Formal Verification of Browser Fingerprinting and Mitigation with Inlined Reference Monitors

Nathan Joslin, Phu H. Phung, Luan Viet Nguyen

University of Dayton
Department of Computer Science

What is Browser Fingerprinting?

- **Definition:** An aggregation of browser attributes
- **Stateless:** Unlike cookies, no information is saved client-side
- **Silent:** User is completely unaware

Attribute	Similarity ratio	Value
1 - User agent	0.00 %	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.40 Safari/537.36
2 - Platform	25.56 %	Linux x86_64
3 - Cookies enabled	90.52 %	✔
4 - Timezone	5.41 %	UTC-04:00
5 - Content language	41.33 %	en-US,en
6 - Canvas	0.00 %	Cwm fjordbank glyphs vext quiz, 😃 Cwm fjordbank glyphs vext quiz, 😃
7 - List of fonts (JS)	0.00 %	AR PL UMing CN, AR PL UMing HK, AR PL UMing TW, AR PL UMing TW MBE, Aakar, And 210 others
8 - Use of Adblock	63.53 %	✘
9 - Do Not Track	65.04 %	✘
10 - Navigator properties	0.79 %	44 properties detected

Source: amiunique.org

Applications and Motivations

Positive

- Ad Fraud Prevention
- Bot Detection
- Multi-Factor Auth

Duality

- Involuntary Tracking
- Voluntary MFA
- Fraud Prevention

Malicious

- Cross-site Tracking
- Malware Targeting
- Social Media Linking

Rising Popularity

Frequency of Fingerprinting on Popular Web Pages - Today

Fingerprinting the Fingerprinters
Iqbal et al. (2021)

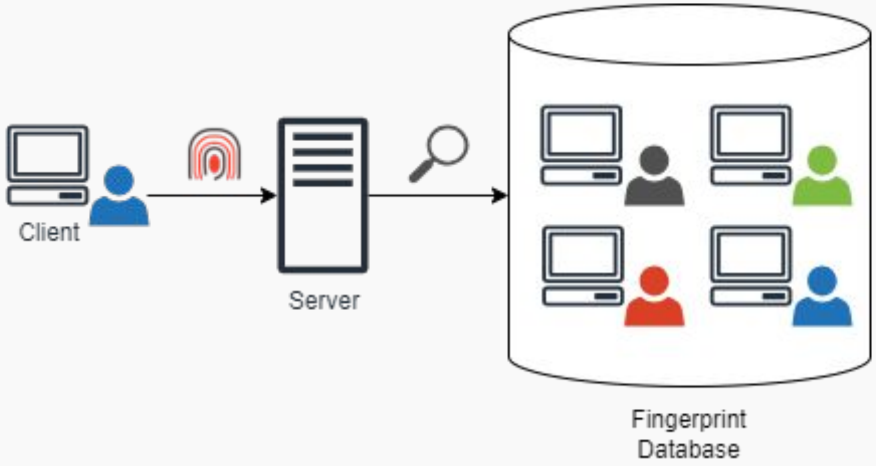
- **Estimated Usage:**
  - 30.60% of Alexa top 1K
  - 10.18% of Alexa top 100K
- **By Category:**
  - 14% of News sites
  - 6% of Shopping
- **Other:**
  - 2,349 domains serving scripts
  - 3.78% considered tracking by Disconnect

The Double Edged Sword
Senol, Ukani et al. (2024)

- **Estimated Usage:**
  - 25.75% of CrUX top 1K
  - 8.9% of CrUX top 100K
- **By Category:**
  - 9.2% of Login Pages
  - 12.5% of Sign-up Pages
- **Other:**
  - 60% of scripts use the Canvas API

Fingerprinting Mitigation

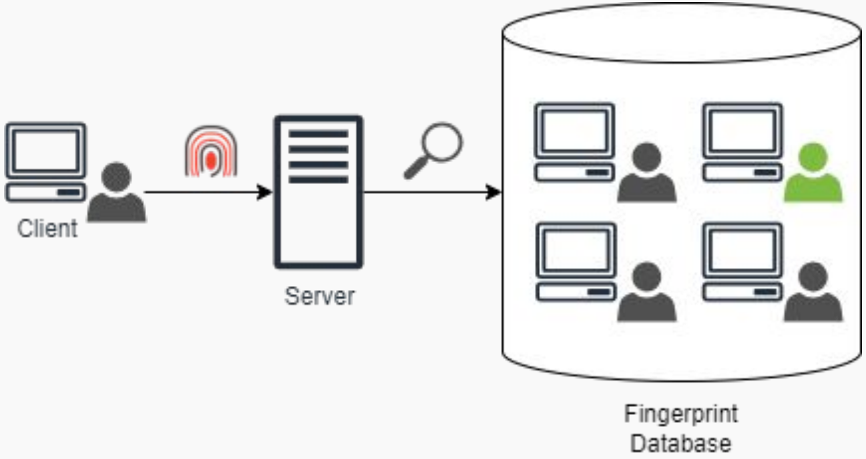
- **Policy Decision Making**
  - Machine Learning Based
  - Developer Defined Heuristics
- **Enforcement Methods**
  - API Blocking
  - Randomization
  - Normalization



Mitigation Approaches: Normalization

Normalization:

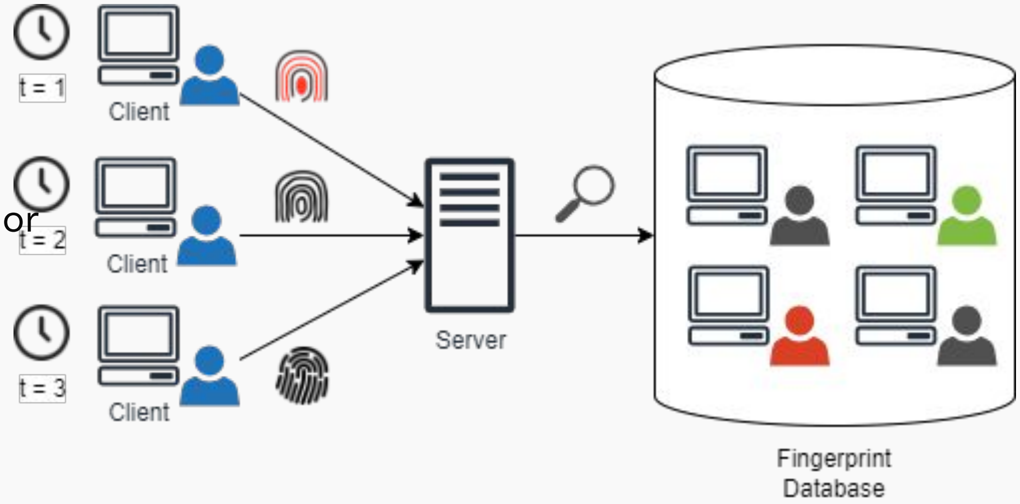
- **Goal:** “Hide in the crowd”
- <u>Reduces</u> fingerprint uniqueness by setting attributes to a shared value.
- **Usage:** Tor Browser



Mitigation Approaches: Randomization

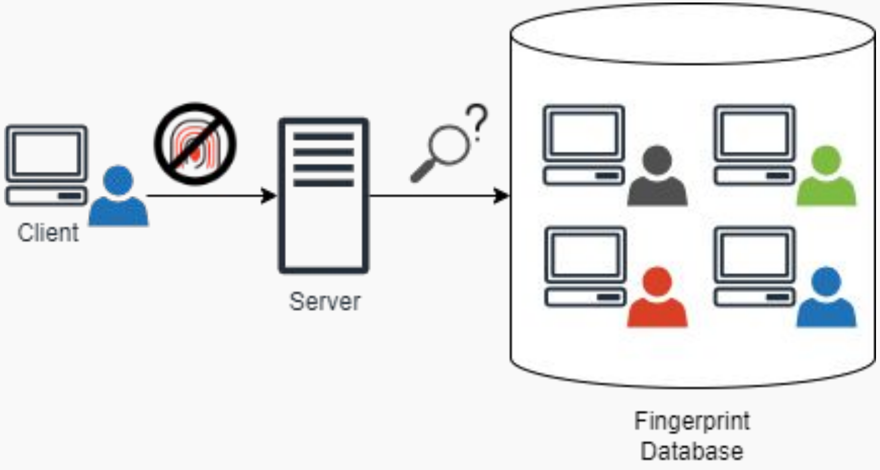
Randomization:

