Micro-Policies for Web Session Security

Stefano Calzavara
Università Ca' Foscari Venezia
calzavara@dais.unive.it

Riccardo Focardi
Università Ca' Foscari Venezia
focardi@dais.unive.it

Niklas Grimm
Saarland University
grimm@cs.uni-saarland.de

Matteo Maffei
Saarland University
maffei@cs.uni-saarland.de

***Abstract*—Micro-policies, originally proposed to implement hardware-level security monitors, constitute a flexible and general enforcement technique, based on assigning security tags to system components and taking security actions based on dynamic checks over these tags. In this paper, we present the first application of micro-policies to web security, by proposing a core browser model supporting them and studying its effectiveness at securing web sessions. In our view, web session security requirements are expressed in terms of a simple, purely declarative information flow policy, which is then automatically translated into a micro-policy implementing it. This leads to a browser-side enforcement mechanism which is elegant, sound and flexible, while being accessible to web developers. We show how a large class of attacks against web sessions can be uniformly and effectively prevented by the adoption of this approach. Since we carefully designed micro-policies with ease of deployment in mind, we are also able to implement our proposal as a Google Chrome extension, Michrome: our experiments show that Michrome can be easily configured to enforce strong security policies without breaking the websites functionality.**

I. Introduction

The Web is nowadays the primary means of access to a plethora of online services with strict security requirements. Electronic health records and online statements of income are a well-established reality as of now, and more and more security-sensitive services are going to be supplied online in the next few years. Despite the critical importance of securing these online services, web applications and, more specifically, *web sessions* are notoriously hard to protect, since they can be attacked at many different layers.

At the network layer, man-in-the-middle attacks can break both the confidentiality and the integrity of web sessions running (at least partially) over HTTP. The standard solution against these attacks is deploying the entire web application over HTTPS with trusted certificates and, possibly, making use of HSTS [17] to prevent subtle attacks like SSL stripping. At the session implementation layer, code injection attacks (or again network attacks) can be exploited to steal authentication cookies and hijack a web session, or to compromise the integrity of the cookie jar and mount dangerous attacks like session fixation [19]. This is particularly problematic because, though the standard HttpOnly and Secure cookie attributes [3] are effective at protecting cookie confidentiality, no effective countermeasure exists as of now to ensure cookie integrity on the Web [32]. Finally, web sessions can also be attacked at the application layer: for instance, since browsers automatically attach cookies set by a website to all the requests sent to it, cross-site request forgery (CSRF) attacks can be mounted by a malicious web page to harm the integrity of the user session with a trusted web application and inject attacker-controlled messages inside it. Standard solutions against this problem include the usage of secret tokens and the validation of the Origin header attached by the browser to filter out malicious web requests [4].

In principle, it is possible to achieve a reasonable degree of security for web sessions using the current technologies, but the overall picture still exhibits several important shortcomings and it is far from being satisfactory. First, there are mechanisms like the HttpOnly cookie attribute which are easy to use, popular and effective, but lack flexibility: a cookie may either be HttpOnly or not, hence JavaScript may either be able to access it or be prevented from doing any kind of computation over the cookie value. There is no way, for instance, to let JavaScript access a cookie for legitimate computations, at the cost of disciplining its communication behaviour to prevent the cookie leakage. Then, there are defenses which are sub-optimal and not always easy to implement: this is the case for token-based protection against CSRF. Not only this approach must be directly implemented into the APIs of a web development framework to ensure that it is convenient to use, but also it is not very robust, since it fails in presence of code injection vulnerabilities which disclose the token value to the attacker. Finally, we observe that some attacks and attack vectors against web sessions are underestimated by existing standards and no effective solution against them can be deployed as of now: this is the case for many threats to cookie integrity [32]. These issues will likely be rectified with ad-hoc solutions in future standards, whenever browser vendors and web application developers become more concerned about their importance, and find a proper way to patch them while preserving the compatibility with existing websites.

In this paper, we advocate that a large class of attacks harming the security of web sessions can be provably, uniformly, and effectively prevented by the adoption of *browser-enforced security policies*, reminiscent of a dynamic typing discipline for the browser. In particular, we argue for the adoption of *micro-policies* [14] as a convenient tool to improve the security of web sessions, by disciplining the browser behaviour when interacting with security-sensitive web applications. Roughly, the specification of a micro-policy involves: (1) the definition of a set of *tags*, used to label selected elements of the web ecosystem, like URLs, cookies, network connections, etc., and (2) the definition of a *transfer* function, defining which operations are permitted by the browser based on the tags and how tags are assigned to browser elements after a successful operation. This kind of security policies has already proved

helpful for deploying hardware-level security monitors and nicely fits existing web security solutions, like cookie security attributes [3] and whitelist-based defenses in the spirit of the Content Security Policy [30].

