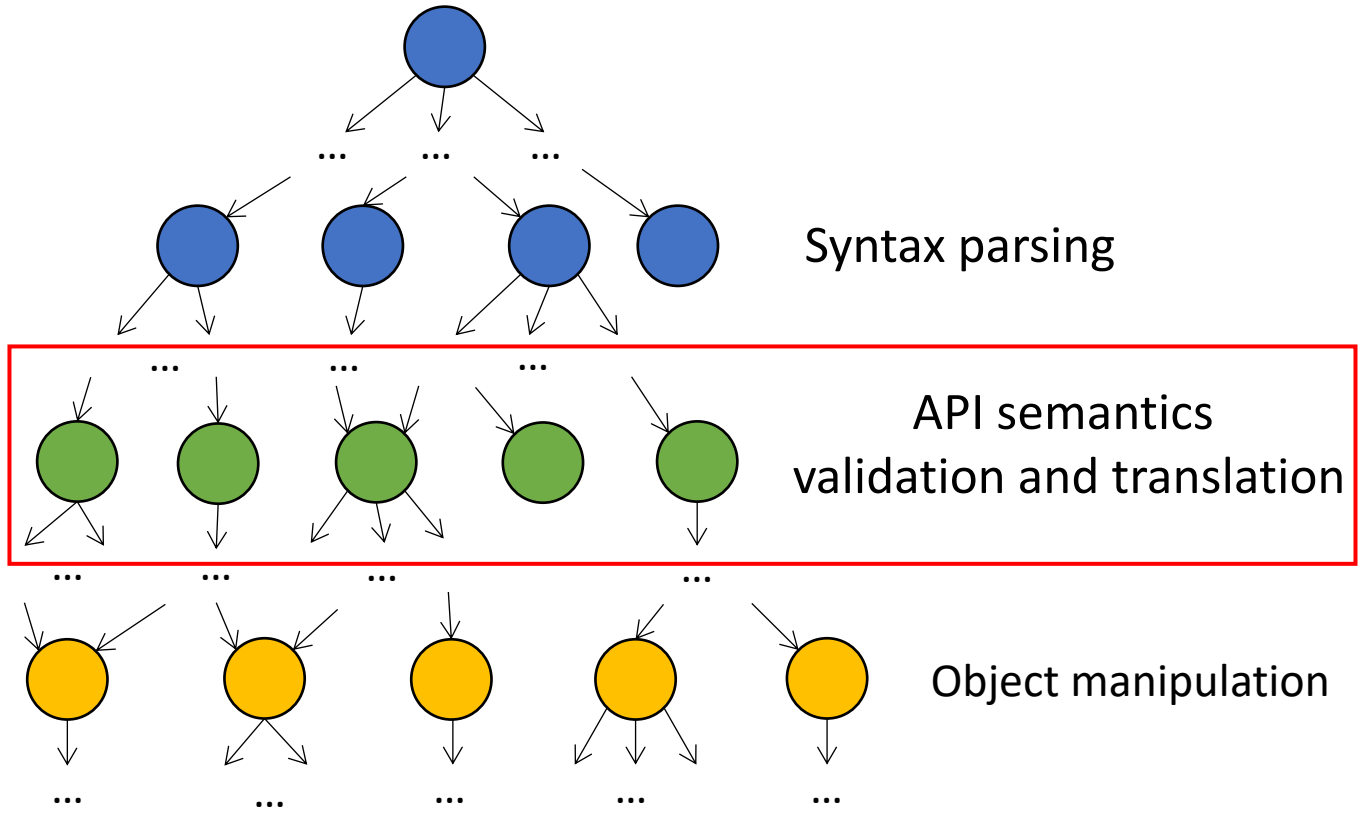


Fig. A typical API-based system

State (properties)

unlink()
an existent **path**

Use a **bounded**
WebGL buffer

Context-aware API Fuzzing

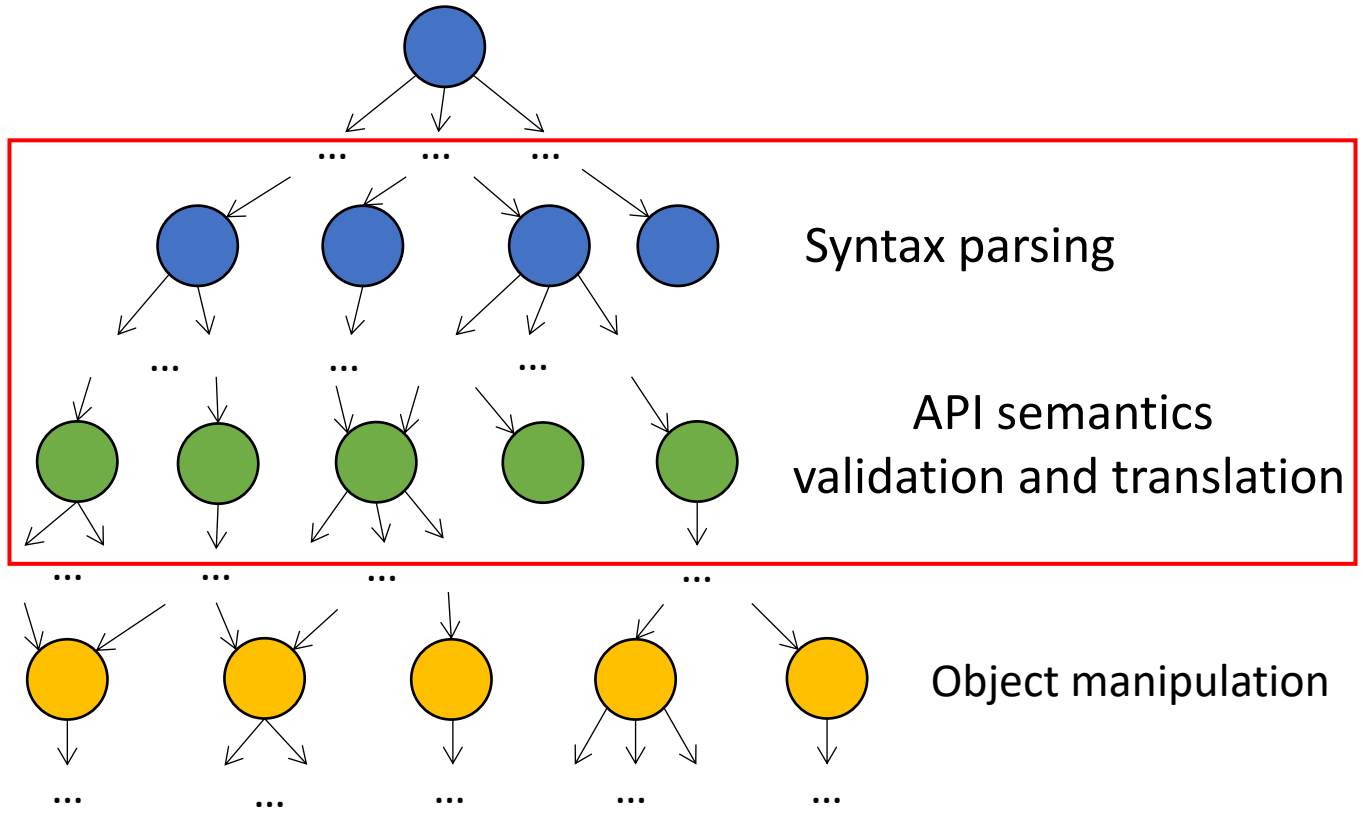


Fig. A typical API-based system

- The bugs are on the decrease
- An API call with syntax or semantic errors easily cause *an early exit*