- **Goal:** “Moving Target”
- Increases fingerprint uniqueness by adding noise or changing attribute values
- **Usage:** Brave Browser
  - Canvas Poisoners



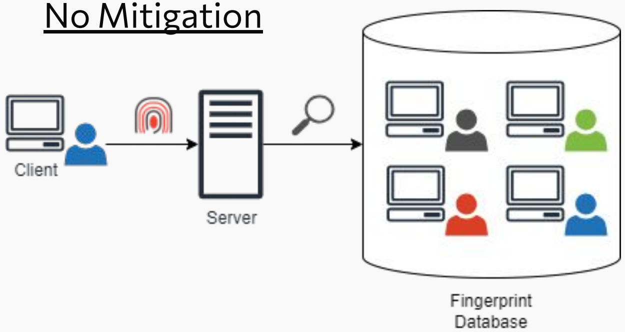
Mitigation Approaches: API Blocking

- **Goal:** Prevent function execution
- Prevents some or all attributes of a fingerprint from being collected
- **Usage:** Tor Browser, preventing canvas API

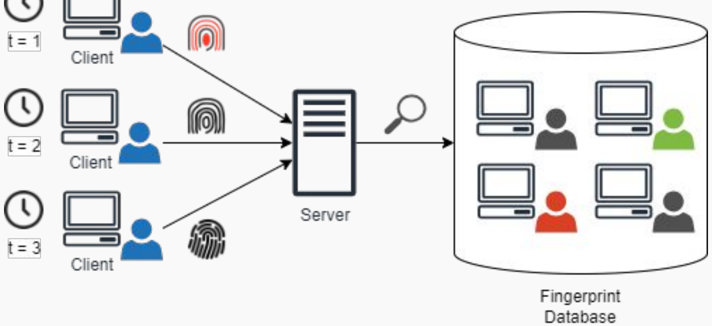


Mitigation Approaches

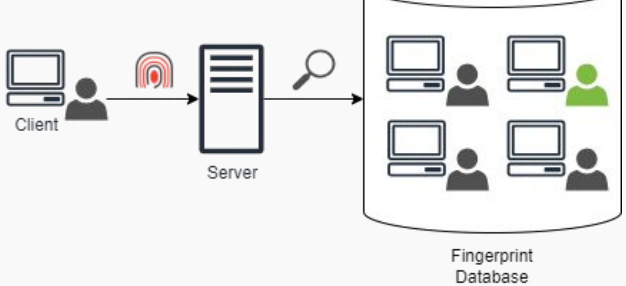
No Mitigation



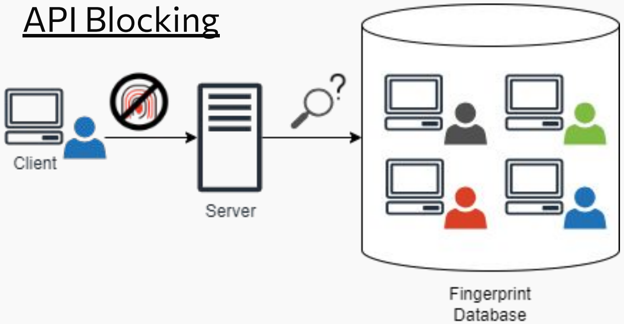
Randomization



Normalization

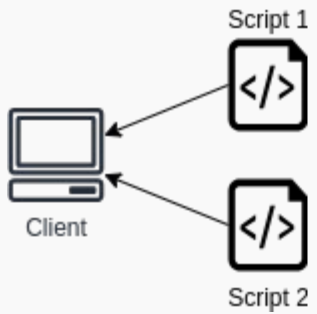


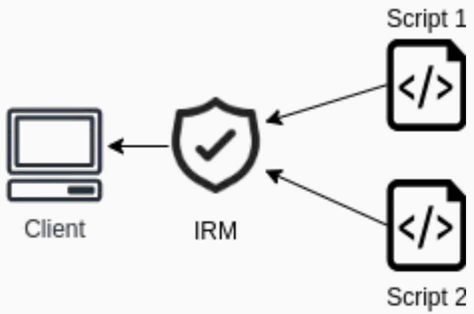
API Blocking



Inlined Reference Monitors

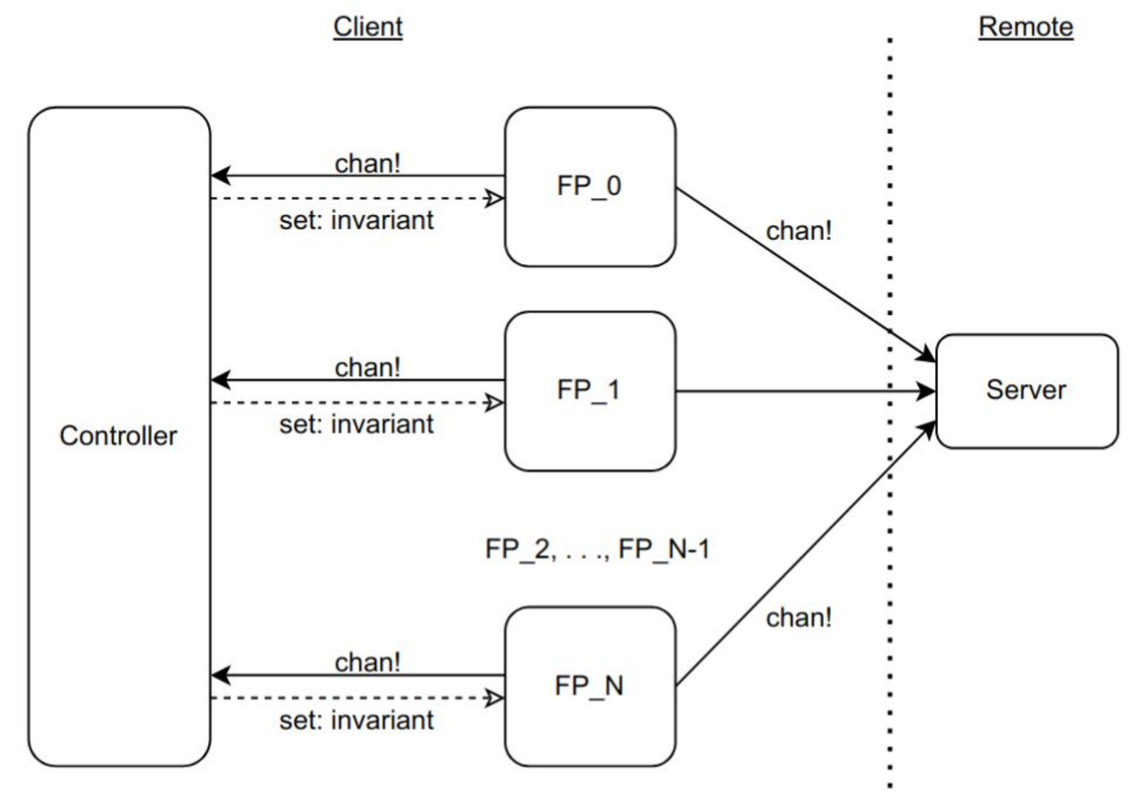
- Language-based security approach
- Rewrite/Weave security policies into the application
- Runtime interception of function calls or property accesses





Building the Components

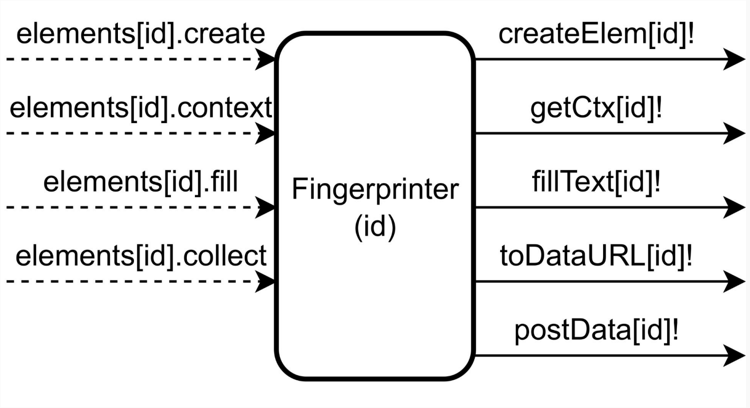
System Overview



Fingerprinter: Overview

Description:

- Models a canvas fingerprinting script
- Based off of open-source libraries and related research
- Attempts to make function calls that are intercepted by the Controller



x : Funcs Monitored

y : Fingerprinter Components

- **Input**: Invariants set by the controller.