Though previous work has already proposed browser-side security policies as a viable approach for protecting the Web [18], [21], [26], [29], [13], we are the first to carry out a foundational study on a possible extension of a web browser with support for micro-policies and discuss web session security as an important application for this framework. There are many different ways to deploy micro-policies in web browsers, but our proposal is driven by two main design goals aimed at simplifying a large-scale adoption. First, it is *light-weight* and it does not need to change existing web browsers too much, as testified by a prototype implementation of our approach as an extension for Google Chrome. Second, it is *practical* to use: although our technical development is based on a non-trivial theory, we strive for supporting declarative policies for web session security, which do not significantly deviate from the tools and the abstractions which web developers already appreciate and use today. We thus propose to express web session security requirements in terms of a simple, purely declarative information flow policy, which can be automatically translated into a micro-policy implementing it.

Our contributions can be summarized as follows:

1) we design FF^τ, a core model of a web browser extended with support for micro-policies. We define the operational behaviour of FF^τ using a small-step reactive semantics in the spirit of previous formal work on browser security [8], [7], [11]. The semantics of FF^τ is parametric with respect to an arbitrary set of tags and the definition of a transfer function operating on these tags;
2) we instantiate the set of tags of FF^τ to intuitive information flow labels and we characterize standard attackers from the web security literature in terms of these labels. We then discuss how to translate simple information flow policies for web session security into micro-policies which enforce them: this is crucial to ensure that most web developers can benefit from our proposal;
3) we discuss example applications of our theory by revisiting known attacks against web sessions and discussing limitations of existing solutions. We then show how these issues are naturally and more effectively solved by our enforcement technique;
4) we develop a prototype implementation of our proposal as a standard Google Chrome extension, Michrome, and we run a set of experiments testing its practicality. Our experiments show that Michrome can be easily configured to enforce strong security policies without breaking the functionality of existing websites.

Michrome and a technical report including full proofs are available online [2].

II. Key Ideas

In this section, we give an intuitive overview of the most salient aspects of our framework. We model the browser as a *reactive system*, transforming a stream of input events into a stream of output events. Our sets of input and output events include key elements of standard web browsing, like HTTP(S) requests, responses and redirects. For example, the input stream $I = [\mathsf{load}(u), \mathsf{doc\_resp}_n(u : \{\mathsf{ck}(k,v)^\ell\}, \mathsf{unit})]$, instructs the browser to establish a new network connection n to the URL u and retrieve from that connection a response including a cookie $\mathsf{ck}(k,v)^\ell$ and an empty document unit. The cookie, formally seen as a mapping between key k and value v, has a security *label* ℓ, consisting of a confidentiality policy and an integrity policy. For instance, the confidentiality policy $\{\mathsf{https}(d)\}$ expresses that the value of the cookie should only have a visible import for an attacker who is able to decrypt the HTTPS communication with the domain d setting the cookie.

We argue for the adoption of browser-side micro-policies enforcing this kind of security policies. Policies are formalized in terms of *reactive non-interference*, a property dictating that two similar input streams must always be transformed into two similar output streams. Confidentiality is characterized by identifying suitable similarity relations on input streams, based on what the attacker is able to observe about the corresponding output streams. For instance, consider a network attacker with full control of the HTTP traffic: to formalize that cookies with the confidentiality policy $\{\mathsf{https}(d)\}$ have no visible import for the attacker, the stream similarity on inputs may relate streams which are identical except for the value of these cookies.

As an example, let u_s be a HTTPS URL on domain d, it is safe to consider the following two input streams, differing in the cookie value, as similar:

$$I_1 = [\mathsf{load}(u), \mathsf{doc\_resp}_n(u : \{\mathsf{ck}(k,v)^\ell\}, \mathsf{unit}), \mathsf{load}(u_s)]$$
$$I_2 = [\mathsf{load}(u), \mathsf{doc\_resp}_n(u : \{\mathsf{ck}(k,v')^\ell\}, \mathsf{unit}), \mathsf{load}(u_s)]$$

The reason is that the browser will react to these input streams by producing the following output streams:

$$O_1 = [\mathsf{doc\_req}(u : \emptyset), \bullet, \mathsf{doc\_req}(u_s : \mathsf{ck}(k,v)^\ell)]$$
$$O_2 = [\mathsf{doc\_req}(u : \emptyset), \bullet, \mathsf{doc\_req}(u_s : \mathsf{ck}(k,v')^\ell)]$$

These streams include a document request to u without cookies, a dummy event as a reaction to the empty document, and a document request to u_s including the previously received cookie, which is the normal behaviour of a web browser. Since u_s is a HTTPS URL, the last events of O_1 and O_2 cannot be distinguished by a network attacker, hence the two output streams are similar and there is no violation to reactive non-interference.