Context-aware API Fuzzing

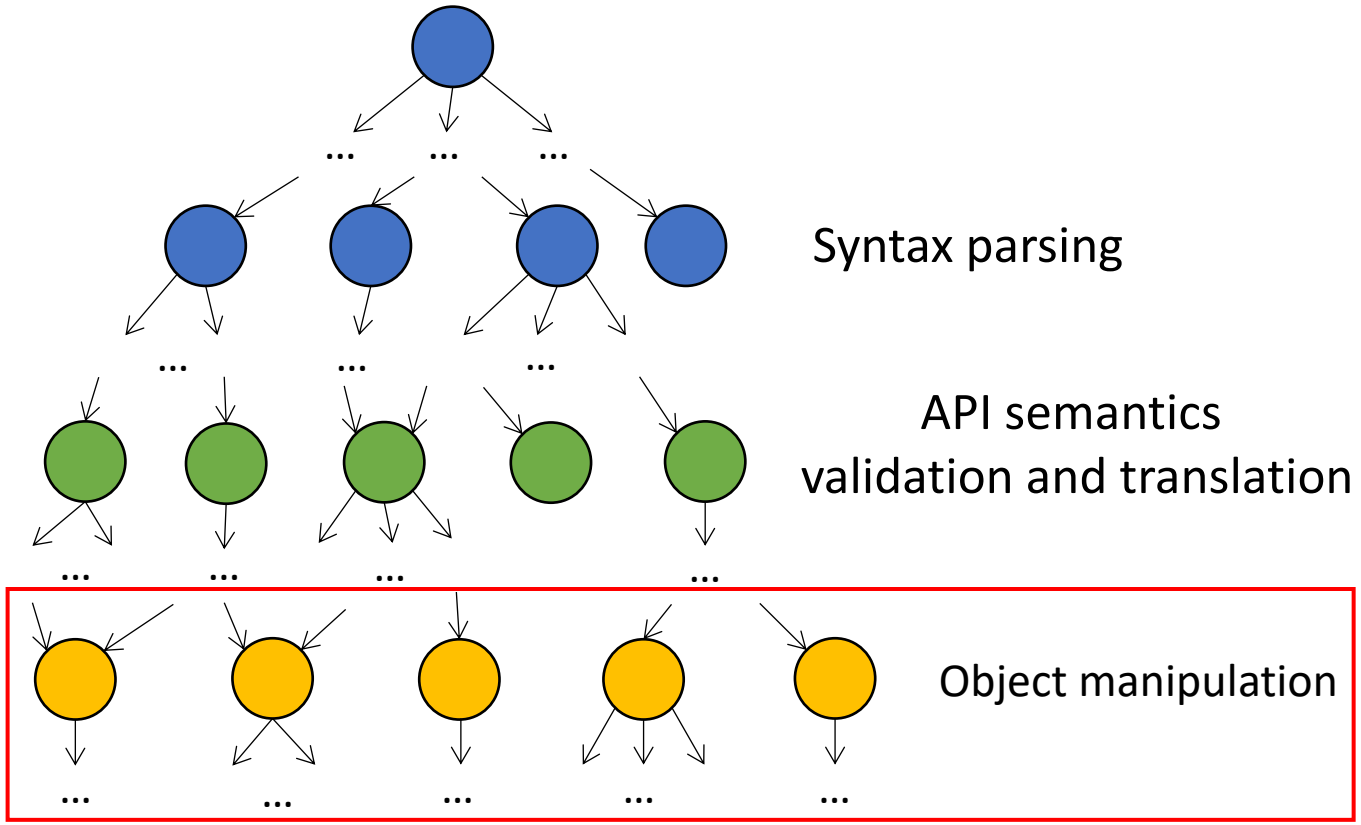
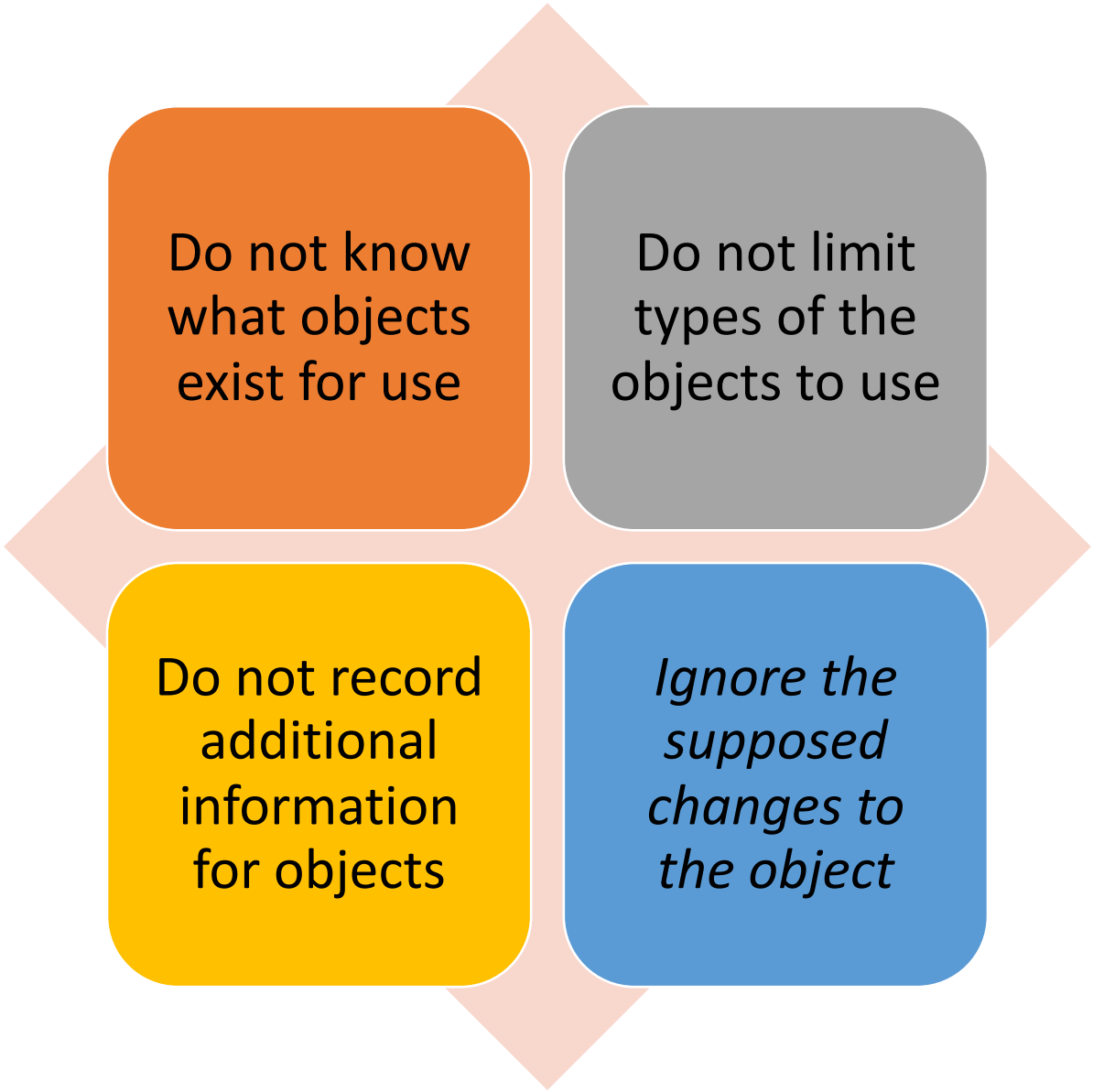