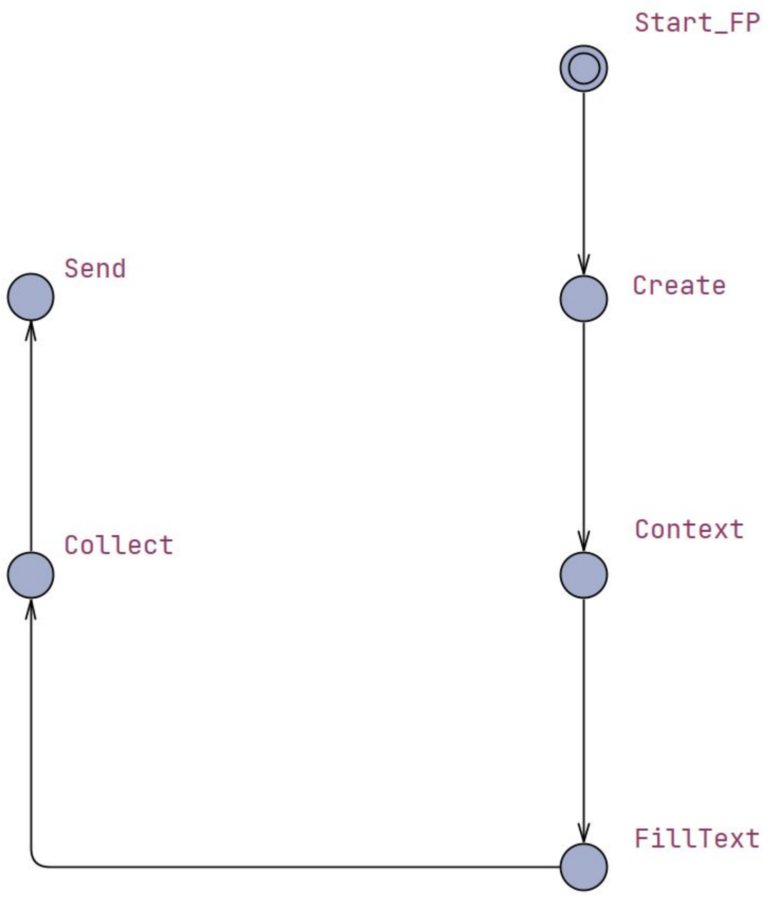
$$f(x,y) = xy$$

- **Output**: Send channel synchronizations.

$$f(x,y) = xy + y$$

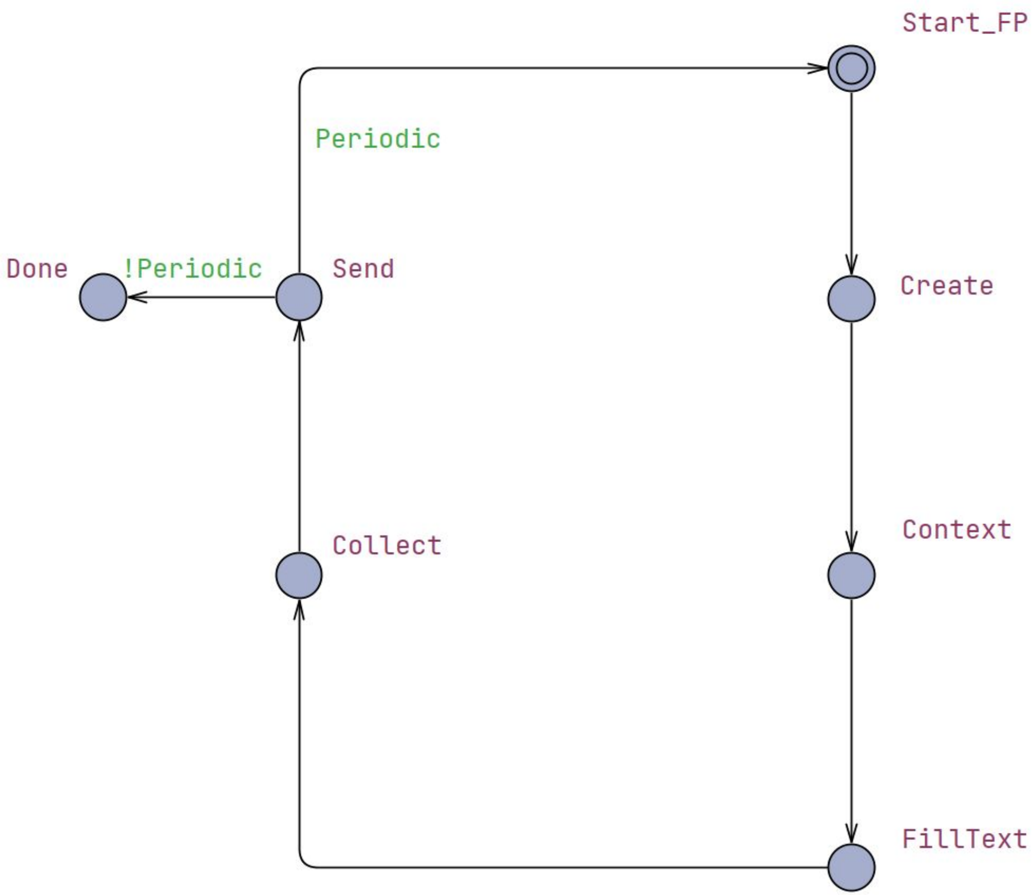
Main States

- Create canvas element
- Get canvas context
- Draw on context
- Collect value
- Send to server