But what if the $\mathsf{load}(u_s)$ event in I_1, I_2 was replaced by $\mathsf{load}(u_h)$, where u_h is a HTTP URL on domain d? The behaviour of the browser will be restricted by the underlying micro-policy for non-interference, forcing the production of two output streams not including any cookie in the last event to ensure similarity upon output. These restrictions are enforced by assigning labels to browser components (cookies, connections, scripts...) and by performing runtime label checks upon event processing, reminiscent of a dynamic typing discipline for the browser. Interestingly, simple and intuitive policies like the one we discussed are expressive enough to prevent a large class of known attacks against web sessions. Moreover, despite their simplicity, these policies are actually stronger

than currently deployed web solutions (cf. Section VI), providing an expressive mechanism to formally define and enforce confidentiality and integrity properties for web sessions.

III. Background on Reactive Systems

Web browsers can be formalized in terms of constrained labelled transition systems known as *reactive systems* [8], [7]. Intuitively, a reactive system is an event-driven state machine which waits for an input, produces a sequence of outputs in response to it, and repeats the process indefinitely.

Definition 1 (Reactive System)**.** *A* reactive system *is a tuple* $R = \langle \mathcal{C}, \mathcal{P}, \mathcal{I}, \mathcal{O}, C_0, \longrightarrow \rangle$*, where* $\mathcal{C}$ *and* $\mathcal{P}$ *are disjoint sets of consumer and producer states respectively, while* $\mathcal{I}$ *and* $\mathcal{O}$ *are disjoint sets of input and output events respectively. The consumer state* C_0 *is the initial state of the system and the last component,* $\longrightarrow$*, is a labelled transition relation over the set of states* $\mathcal{Q} \triangleq \mathcal{C} \cup \mathcal{P}$ *and the set of events* $\mathcal{A} \triangleq \mathcal{I} \cup \mathcal{O}$*, subject to the following constraints:*

1) *if* $C \in \mathcal{C}$ *and* $C \xrightarrow{a} Q$*, then* $a \in \mathcal{I}$ *and* $Q \in \mathcal{P}$*;*
2) *if* $P \in \mathcal{P}$ *and* $P \xrightarrow{a} Q$ *for some* $Q \in \mathcal{Q}$*, then* $a \in \mathcal{O}$*;*
3) *if* $C \in \mathcal{C}$ *and* $i \in \mathcal{I}$*, then there exists* $P \in \mathcal{P}$ *s.t.* $C \xrightarrow{i} P$*;*
4) *if* $P \in \mathcal{P}$*, then there exist* $o \in \mathcal{O}$ *and* $Q \in \mathcal{Q}$ *s.t.* $P \xrightarrow{o} Q$*.*

We define *streams* of events through the coinductive interpretation of the following grammar: $S ::= [\,] \mid a :: S$. The semantics of a reactive system R is defined in terms of *traces* (I, O), where I is a stream of input events and O is a stream of output events generated by R as the result of processing I.

Definition 2 (Trace)**.** *Let* $R = \langle \mathcal{C}, \mathcal{P}, \mathcal{I}, \mathcal{O}, C_0, \longrightarrow \rangle$ *be a reactive system. Given an input stream* I*, the state* $Q \in \mathcal{C} \cup \mathcal{P}$ *generates the output stream* O *iff the judgement* $Q(I) \Downarrow O$ *can be coinductively derived by the following inference rules:*

$$\text{(C-Nil)}\ \frac{}{C([\,]) \Downarrow [\,]} \qquad \text{(C-In)}\ \frac{C \xrightarrow{i} P \quad P(I) \Downarrow O}{C(i :: I) \Downarrow O} \qquad \text{(C-Out)}\ \frac{P \xrightarrow{o} Q \quad Q(I) \Downarrow O}{P(I) \Downarrow o :: O}$$

We say that R *generates the* trace (I, O) *iff* $C_0(I) \Downarrow O$*.*

A natural definition of information security for reactive computations is *reactive non-interference* [8]. We presuppose the existence of a label pre-order $(\mathcal{L}, \sqsubseteq)$ and we represent the attacker as a label $\ell \in \mathcal{L}$, defining its abilities to observe and corrupt data. These abilities are formalized by a label-indexed family of predicates rel_ℓ, identifying security relevant events, and a label-indexed family of similarity relations $\sim_\ell$, identifying indistinguishable events. We collect these two families of relations in a *policy* $\pi = \langle rel_\ell, \sim_\ell \rangle$.

Given a policy π, we define a notion of *similarity* between two streams of events for an attacker ℓ as follows.