Fig. A typical API-based system

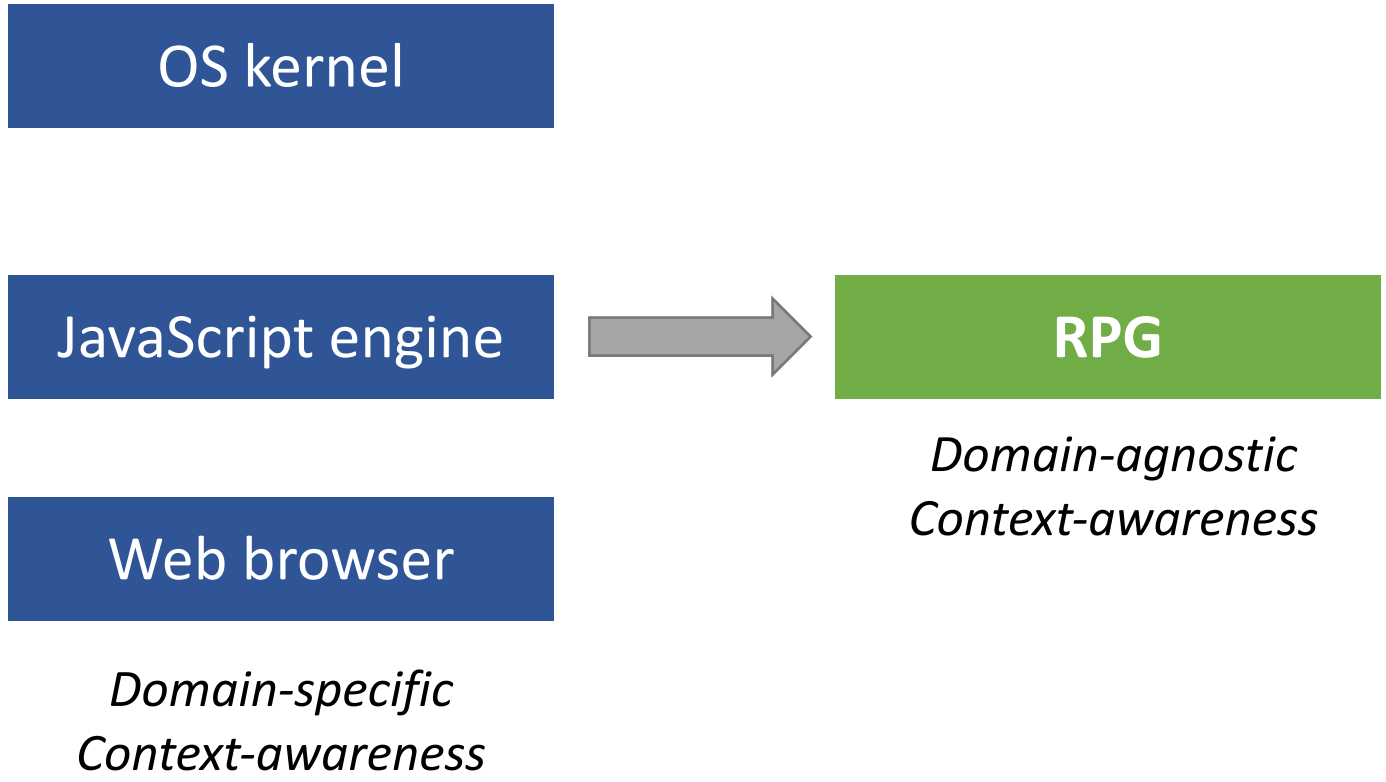
- Syntactically and semantically correct API calls arrive here
- Joint effects of API calls trigger context-aware bugs
  - Objects are the bridge

Unfortunately, existing API fuzzers fail to resolve semantic errors…



Research on *context-aware* API fuzzing

- **Janus** [SP 19]
- **Hydra** [SOSP 19]
- **DIE** [SP 20]
- **FreeDom** [CCS 20]



Research on fuzzing performance

- **OS primitives for fuzzing** [CCS 17]
- **Hardware primitives for fuzzing** [CCS 21]

Research Purpose

Grammar-based API fuzzing

- A custom context-free grammar to describe API specifications

```
1 <selector> = #<elementid>
2 <elementid> = htmlvar0000<int min=1 max=9>
3
4 <selector> = <element>
5 <element p=0.5> = <tagname>
6 <tagname> = a | abbr | acronym | address
```

Fig. Domato's grammar rules

```
1  filter_attributes :=
2          filter="+filter_value+"
3
4  filter_value :=
5          +uri+
6          none
7          inherit
8
9  uri :=
10         url(#!element_id!)
11         url(#xpointer(id('!element_id!')))
```