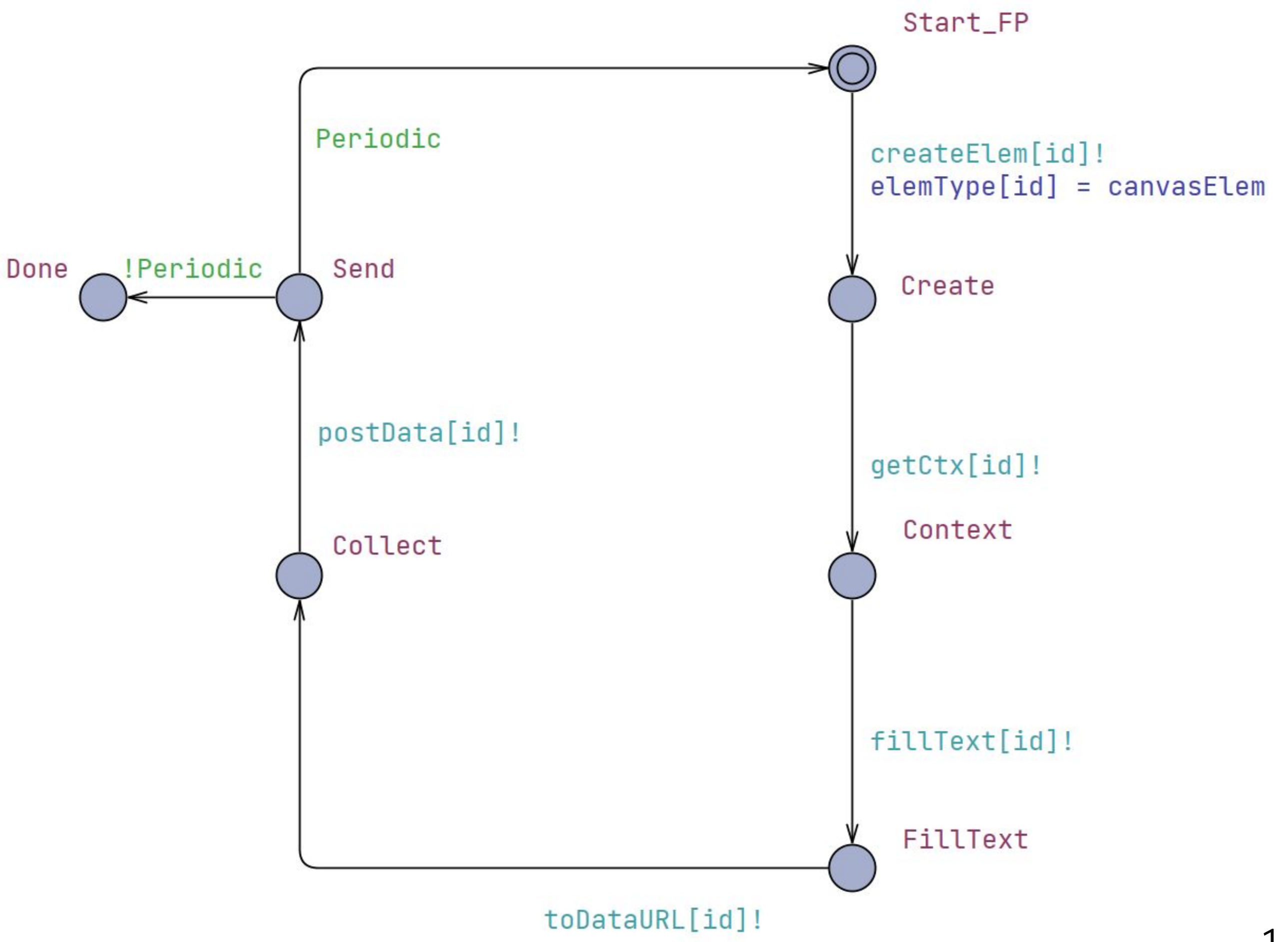
Periodicity

- Supports modeling one-and-done and repetitive scripts
- Helpful for analyzing behavior across runs
- Easily modified to an integer value



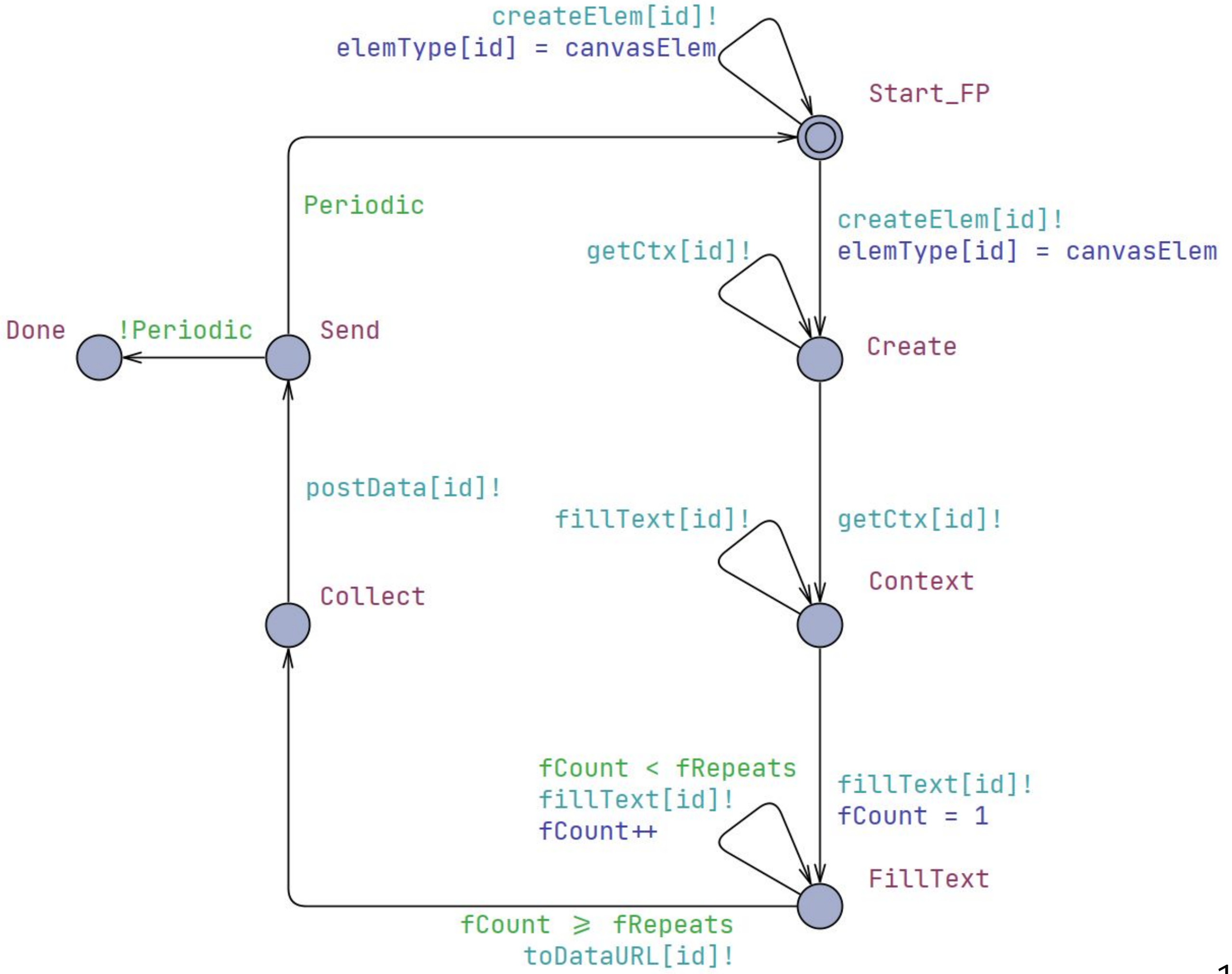
Synchronization Channels

- Instrumentation for Controller
- Models IRM function interception



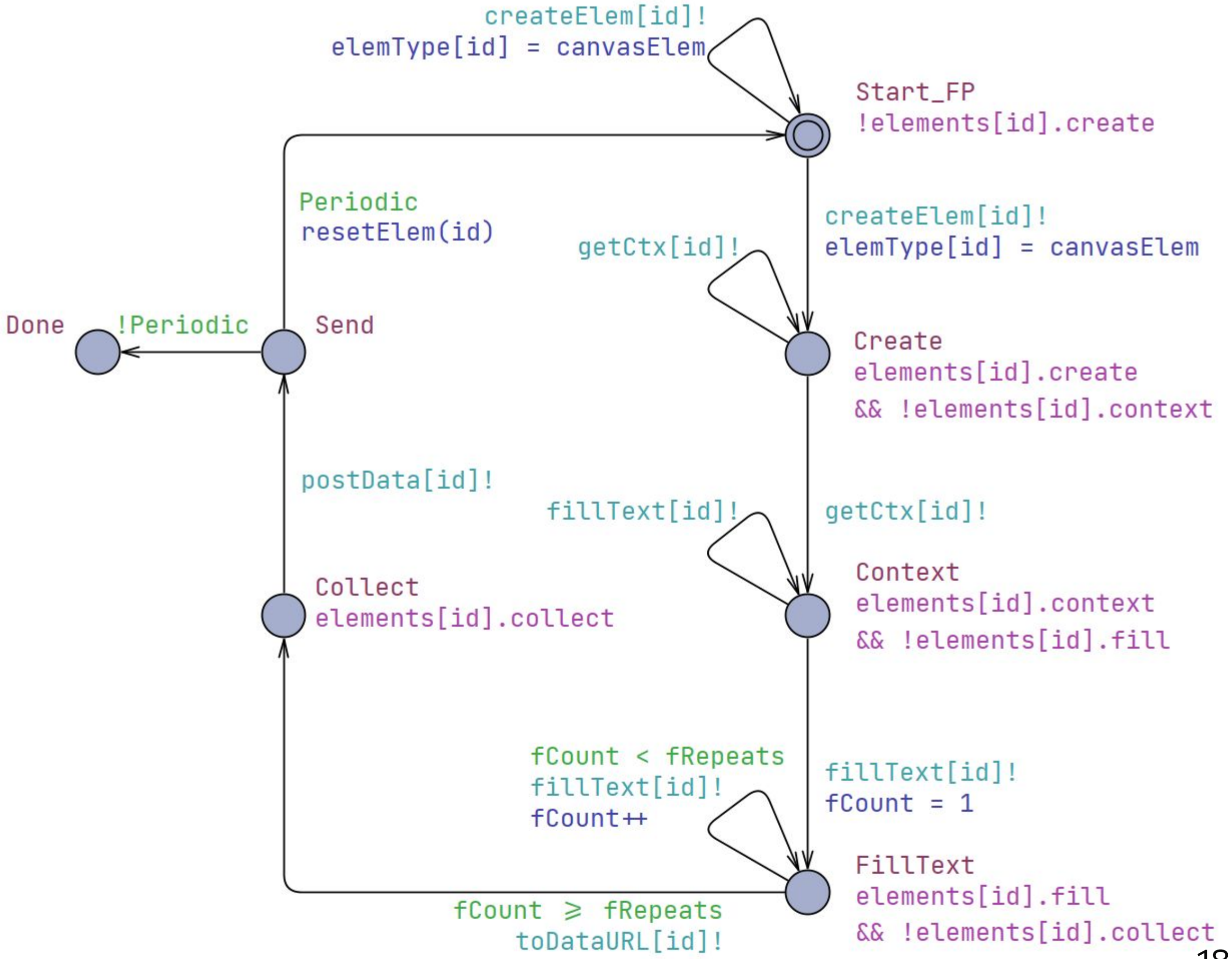
Persistent Loops

- Supports a wider variety of scripts that may not be “well formed”