Definition 3 (ID-similarity)**.** *Two streams of events* S *and* S' *are* ID-similar *(*similar *for short) for* ℓ *under* $\pi = \langle rel_\ell, \sim_\ell \rangle$ *iff the judgement* $S \approx^\pi_\ell S'$ *can be coinductively derived by the following inference rules:*

$$\text{(S-Empty)}\ [\,] \approx^\pi_\ell [\,] \qquad \text{(S-Match)}\ \frac{rel_\ell(s) \quad rel_\ell(s') \quad s \sim_\ell s' \quad S \approx^\pi_\ell S'}{s :: S \approx^\pi_\ell s' :: S'}$$

$$\text{(S-Left)}\ \frac{\neg rel_\ell(s) \quad S \approx^\pi_\ell S'}{s :: S \approx^\pi_\ell S'} \qquad \text{(S-Right)}\ \frac{\neg rel_\ell(s) \quad S \approx^\pi_\ell S'}{S \approx^\pi_\ell s :: S'}$$

Intuitively, a reactive system satisfies non-interference under a policy π if and only if, whenever it is fed two similar input streams, it produces two similar output streams for all the possible attackers (labels).

Definition 4 (Reactive Non-interference)**.** *A reactive system is* non-interferent *under* π *iff, for all labels* ℓ *and all its traces* (I, O) *and* (I', O') *such that* $I \approx^\pi_\ell I'$*, we have* $O \approx^\pi_\ell O'$*.*

Reactive non-interference has been proposed in the past as a useful security baseline to prove protection against common attacks against web sessions, including authentication cookie theft [16], [9], [10] and cross-site request forgery [20].

IV. Micro-Policies for Browser-Side Security

Our model FF^τ is inspired by existing formal models for web browsers based on reactive systems [7], [11]. It is an extension of the Flyweight Firefox model [11] with tags and support for enforcing micro-policies based on them.

A. Syntax

A *map* M is a partial function from keys to values. We let $\{\}$ stand for the empty map and we let $dom(M)$ denote the *domain* of M, i.e., the set of keys bound to a value in M. We let $M_1 \uplus M_2$ be the union of two maps with disjoint domains.

1) Tags: We presuppose the existence of a denumerable set of tags *Tags* and we let τ range over them. We do not put any restriction on the format of these tags, though we instantiate them to a specific format in the next section.

2) Terms: We presuppose a set of domain names $\mathcal{D}$ (ranged over by d) and a set of strings $\mathcal{S}$ (ranged over by s). The *signature* for the set of terms $\mathcal{T}$ is:

$$\Sigma = \{\mathsf{http}, \mathsf{https}, \mathsf{url}(\cdot,\cdot,\cdot), \mathsf{ck}(\cdot,\cdot,\cdot)\} \cup \mathcal{D} \cup \mathcal{S} \cup \mathit{Tags}.$$

Let $\mathcal{X}$ be a set of variables and $\mathcal{N}$ be a set of names, the set of terms $\mathcal{T}$ is defined as follows: if $t \in \mathcal{X} \cup \mathcal{N}$, then $t \in \mathcal{T}$; if f is an n-ary function symbol in Σ and $\{t_1, \ldots, t_n\} \subseteq \mathcal{T}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}$.

3) URLs: We let $\mathcal{U} \subseteq \mathcal{T}$ be the set of the URLs, i.e., the set of terms of the form $\mathsf{url}(t, d, s)$ with $t \in \{\mathsf{http}, \mathsf{https}\}$. Given $u = \mathsf{url}(t, d, s)$, let $prot(u) = t$, $host(u) = d$ and $path(u) = s$. We assume that each URL $u \in \mathcal{U}$ comes with an associated tag, returned by a function $tag : \mathcal{U} \to \mathit{Tags}$. For instance, the *tag* function may assign the Secure tag to HTTPS pages and the Insecure tag to HTTP pages: this information can be used to apply different micro-policies in the browser.

4) Cookies: We let $\mathcal{CK} \subseteq \mathcal{T}$ be the set of cookies, i.e., the set of terms of the form $\mathsf{ck}(s, s', \tau)$. Formally, cookies are just key-value pairs (s, s') extended with a tag τ. We assume this tag is assigned by a function $\kappa : \mathcal{D} \times \mathcal{S} \to \mathit{Tags}$, so that cookies with the same key set by the same domain must have the same tag. We typically use the more evocative notation $\mathsf{ck}(k, v)^\tau$ to represent cookies. Given $ck = \mathsf{ck}(k, v)^\tau$, we let $\mathit{key}(ck) = k$ and $\mathit{value}(ck) = v$.

5) Scripts: We let values v, expressions e and scripts scr be defined by the following productions:

$$\begin{array}{llll} \textit{Values} & v & ::= & t \mid \mathsf{unit} \mid \lambda x.e \\ \textit{Expr.} & e & ::= & v\, v' \mid \mathsf{let}\ x = e\ \mathsf{in}\ e' \mid \mathsf{get\text{-}ck}(v) \\ & & \mid & \mathsf{set\text{-}ck}(v, v') \mid \mathsf{xhr}(v, v') \mid v \\ \textit{Scripts} & scr & ::= & \lceil e \rfloor^\tau_{@u} \end{array}$$

A script $\lceil e \rfloor^\tau_{@u}$ is an expression e running in the *origin* u with an associated tag τ. The origin u is needed to enforce the same-origin policy on accesses to the cookie jar, while the tag τ is used to enforce micro-policies on the script.