Fig. Dharma's grammar rules

Grammar-based API fuzzing

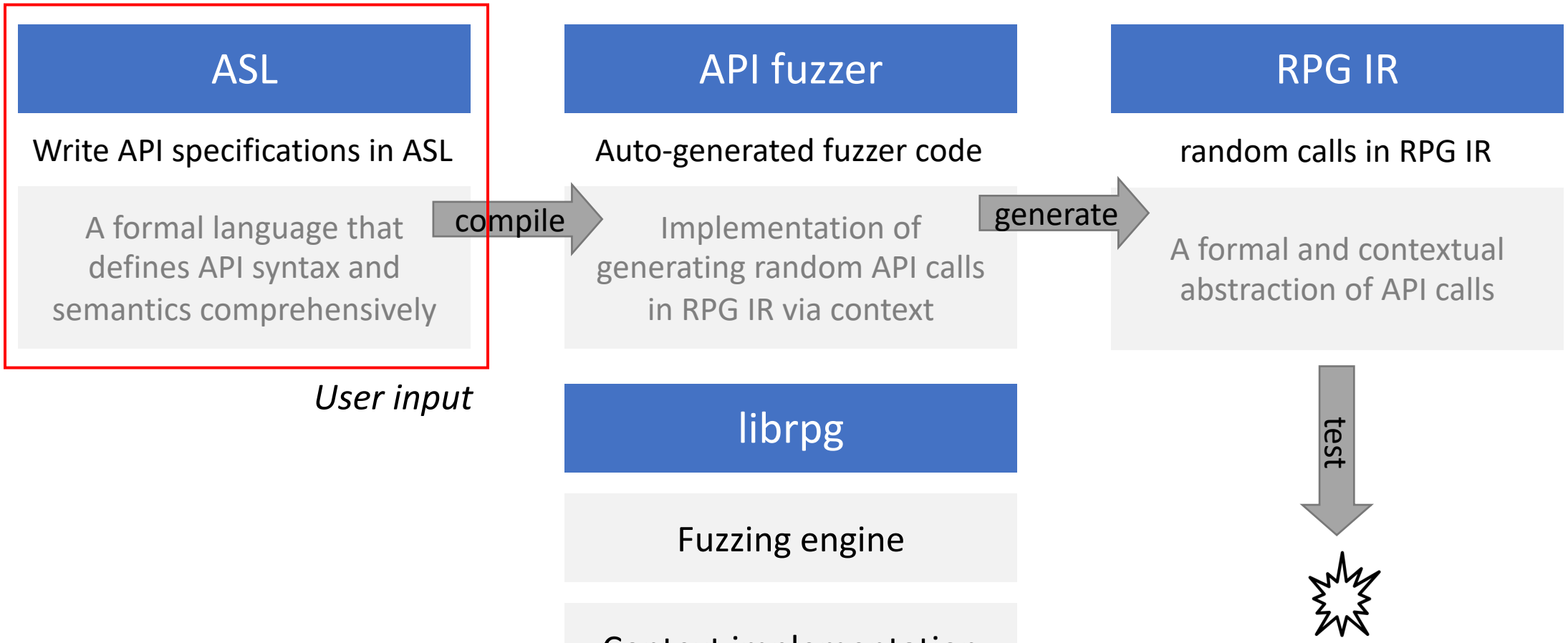
- A custom context-free grammar to describe API specifications
- **Pros**
  - Flexible to depict any API format in theory
  - Ease fuzzer development
- **Cons**
  - Contextless description addressing limited semantics only
    - At most references and types
  - A few designed for a specific class of targets (e.g., *syzlang*)

Context-aware API fuzzing

- Maintain context information during API generation
- Reflect API semantics via context dependence and updates
  - e.g., Janus, Hydra, FreeDom, etc.
- **Pros**
  - Semantic correctness
- **Cons**
  - Manual implementation of API representations / logics in specific domains

RPG: Approach

RPG – Random Program Generator



RPG – Random Program Generator

- An IR-based approach
  - RPG fuzzes API programs in **RPG IR**
  - ASL strictly follows the API model by **RPG IR**

RPG – Random Program Generator

- Frontend solution to API fuzzing
  - Interfaces for random API generation
  - Core task of every API fuzzer

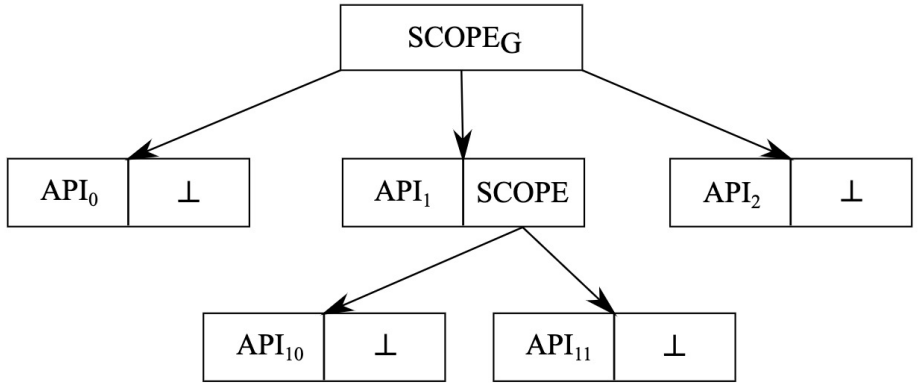
RPG: Design

RPG IR

- A formal and contextual model of an API program
  - API calls (syntax and semantics)
  - Program context

Scope model

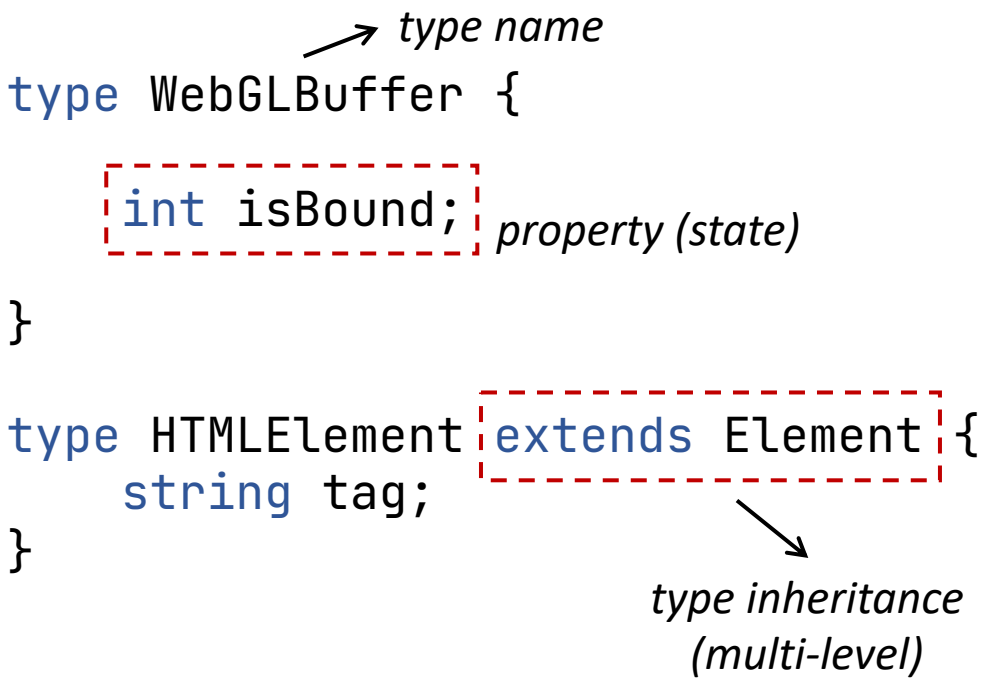
- A scope tree rooted by the global scope
- An API call can create **one** scope
- Each scope contains
  - Objects belonging to the scope – **context**
  - API calls invoked in the scope
- Each API call – program point – located by
  - (scope, index)
- Validate references