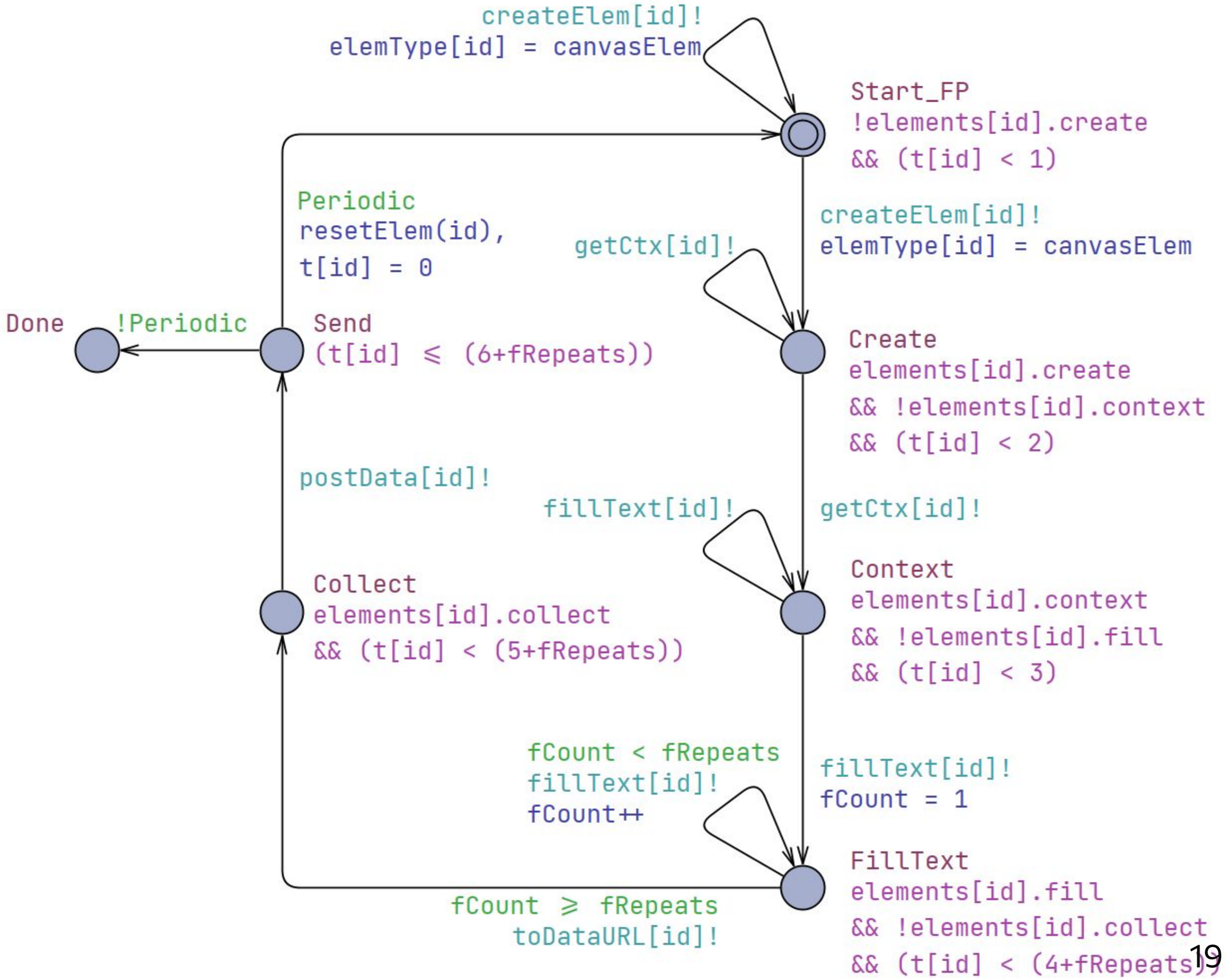
Controller Invariants

- Instrumentation for Controller
- Models IRM policy enforcement



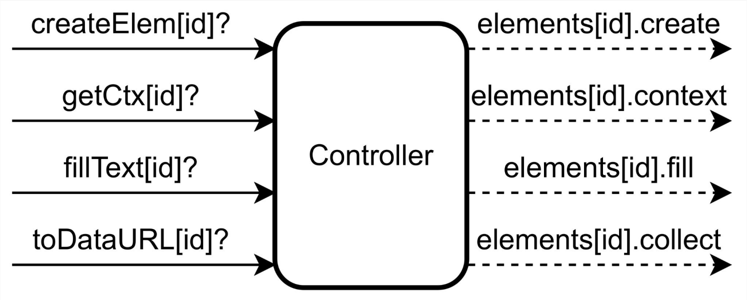
Timing Constraints

- Ensures progression, if possible
- Aids in evaluating liveness and reachability properties



Controller: Overview

- **Description:** An abstraction of an Inline Reference Monitor intercepting function calls.
- Synchronizes with Fingerprinter components



x : Funcs Monitored

y : Fingerprinter Components

- **Input**: Receive channel synchronizations.

$$f(x,y) = xy$$

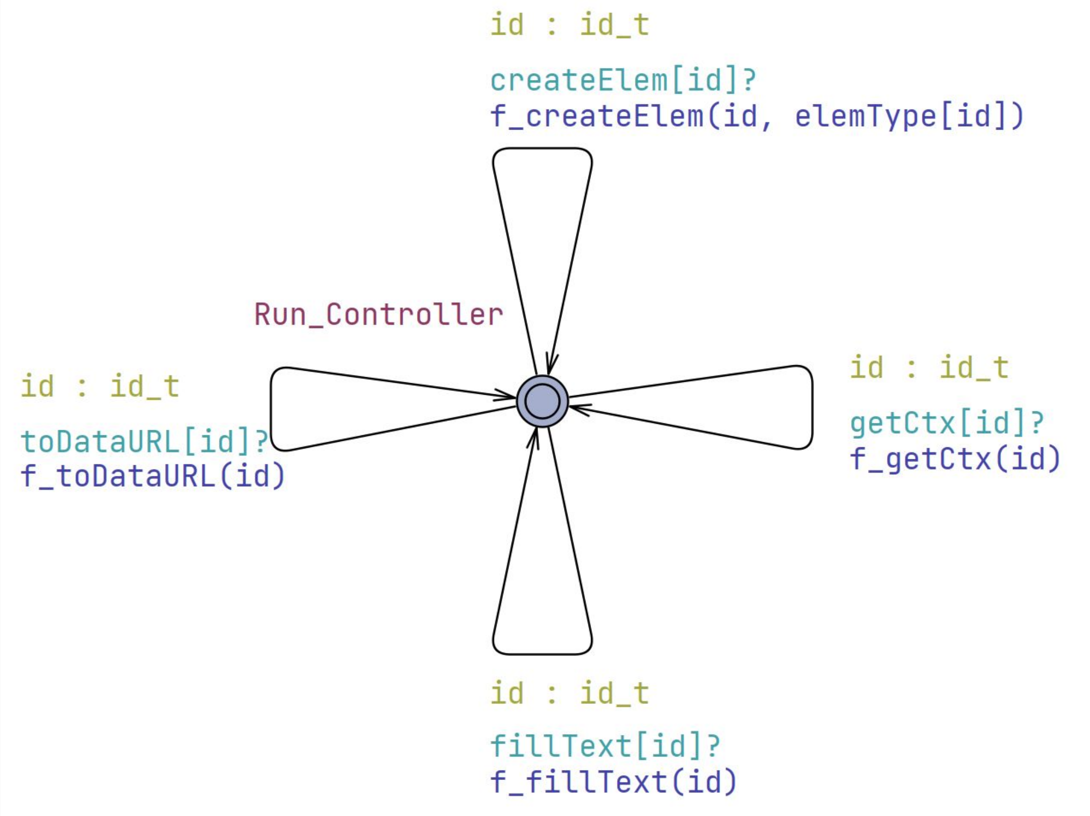
- **Output**: Set state invariants.