The expression $(\lambda x.e)\, v$ evaluates to $e\{v/x\}$; the expression $\mathsf{let}\ x = e\ \mathsf{in}\ e'$ first evaluates e to a value v and then behaves as $e'\{v/x\}$; the expression $\mathsf{get\text{-}ck}(k)$ returns the value of the cookie with key k, provided that the tag assigned to the cookie allows this operation; the expression $\mathsf{set\text{-}ck}(k, v)$ stores the cookie $\mathsf{ck}(k, v)^\tau$ in the cookie jar, where $\tau = \kappa(\mathit{host}(u), k)$ is a tag derived by the origin u in which the expression is running and the cookie key k; again, the setting operation may fail due to the enforcement of a micro-policy. The expression $\mathsf{xhr}(u, \lambda x.e)$ sends an AJAX request to u and, when a value v is available as a response, it runs $e\{v/x\}$ in the same origin of the script which sent the request. Notably, micro-policies may also be used to constrain AJAX communication in FF^τ. For simplicity, in our model we assimilate to AJAX requests any network request which may be triggered by a script, e.g., the request for an image triggered by the insertion of a markup element in the page where the script is running.

6) Events: Input events i are defined as follows:

$$\begin{array}{lll} i & ::= & \mathsf{load}(u) \\ & \mid & \mathsf{doc\_resp}_n(u : CK, e) \mid \mathsf{doc\_redir}_n(u : CK, u') \\ & \mid & \mathsf{xhr\_resp}_n(u : CK, v) \mid \mathsf{xhr\_redir}_n(u : CK, u'). \end{array}$$

The event $\mathsf{load}(u)$ models a user navigating the browser to the URL u: the browser opens a new network connection to u, sends a HTTP(S) request and then waits for a corresponding HTTP(S) response to process over the connection. The event $\mathsf{doc\_resp}_n(u : CK, e)$ represents the reception of a document response from u, including a set of cookies CK to set and an expression e to run in the origin u, which leads to the execution of a new script. The event is annotated with the name n of the network connection where the response is received: this connection gets closed when processing the event. The event $\mathsf{doc\_redir}_n(u : CK, u')$ models the reception of a HTTP(S) redirection from u to u' along the connection n, setting the set of cookies CK; the event keeps the connection open, while pointing it to u'. A similar intuition applies to XHR responses and redirects. For simplicity, we use $\mathsf{net\_resp}_n(u : CK, e)$ to stand for any network response, including redirects.

Output events o are defined as follows:

$$o \quad ::= \quad \bullet \mid \mathsf{doc\_req}(u : CK) \mid \mathsf{xhr\_req}(u : CK).$$

The event $\bullet$ represents a silent reaction to an input event with no visible side-effect. The event $\mathsf{doc\_req}(u : CK)$ models a document request sent to u, including the set of cookies CK. The event $\mathsf{xhr\_req}(u : CK)$ models an XHR request sent to u, including the set of cookies CK. We let $\mathsf{net\_req}(u : CK)$ represent an arbitrary network request when we do not need to precisely identify its type.

7) States: Browser states are tuples $Q = \langle K, N, H, T, O \rangle$:

$$\begin{array}{llll} \textit{Cookie Jar} & K & ::= & \{\} \mid K \uplus \{d : CK\}, \\ \textit{Connections} & N & ::= & \{\} \mid N \uplus \{n^\tau : u\} \\ \textit{Handlers} & H & ::= & \{\} \mid H \uplus \{n^\tau : (u', \lceil \lambda x.e \rfloor_{@u})\}, \\ \textit{Tasks} & T & ::= & \mathsf{wait} \mid scr, \\ \textit{Outputs} & O & ::= & [\,] \mid o :: O'. \end{array}$$

The cookie jar K maps domain names to the cookies they set in the browser. The network connection store N keeps track of the pending document requests: if $\{n^\tau : u\} \in N$, then the browser is waiting for a document response from u over the connection n. Notice that the network connection includes a tag τ, which makes it possible to enforce micro-policies on it. The handler store H tracks pending XHR requests: if $H(n^\tau) = (u', \lceil \lambda x.e \rfloor_{@u})$, the continuation $\lambda x.e$ is ready to be run in the origin u when an XHR response is received from u' over the connection n. Also these connections have an associated tag.

We use T to represent *tasks*: if $T = \lceil e \rfloor^\tau_{@u}$, then a script is running; if $T = \mathsf{wait}$, no script is running. Finally, O is a buffer of output events, needed to interpret FF^τ as a reactive system: let $Q = \langle K, N, H, T, O \rangle$ be a consumer state when $T = \mathsf{wait}$ and $O = [\,]$, otherwise let Q be a producer state. We let $C_0 = \langle \{\}, \{\}, \{\}, \mathsf{wait}, [\,] \rangle$ be the initial state of FF^τ.