Object model

API calls operate objects

- **Typed** and **stateful** entity being operated by API calls



Type awareness

State awareness

Object model

- **Birth** location
  - The API call creates the object
- **Death** location
  - The API call invalidates the object

Reference

API model

```
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";

}
```

API model

```
      return type (void)
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";

}
```

API model

```
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";

}
```

zero or more API arguments

API model

```
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";

}
```

API effects

API model

```
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";
}
```

API string representation

Argument

- Basic building blocks of an API
  - API syntax and semantics delivered via arguments
  - API generation and mutation enabled by argument generation and mutation
- Named variables ($)

Argument

- Type
  - Primitive types: *int*, *float*, *string*, etc.
  - Object types

```
arg Document DocumentRef {
    require { Document $object; }
    is $object;
}
```

Argument

- Object imports
  - Context-dependent arguments

```
arg Document DocumentRef {
    require { Document $object; }
    is $object;
}
```

type

Argument

- Sub-arguments
  - Recursive definition to systematically construct complicated arguments
  - Fine-grained (field-sensitive) fuzzing

```
arg unquoted NthChildPseudoClass {
    args {
        BaseSelector $base;
        unquoted $nth = <CSSNth>;
    }
    is $base.":nth-child(".$nth.")";
}
```

Argument

- Value definition
  - An expression consisting of *objects*, *object properties*, sub-args, literals, etc.

```
arg Document DocumentRef {
    require { Document $object; }
    is $object;
}
```

Argument

- Value definition
  - An expression consisting of *objects*, *object properties*, sub-args, literals, etc.

```
arg unquoted NthChildPseudoClass {

    args {
        BaseSelector $base;
        unquoted $nth = <CSSNth>;
    }

    is $base.":nth-child(".$nth.")";

}
```

Argument

- Value definition
  - An expression consisting of *objects*, *object properties*, sub-args, literals, etc.

```
arg unquoted HTMLElementTagSelector {

    require {
        HTMLElement $element;
    }

    is $element["tag"];

}
```

Argument

- Object state assertions
  - Avoid state errors
- Supported conditions
  - HAS_PROPERTY
  - PROPERTY_EQ
  - PROPERTY_NEQ

```
arg FD writeFDRef {
    require { FD $fd; }
    assert { $fd["W"] == 1; }
    is $fd;
}
```

Argument

- Context-free arguments
  - Constants or random values independent from any object

```
string $tag = "input";

string $propertyName = <CSSProperty>;
```

API model

- API effects – object operations

```
api HTMLInputElement Document_createElement_input {
    ...
    effects {
        $return["tag"] = "input";
    }
    ...
```

$return: reserved variable representing returned object

- Supported effects
  - NEW_OBJECT – *implicit*
  - DEL_OBJECT
  - SET_PROPERTY

API model

- API effects – object operations

```
arg FD FDRef {
    args {
        FD $obj;
    }
    is $obj;
}
```

```
api void close {
    args {
        FDRef $fd;
    }

    effects {
        delete $fd::obj;
    }
}
```

:: operator to access variables in a (sub-)argument in ASL

API model

- API effects – object operations

```
arg string FilePathname {
    require {
        File $file;
    }
    is $file["dir"]."/".$file["filename"];
}
```

```
api FD OpenWriteOld {
    args {
        FilePathname $pathname;
        int $flags = <WriteOldFlags>;
        int $mode = <Mode>;
    }

    effects {
        $return["W"] = 1;
        $return["type"] = "FILE";
    }
}
```

API model

- Print expression
  - A string concatenation of arguments, return objects and literals

```
api HTMLInputElement Document_createElement_input {

    args {
        DocumentRef $document;
        string $tag = "input";
    }

    effects {
        $return["tag"] = "input";
    }

    print "var ".$return." = ".$document.".createElement(".$tag.");";

}
```

API model

- Certain API calls create a new scope that contains other API calls

XML (DOM tree) | Function | Loop | Branch

```
api EventHandler DefineEventHandler {
    children {
        Document_createElement,
        Document_execCommand,
        Document_getSelection,
        ...
    }

    print "function ".$return."() {";

    eprint "}";
}
```

API model

- Certain API calls create a new scope that contains other API calls

XML (DOM tree) | Function | Loop | Branch

```
api EventHandler DefineEventHandler {

    children {
        Document_createElement,
        Document_execCommand,
        Document_getSelection,
        ...
    }

    print "function ".$return."() {";

    eprint "}";
}
```