$$f(x,y) = xy$$

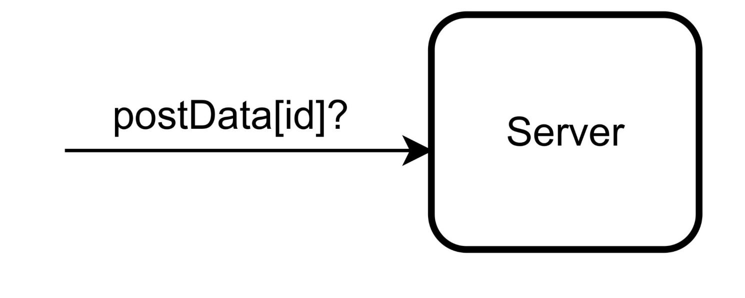
Controller: Timed Automata

Transitions:

- One for each func controlled/monitored
- **Sync**: Receive from any channel
- **Select**: Sending component ID
- **Update**: Policy Evaluation, or other actions



Server: Overview



<u>Description:</u>

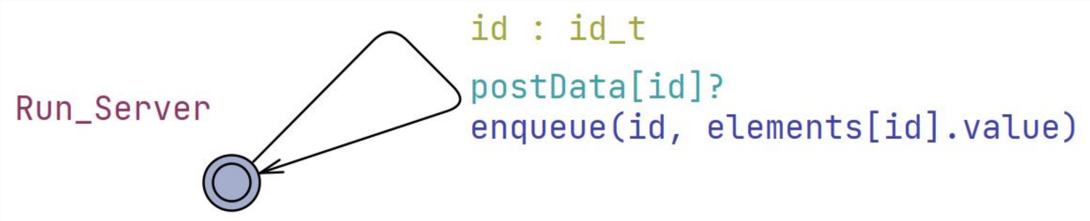
- Models a remote server and database
- A comprehensive model of the remote components is out of scope
- Combining remote components reduces state space
- Allows fingerprint values to be evaluated over time

- **Input**: Receive from data channel
- **Output**: n/a, internally stores data

Server: Timed Automata

<u>**Transitions:**</u>

- **Sync**: Receive from any channel
- **Select**: Sending component ID
- **Update**: Store data
- One data channel for each Fingerprinter component



Requirements and Policy Configuration

Informal Requirements

FP_0

No Mitigation

Allow fingerprints to be freely collected, without intervention from the Controller.

FP_1

Randomization

Allow fingerprints to be collected, but poison the data first.

FP_2

API Blocking

Do not allow fingerprints to be collected whatsoever.

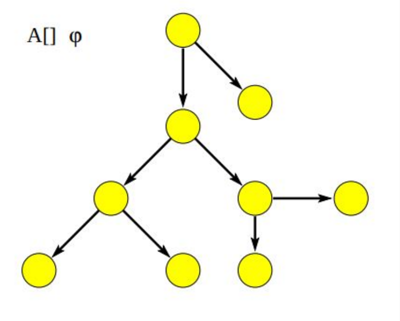
Policy Configuration

Policy	**Type**	**FP_0**	**FP_1**	**FP_2**
Create Element	Blocklist	False	False	False
Get Canvas Context	Blocklist	False	False	False
Fill Text	Blocklist	False	False	False
Collect Data	Blocklist	False	False	*True*
Poison Data	Allowlist	*True*	False	False

Verifying Formal Safety and Liveness Properties

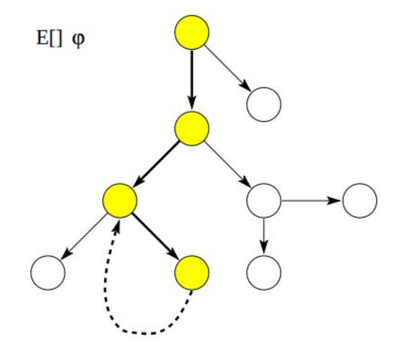
Safety Properties

`A[]φ`



- Some property is invariantly true
- φ is true in all reachable states

`E[]φ`



- Some property is *possibly* always true
- There should exist a maximal path where φ is always true

Source: UPPAAL Tutorial

Safety Properties

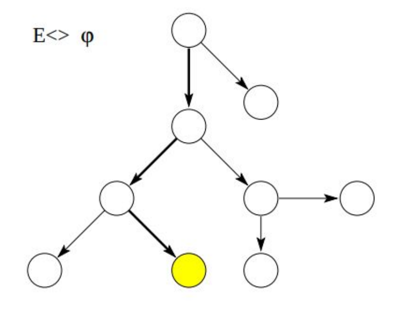
Prop.	**Sat.**	**CTL/Meaning**
A	True	**A[ ]** FP_0.Collect imply (elements[0].value > 0)
		For all reachable states, component **FP_0** being in the location *Collect* implies that its attribute value is *not* the default and is *not* poisoned.
B	True	**A[ ]** FP_1.Collect imply (elements[1].value < 0)
		For all reachable states, component **FP_1** being in the location *Collect* implies that its attribute value is poisoned.

Safety Properties

Prop.	**Sat.**	**CTL/Meaning**
C	True	**A[ ]** FP_2.Collect imply evalPolicy(p_toDataURL, 2)
		For all reachable states, component **FP_2** being in the location *Collect* implies the policy configuration allows it.
D	True	**A[ ]** !FP_2.Collect
		For all reachable states, component **FP_2** is never in the *Collect* location.
E	True	**A[ ]** Server.db[2].len == 0
		For all reachable states, the server never receives fingerprint values from **FP_2**.

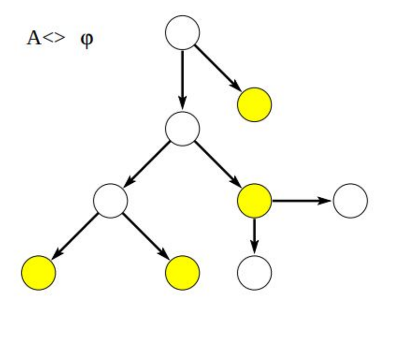
Liveness Properties

E <> φ



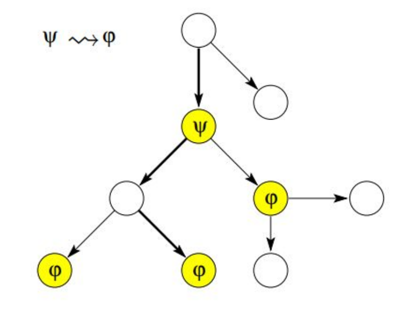
- It is *possible* for some property to be satisfied
- φ possibly can be satisfied by any reachable state

A <> φ



- Something will eventually happen
- φ is eventually satisfied

ψ ⟶ φ



- When some condition is met, eventually some property is satisfied
- Whenever ψ is satisfied, eventually φ is satisfied

Source: UPPAAL Tutorial

Liveness Properties

Prop.	**Sat.**	**CTL/Meaning**
F	True	**E< >** FP_0.Collect
		The *Collect* location is reachable in the **FP_0** component.
G	True	**E< >** FP_1.Collect
		The *Collect* location is reachable in the **FP_1** component.
H	False	**E< >** FP_2.Collect
		The *Collect* location is *not* reachable in the **FP_2** component.

Liveness Properties

Prop.	Sat.	CTL/Meaning
I	True	**A<>** ((Sever.db[0].len > 0) && (Server.db[0].entries[0] == Server.db[0].entries[1]) && (Server.db[0].entries[1] == Server.db[0].entries[2]))
		Eventually all database entries for **FP_0** are the same.
J	False	**A<>** ((Server.db[1].len > 0) && (Server.db[1].entries[0] == Server.db[1].entries[1]) && (Server.db[1].entries[1] == Server.db[1].entries[2]))
		Eventually all database entries for **FP_1** are the same.