B. Reactive Semantics

The reactive semantics of FF^τ is parametric with respect to a partial function *transfer* [14], which is roughly a tag-based security monitor operating on the browser model. The transfer function we consider has the following format:

$$\mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) = (\tau_n, \tau_{ci}, \tau_{co}, \tau_s),$$

where τ_1 and τ_2 are the (at most two) arguments passed to the function when the browser model processes an event of type $\mathsf{event\_type}$, while $\tau_n, \tau_{ci}, \tau_{co}, \tau_s$ are the (at most four) tags assigned to the new browser elements which are instantiated as the result of the event processing. Specifically, τ_n is the tag of the new network connection which is created, τ_{ci} is the tag passed to the cookie jar when storing some new cookies, τ_{co} is the tag passed to the cookie jar when retrieving the cookies to be attached to HTTP(S) requests, and τ_s is the tag of the new running script. If any of these elements is not needed when processing an event of a given type, e.g., since no new cookie is set, we replace it with a dash $(-)$. If the transfer function is undefined for a given set of arguments, an operation is not permitted by the security monitor. For space reasons, the full

reactive semantics of FF^τ is given in Appendix A. Here, we just present the main ideas needed to understand the paper.

A set of transitions of the form $C \xrightarrow{i} P$ describes how the consumer state C reacts to the input event i by evolving into a producer state P. Conversely, a set of transitions of the form $P \xrightarrow{o} Q$ describes how a producer state P generates an output event o and evolves into another state Q. Most of the transitions invoke the transfer function before being fired, with the following intuitive semantics:

- *transfer*$(\mathsf{load}, \tau_u, -) = (\tau_n, -, \tau_{co}, -)$: invoked when a URL u such that *tag*$(u) = \tau_u$ is loaded. The event creates a new network connection with tag τ_n and uses tag τ_{co} to access the cookie jar and retrieve the cookies to be attached to the document request sent to u;
- *transfer*$(\mathsf{doc\_resp}, \tau_n, -) = (-, \tau_{ci}, -, \tau_s)$: invoked when a document response is received over a network connection with tag τ_n. The event uses tag τ_{ci} to access the cookie jar and set the cookies received in the response, while tag τ_s is given to the new script which is executed as the result of processing the response;
- *transfer*$(\mathsf{doc\_redir}, \tau_n, \tau_u) = (\tau_m, \tau_{ci}, \tau_{co}, -)$: invoked when a document redirect is received over a network connection with tag τ_n, asking the browser to load a URL u such that *tag*$(u) = \tau_u$. As a result, the tag of the network connection is changed from τ_n to τ_m. The tags τ_{ci} and τ_{co} are used to access the cookie jar: specifically, τ_{ci} is used to set the cookies received along with the processed redirect, while τ_{co} is used to get the cookies to be sent to u upon redirection;
- *transfer*$(\mathsf{xhr\_resp}, \tau_n, -) = (-, \tau_{ci}, -, \tau_s)$: similar to the case for $\mathsf{doc\_resp}$, but for XHR responses;
- *transfer*$(\mathsf{xhr\_redir}, \tau_n, \tau_u) = (\tau_m, \tau_{ci}, \tau_{co}, -)$: similar to the case for $\mathsf{doc\_redir}$, but for XHR redirects;
- *transfer*$(\mathsf{send}, \tau_s, \tau_u) = (\tau_n, -, \tau_{co}, -)$: invoked when a script with tag τ_s sends an XHR request to a URL u such that *tag*$(u) = \tau_u$. Tag τ_n is given to the new network connection which is opened by the script, while τ_{co} is used to access the cookie jar and get the cookies to be sent to u;
- *transfer*$(\mathsf{get}, \tau_r, \tau_c) = (-, -, -, -)$: invoked when a cookie with tag τ_c is read from the cookie jar. Here, τ_r is the tag modelling the security assumptions about the reader: for instance, when cookies are fetched by the browser for inclusion in a HTTP(S) request to u, this tag could correspond to the protocol of u;
- *transfer*$(\mathsf{set}, \tau_w, \tau_c) = (-, -, -, -)$: invoked when a cookie with tag τ_u is written into the cookie jar. Similarly to the previous case, τ_w is the tag modelling the security assumptions about the writer.

If the transfer function is undefined for a specific set of tags, the corresponding transitions $C \xrightarrow{i} P$ and $P \xrightarrow{o} Q$ just lead to a dummy producer state firing the dummy event $\bullet$.

V. Enforcing Reactive Non-Interference

The operational semantics of FF^τ is parametric with respect to an arbitrary set of tags and a transfer function. Here, we instantiate these parameters to show that FF^τ can enforce a useful security property, i.e., reactive non-interference.