Text printed after body

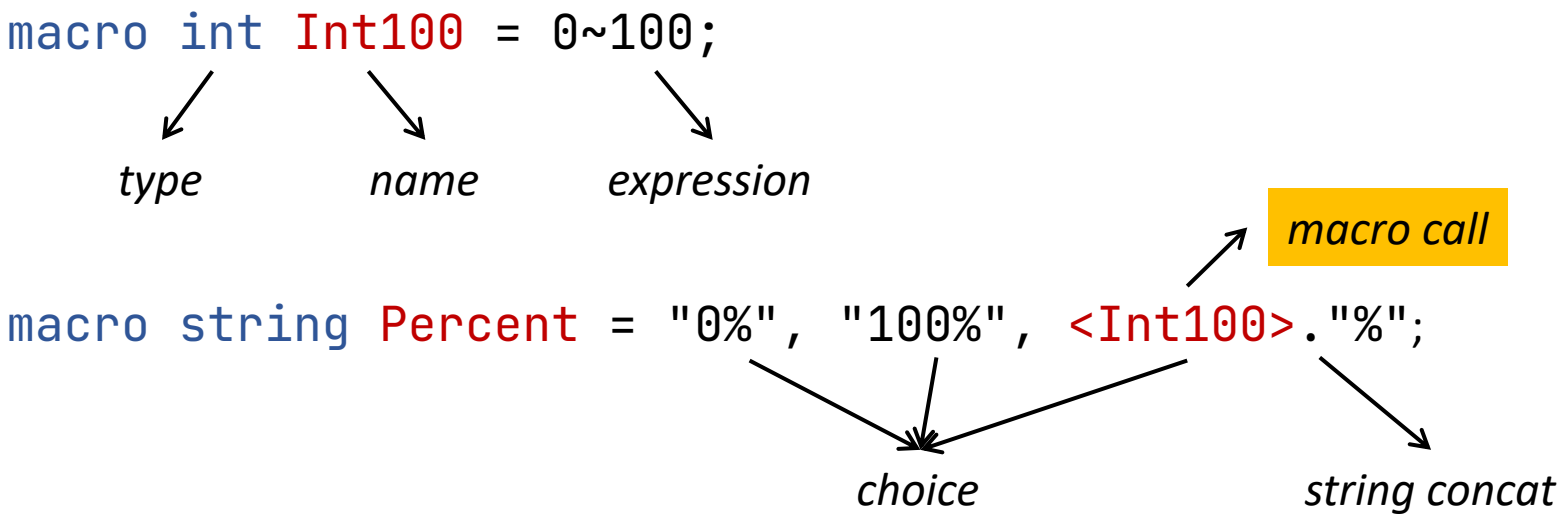
Summary

Syntax	Object-based context	Semantics

• Recursive argument definition
• Argument value expression
• API body
• API print expression
	• Scope model
	• Object model
	  • Type
	  • State
		• Object imports
		• Object state assertions
		• API effects

ASL Macros

- Context-free grammar in *ASL* that defines random values
  - Argument values independent from objects



ASL Macros

- More complicated macro expressions

```
macro string BorderWidth = <LineWidth>{1,4};
macro string Matrix = "matrix(".<Number>#{6}.")";
macro string Border = |||[<LineWidth>, <LineStyle>, <Color>];
```

Repeat

Repeat by comma

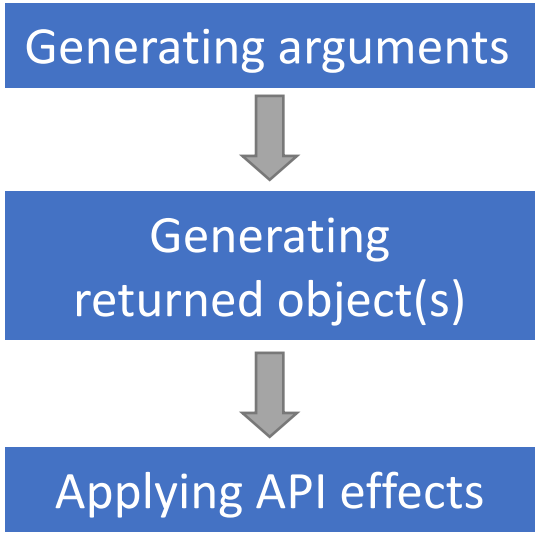
One or more items appear

ASL Macros

- API randomness (partially) comes from macros
- Guarantee what existing context-free grammar-based fuzzers generates

Context-aware API generation

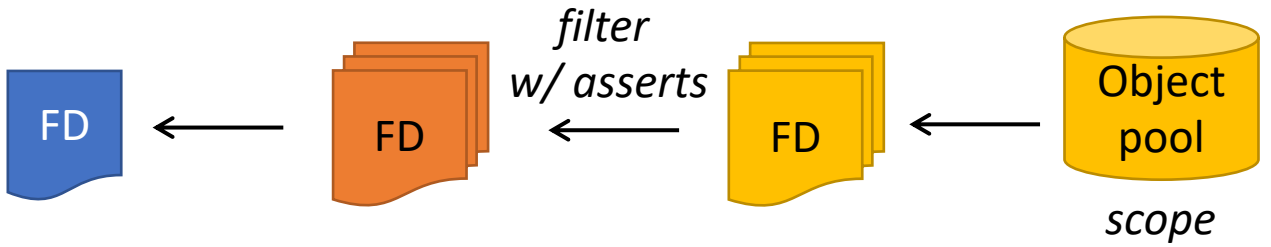
```
                ②
api HTMLInputElement Document_createElement_input {
    args {
        DocumentRef $document;
        string $tag = "input"; ①
    }
    effects {
        $return["tag"] = "input"; ③
    }
}
```



Context-aware API generation

```
arg Document DocumentRef {
    require { Document $object; }
    is $object;
}
```


```
arg FD writeFDRef {
    require { FD $fd; }
    assert { $fd["W"] == 1; }
    is $fd;
}
```

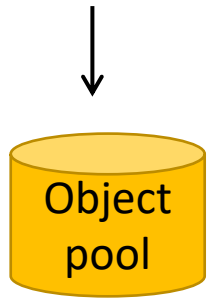


Context-aware API generation

```
api HTMLInputElement Document_createElement_input {
    args {
        DocumentRef $document;
        string $tag = "input";
    }
    effects {
        $return["tag"] = "input";
    }
}
```