Wrapping Up

Contributions

- Formal Models
  - Canvas Fingerprinter
  - IRM Controller
- Evaluation of Models using CTL
  - Formal properties reflect requirements of mitigation methods

Takeaways:

- Effectiveness of IRMs to enforce mitigation methods:
  - Randomization
  - Normalization
  - API Blocking
- Proof of IRM reliability
- Extensible Framework

Limitations and Future Work

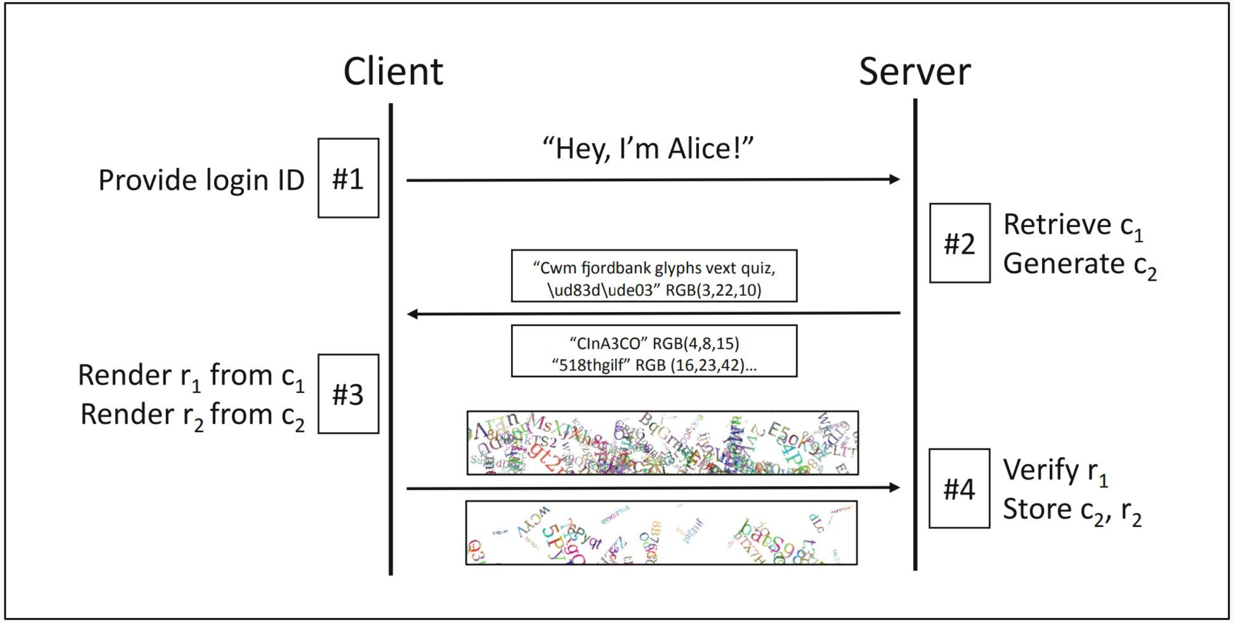
- Lack of Comprehensive Model
  - Extend our framework to support common attributes used by fingerprinters
- Attack Model
  - Evaluate minimum effective mitigation strategies
- Model-based Code Generation
  - Verified system to practical application
  - Bridge the gap between research and real-world implementations

Thank You!

External Links

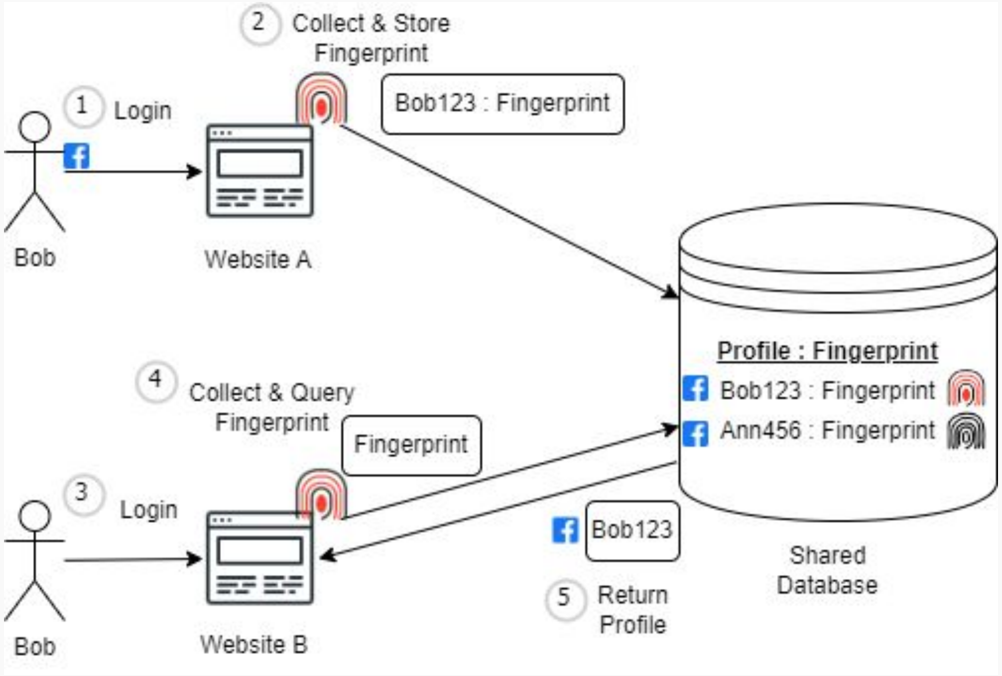
- UPPAAL Documentation
- This work's Github
- amiunique.org

Benign Application Example



A challenge/response-based authentication mechanism proposed by Laperdrix et al. (2019).

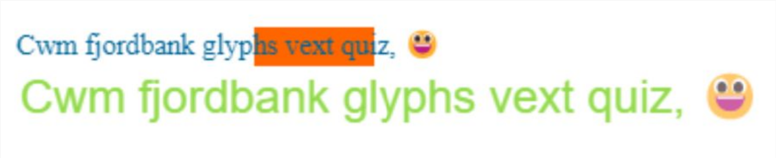
Malicious Application Example



Source: Khademi et al. (2015)