A. Labels, Policies and Threat Model

We define different threat models for the Web in terms of labels from a pre-order $(\mathcal{L}, \sqsubseteq)$, as required by the definition of reactive non-interference. We start by introducing *simple labels*, which we use to express confidentiality and integrity policies. A simple label l is a (possibly empty) set of elements of the form $\mathsf{http}(d)$ or $\mathsf{https}(d)$ for some domain name $d \in \mathcal{D}$:

$$l ::= \emptyset \mid \{\mathsf{http}(d)\} \mid \{\mathsf{https}(d)\} \mid l \cup l.$$

Intuitively, simple labels define sets of endpoints which are allowed to read/write a given datum or to observe/produce a given event. A *label* $\ell = (l_C, l_I)$ is a pair of simple labels, combining confidentiality and integrity. We write $C(\ell)$ for l_C and $I(\ell)$ for l_I. We let $\ell \sqsubseteq \ell'$ iff $C(\ell) \subseteq C(\ell')$ and $I(\ell) \subseteq I(\ell')$. Simple labels form a bounded lattice under set inclusion, while labels form a bounded lattice under $\sqsubseteq$: the bottom and top elements are $\bot_s = \emptyset$, $\top_s = \{\mathsf{http}(d), \mathsf{https}(d) \mid d \in \mathcal{D}\}$, $\bot = (\bot_s, \bot_s)$ and $\top = (\top_s, \top_s)$.

We assign (simple) labels to URLs, so that it is easy to define which events an attacker can observe and/or corrupt. For a URL $u \in \mathcal{U}$ with *host*$(u) = d$, we let:

- *msg_label*$(u) = \{\mathsf{http}(d)\}$ iff *prot*$(u) = \mathsf{http}$;
- *msg_label*$(u) = \{\mathsf{https}(d)\}$ iff *prot*$(u) = \mathsf{https}$;
- *evt_label*$(u) = \{\mathsf{http}(d)\}$.

We use these functions to define the capabilities of an attacker ℓ. The *presence* of a message sent to u is visible to ℓ whenever *evt_label*$(u) \subseteq C(\ell)$, while the *content* of the message is only disclosed if also *msg_label*$(u) \subseteq C(\ell)$. If *evt_label*$(u) \subseteq C(\ell)$ while *msg_label*$(u) \not\subseteq C(\ell)$, the attacker is aware of the presence of all messages sent to u, but he has no access to their contents; the presence of a message may be used to create a side-channel and leak information through an implicit flow. As to integrity, a message coming from u can be *forged* by an attacker ℓ if and only if *msg_label*$(u) \subseteq I(\ell)$. There is no distinction between message presence and message content when it comes to integrity.

Based on this informal description, the following well-formation hypothesis we make on the attacker should be clear. It ensures that we cannot model attackers who are not aware of the presence of a message, but still have access to its contents.

Definition 5 (Well-formed Attacker)**.** *An attacker ℓ is* well-formed *if and only if, for all domains $d \in \mathcal{D}$, $\mathsf{https}(d) \in C(\ell)$ implies $\mathsf{http}(d) \in C(\ell)$.*

From now on, we always implicitly consider only well-formed attackers. It is easy to represent using labels several popular web security attackers:

1) a *web attacker* on domain d is defined by:

$$\ell_w(d) \triangleq (\{\mathsf{http}(d), \mathsf{https}(d)\}, \{\mathsf{http}(d), \mathsf{https}(d)\})$$

2) a *passive network attacker* is defined by:

$$\ell_{pn} \triangleq (\{\mathsf{http}(d) \mid d \in \mathcal{D}\}, \emptyset)$$

TABLE I Attacker capabilities for ℓ

Visibility of outputs:

$$\frac{\mathit{evt\_label}(u) \subseteq C(\ell)}{\mathit{vis}_\ell(\mathsf{net\_req}(u : CK))}$$

Indistinguishability of outputs:

$$\frac{\mathit{msg\_label}(u) \not\subseteq C(\ell)}{\mathsf{net\_req}(u : CK) \sim_\ell^C \mathsf{net\_req}(u : CK')}$$

Taintedness of inputs:

$$\frac{\mathit{msg\_label}(u) \subseteq I(\ell)}{\mathit{tnt}_\ell(\mathsf{net\_resp}_n(u : CK, e))}$$

3) an *active network attacker* is defined by:

$$\ell_{an} \triangleq (\{\mathsf{http}(d) \mid d \in \mathcal{D}\}, \{\mathsf{http}(d) \mid d \in \mathcal{D}\}).$$

To formalize the previous intuitions, we introduce a few simple ingredients: a *visibility* predicate vis_ℓ on output events, a binary *indistinguishability* relation $\sim_\ell^C$ on output events, and a *taintedness* predicate tnt_ℓ on input events. These are defined in Table I. The indistinguishability relation identifies network requests which only differ for contents (cookies) which are not visible to the attacker, because encrypted. We implicitly assume that two indistinguishable requests have the same type.