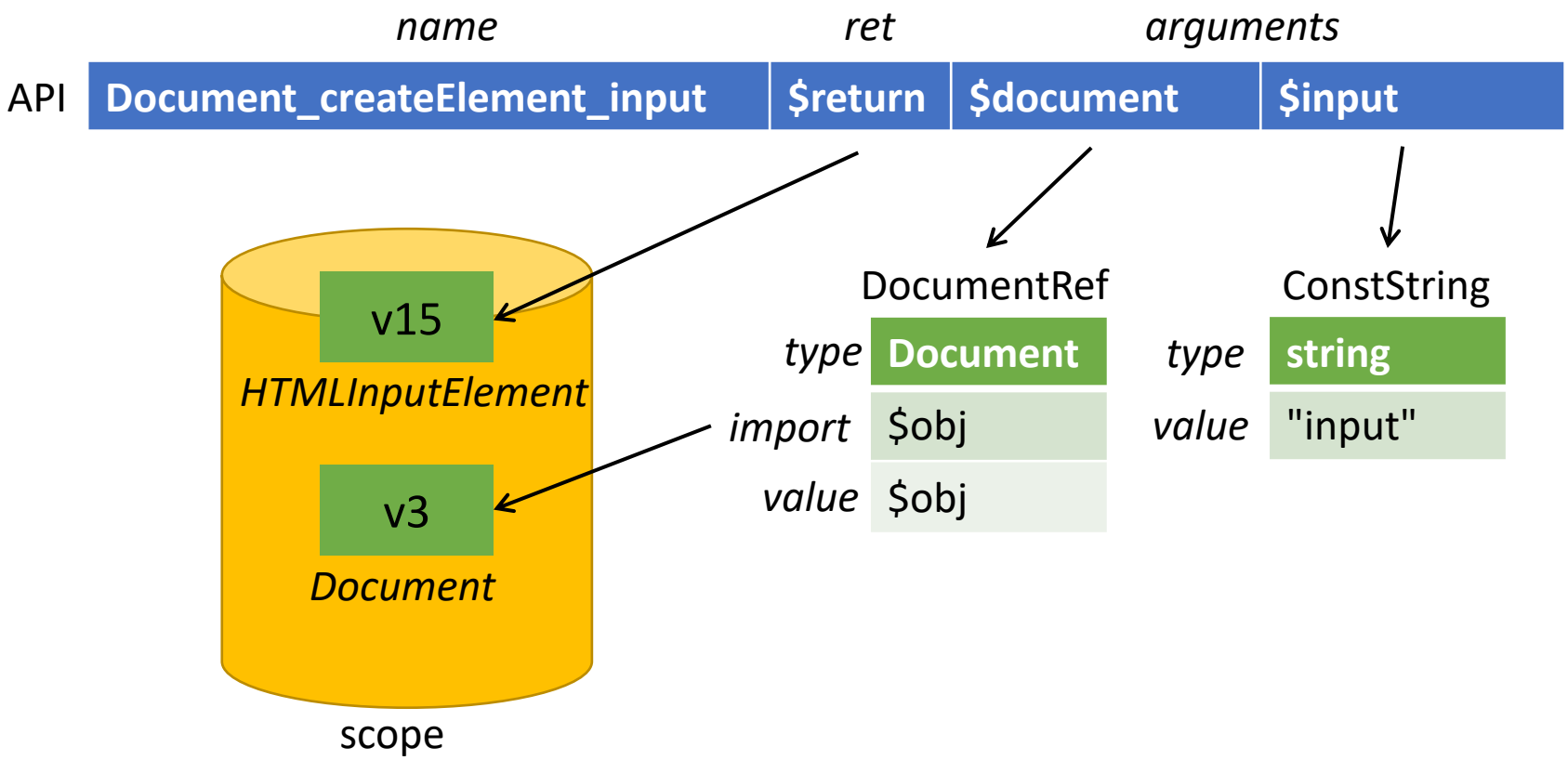
$return

Type	HTMLInputElement	
ID	v15	
Birth	[scope]:[index]	
Properties	tag	"input"



scope

Context-aware API generation




```
var v15 = v3.createElement("input");
```

API mutation

- Mutate an existing RPG IR program into a new one
  - API insertion
  - API deletion
  - (Sub-)argument mutation
- Avoid mutations that break the context

Further notes on ASL

- Predefined global objects

```
global Document document;
```

- External user-defined calls

```
%timestamp()
```

- Comment

```
//
/* */
```

Further notes on RPG IR

- An RPG IR program is serializable
  - Saved to the storage
  - Loaded for mutation
- Interpret API calls via RPG IR interfaces
  - Custom executor for testing OS kernels or hypervisors
  - Printed text as proof of concept (PoC)

RPG: Implementation

Implementation (LoC)

- ASL
  - Grammar in *ANTLR v4:* 340
  - Compiler: ~1,500
- Default ASL files
  - DOM specifications: 51,150
  - SVG specifications: 21,282

RPG: Evaluation

Comparing ASL with existing grammars

Fuzzer	Syntactic features				Semantic features			
	Format	Symbol	Operation	External	Reference	Type	State	Effect
Dharma	✓	✓	✓	✓	✓			
Avalanche	✓	✓	✓	✓	✓			
Domato	✓	✓	✓	✓	⚠			
syzlang		✓	✓	✓	✓	✓		
RPG	✓	✓	✓	✓	✓	✓	✓	✓

Format – Customizable API formats.
Symbol – An identifier representing a specific random value (i.e., macro in ASL).
Operation – Symbol operations for describing random values.
External – Extension via user-defined functions.

Comparing IR with existing API representations

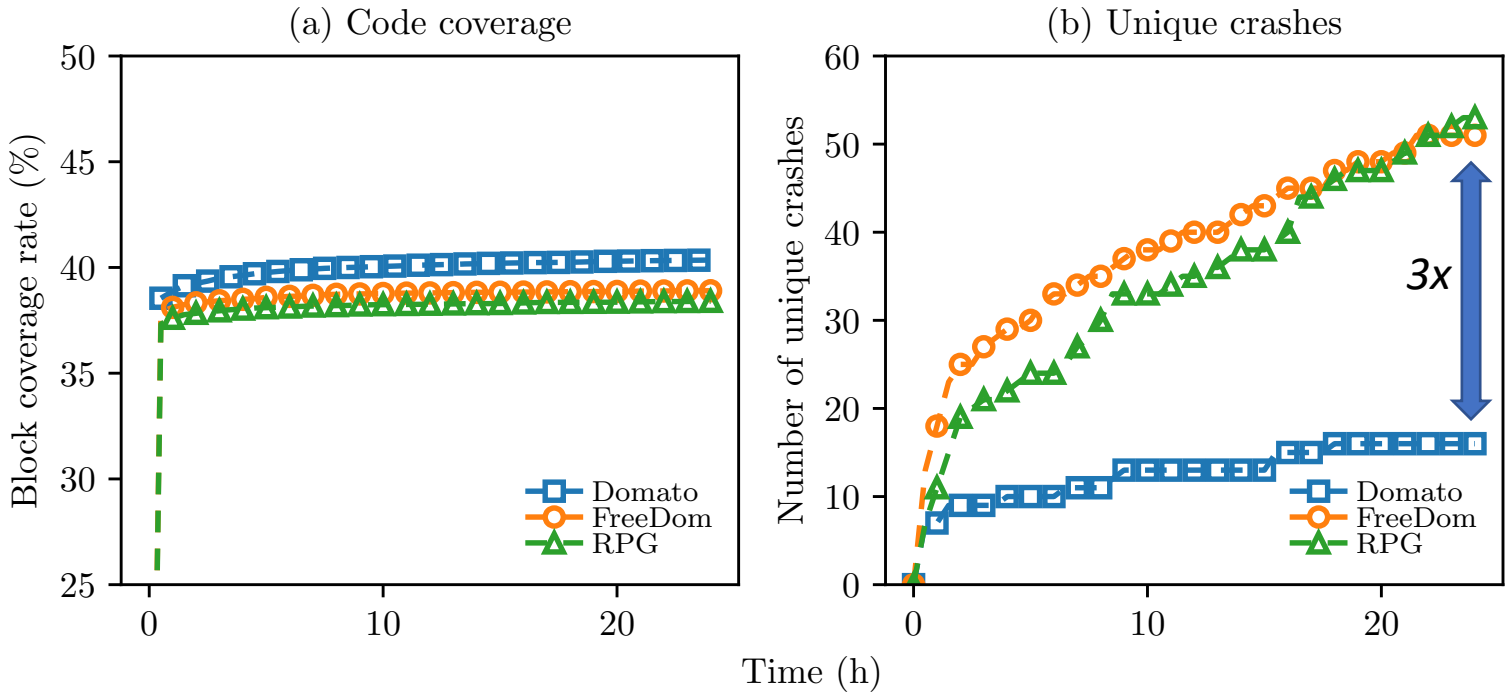
Fuzzer	Target	Reference	Type	State	Effect	Scope
FreeDom	Web browsers	✓	✓	✓	✓	⚠
Syzkaller	OS kernels	✓	✓			
Janus		✓	✓	✓	✓	
Fuzzilli	Language processors	✓	✓			✓
PolyGlot		✓	✓			✓
Nyx	Hypervisor	✓	✓		⚠	
RPG	*	✓	✓	✓	✓	✓