Canvas Poisoning Examples

Base Canvas Image

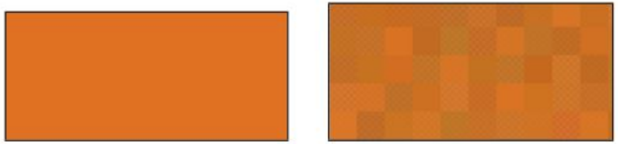


Poisoned Versions

Testing Tool Used: https://amiunique.org/

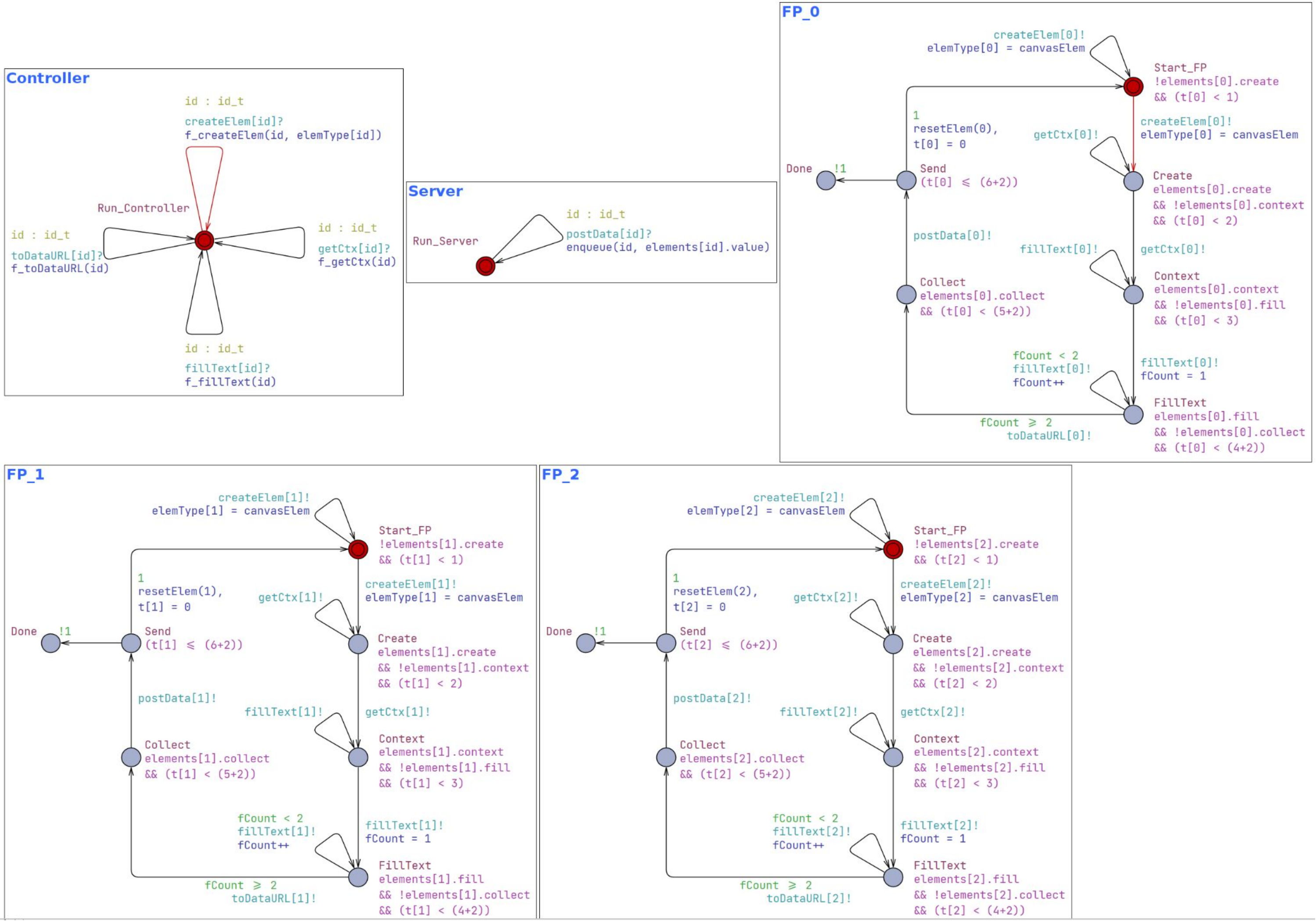
Source: Laperdrix et al. (2017)

Canvas Poisoning Examples



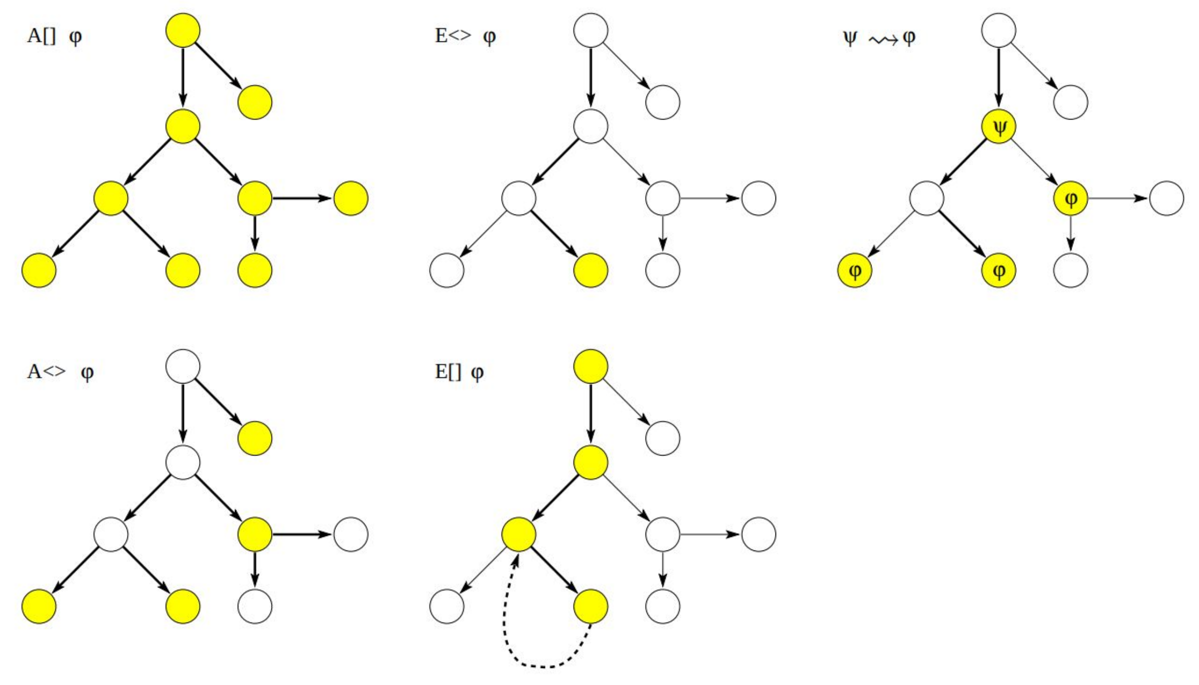
(a) *Without a poisoner* (b) *With a poisoner*

Source: Laperdrix et al. (2019)



Controller
id : id_t
createElem[id]?
f_createElem(id, elemType[id])
Run_Controller
id : id_t
toDataURL[id]?
f_toDataURL(id)
id : id_t
getCtx[id]?
f_getCtx(id)
id : id_t
fillText[id]?
f_fillText(id)
Server
Run_Server
id : id_t
postData[id]?
enqueue(id, elements[id].value)
FP_0
createElem[0]!
elemType[0] = canvasElem
Start_FP
!elements[0].create
&& (t[0] < 1)
1
resetElem(0),
t[0] = 0
createElem[0]!
elemType[0] = canvasElem
getCtx[0]!
Done
!1
Send
(t[0] ≤ (6+2))
Create
elements[0].create
&& !elements[0].context
&& (t[0] < 2)
postData[0]!
fillText[0]!
getCtx[0]!
Collect
elements[0].collect
&& (t[0] < (5+2))
Context
elements[0].context
&& !elements[0].fill
&& (t[0] < 3)
fCount < 2
fillText[0]!
fCount++
fillText[0]!
fCount = 1
FillText
elements[0].fill
&& !elements[0].collect
&& (t[0] < (4+2))
fCount ≥ 2
toDataURL[0]!
FP_1
createElem[1]!
elemType[1] = canvasElem
Start_FP
!elements[1].create
&& (t[1] < 1)
1
resetElem(1),
t[1] = 0
getCtx[1]!
createElem[1]!
elemType[1] = canvasElem
Done
!1
Send
(t[1] ≤ (6+2))
Create
elements[1].create
&& !elements[1].context
&& (t[1] < 2)
postData[1]!
fillText[1]!
getCtx[1]!
Collect
elements[1].collect
&& (t[1] < (5+2))
Context
elements[1].context
&& !elements[1].fill
&& (t[1] < 3)
fCount < 2
fillText[1]!
fCount++
fillText[1]!
fCount = 1
FillText
elements[1].fill
&& !elements[1].collect
&& (t[1] < (4+2))
fCount ≥ 2
toDataURL[1]!
FP_2
createElem[2]!
elemType[2] = canvasElem
Start_FP
!elements[2].create
&& (t[2] < 1)
1
resetElem(2),
t[2] = 0
getCtx[2]!
createElem[2]!
elemType[2] = canvasElem
Done
!1
Send
(t[2] ≤ (6+2))
Create
elements[2].create
&& !elements[2].context
&& (t[2] < 2)
postData[2]!
fillText[2]!
getCtx[2]!
Collect
elements[2].collect
&& (t[2] < (5+2))
Context
elements[2].context
&& !elements[2].fill
&& (t[2] < 3)
fCount < 2
fillText[2]!
fCount++
fillText[2]!
fCount = 1
FillText
elements[2].fill
&& !elements[2].collect
&& (t[2] < (4+2))
fCount ≥ 2
toDataURL[2]!

Computation Tree Logic (CTL) In UPPAAL



Source: UPPAAL Tutorial

Abstracting Fingerprinting Scripts