Experimental setup

- Hardware
  - 24 cores
  - Intel Xeon Gold 6126 CPU
  - 384 GB memory
- Setting
  - 40 fuzzer instances
  - 24 hours
- Target
  - ASan build of WebKit 2.28.0

DOM API fuzzing

- FreeDom – the state-of-the-art context-aware DOM fuzzer
- Domato – popular grammar-based DOM fuzzer

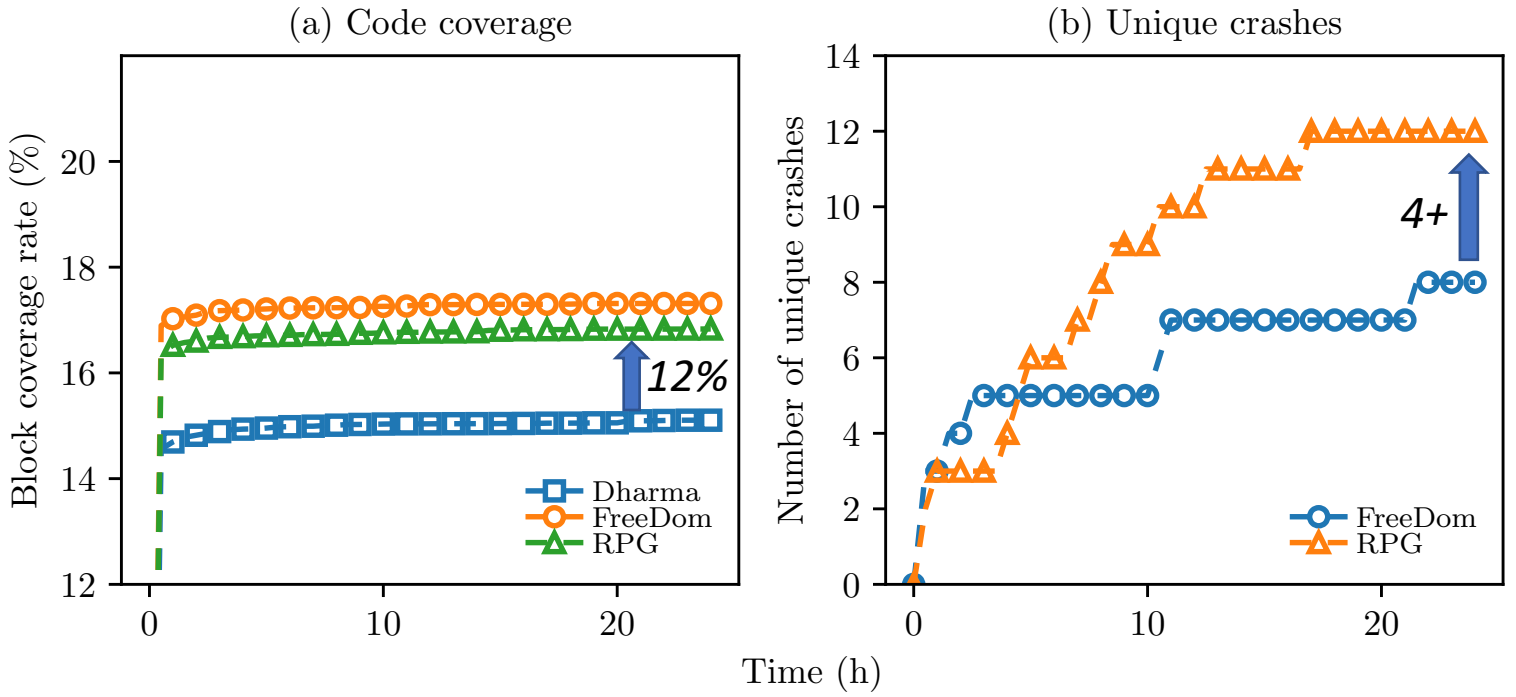


DOM API fuzzing

- Code coverage difference <2%
- 53 crashes found by **RPG**
  - Similar to **FreeDom** (51)
  - Covering 17/18 (94%) found by **Domato**
  - 53% requires a contextual input
  - 60% of the crashes missed by **Domato** (37) requires a contextual input

SVG API fuzzing

- Dharma – another context-free grammar-based fuzzer



- SVG animations
- Object-dependent functions
  - `xlink:href`
  - `marker`
  - `filter`

0 crashes found by Dharma

Summary

- As effective as the context-aware domain-specific fuzzer
- Largely outperform context-free grammar-based fuzzers

RPG: Discussion

Discussion

- Improve ASL to be a new **programming language** for making fuzzers
  - Complex conditions in object state checks with solver
  - Conditional and loop statements for object checks and API effects
- Features not supported by RPG IR
  - Bit-level values
  - Length values
  - Branches
  - Union typing
- Write more ASL files to evaluate more targets

RPG: Conclusion

Conclusion

- Existing API fuzzing methods
  - Fail to fully express API semantics being unaware of the context
  - Commonly have a domain-specific design
- First step towards domain-agnostic semantic-aware API fuzzing for finding context-aware bugs
- This thesis demonstrates this idea with RPG
  - RPG IR – A formal and context-sensitive API model
  - ASL – Language to describe API syntax and semantics

Acknowledgement

- Georgia Tech
  - Taesoo Kim
  - Wenke Lee
  - Alessandro Orso
  - Qirun Zhang
  - Meng Xu
  - Ren Ding
  - Soyeon Park
  - Seulbae Kim
  - Jungyeon Yoon
  - Fan Sang
  - Gururaj Saileshwar
- Virginia Tech
  - Changwoo Min
- Microsoft Research
  - Weidong Cui
  - Sangho Lee
- UNIST
  - Hyungon Moon
- KAIST
  - Insu Yun
- Sungshin W. University
  - Daehee Jang
- EPFL
  - Sanidhya Kashyap
- Seoul National University
  - Byoungyoung Lee
- Pennsylvania State University
  - Hong Hu

Thank you!

Wen Xu

wxu92@gatech.edu