We then define two specific classes of non-interference policies, which correctly capture the attacker capabilities we described. Our non-interference results will be restricted to these two classes of policies.

Definition 6 (Confidentiality Policy)**.** *A* confidentiality policy *is a pair* $\pi_C = \langle \mathit{rel}_\ell, \sim_\ell \rangle$ *such that:*

1) $\forall o \in \mathcal{O} : \mathit{rel}_\ell(o) \triangleq \mathit{vis}_\ell(o)$*;*
2) $\forall o, o' \in \mathcal{O} : o \sim_\ell o' \Leftrightarrow o = o' \vee o \sim_\ell^C o'$*.*

Definition 7 (Integrity Policy)**.** *An* integrity policy *is a pair* $\pi_I = \langle \mathit{rel}_\ell, \sim_\ell \rangle$ *such that:*

1) $\forall i \in \mathcal{I} : \mathit{rel}_\ell(i) \triangleq \neg \mathit{tnt}_\ell(i)$*;*
2) $\forall i, i' \in \mathcal{I} : i \sim_\ell i' \Leftrightarrow i = i'$*.*

B. A Canonic Transfer Function for Non-Interference

Our goal is to instantiate the operational semantics of FF^τ with a transfer function that enforces confidentiality and integrity policies. In principle, we could let web developers provide selected entries of the transfer function, defining the browser behaviour upon interaction with their own websites, but this would be quite inconvenient for them. We believe that web developers need a more effective and declarative way to specify their desired confidentiality and integrity policies. In our view, web developers should only:

1) assign security labels to the cookies they set. This is not a hard task, since web developers are already familiar with cookie security attributes like HttpOnly and Secure, and the format of the labels is pretty intuitive;
2) assign security labels to the URLs they control. We argue that also this is not hard to understand for web developers, since this kind of policies is close in spirit to standard CSP specifications [30];

These labels define the expected security properties for cookies and network connections:

1) *cookie secrecy*: if a cookie has label ℓ, its value can only be disclosed to an attacker ℓ' such that $C(\ell) \cap C(\ell') \neq \emptyset$;
2) *cookie integrity*: if a cookie has label ℓ, it can only be set or modified by an attacker ℓ' such that $I(\ell) \cap I(\ell') \neq \emptyset$;
3) *session confidentiality*: if a URL u has label ℓ, an attacker ℓ' can observe that the browser is loading u only if we have $C(\ell) \cap C(\ell') \neq \emptyset$;
4) *session integrity*: if a URL u has label ℓ, an attacker ℓ' can force the browser into sending requests to u only if we have $I(\ell) \cap I(\ell') \neq \emptyset$.

Formally, we define a *URL labelling* as a function $\Gamma : \mathcal{U} \to \mathcal{L}$, assigning labels to URLs. If $\Gamma(u) = \ell$ for some label ℓ, let $\Gamma_C(u)$ stand for $C(\ell)$ and $\Gamma_I(u)$ stand for $I(\ell)$. We propose a technique to automatically generate a *canonic* transfer function from a labelling Γ: this function enforces the session confidentiality and integrity properties formalized by Γ, while ensuring that cookies are accessed correctly according to their security label. The canonic transfer function operates on the set of tags $\mathit{Tags} \triangleq \mathcal{L} \cup \mathcal{U}$ including labels and URLs, and assumes that the tagging function for URLs *tag* is the identity on $\mathcal{U}$.

The definition of the canonic transfer function is formalized by using judgements of the following format:

$$\Gamma, f \rhd \mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) \rightsquigarrow \vec{\ell},$$

where Γ, f, $\mathsf{event\_type}$ and τ_1, τ_2 are known, while we compute the labels $\vec{\ell}$ to be assigned to the newly created or updated browser elements when processing an event of type $\mathsf{event\_type}$. Here, $f : \mathcal{L} \to \mathcal{L}$ is a *script labelling*, providing a mapping from labels of network connections to labels of scripts downloaded via these connections. Remarkably, while Γ is used to specify different security policies for different URLs and should be provided by web developers, f is just a parameter used to tweak the generation of the canonic transfer function for different use cases. It is useful to have f in the formalism for additional generality, in particular to support the examples in the next section, but in practice (and in our implementation) a good candidate for f is simply the identity function on $\mathcal{L}$. The non-interference results we present hold for any choice of f, as long as it satisfies the following well-formation condition (implicitly assumed from now on).

Definition 8 (Well-formed Script Labelling)**.** *A script labelling* f *is* well-formed *if and only if, for all labels* $\ell \in \mathcal{L}$*, we have* $C(f(\ell)) \subseteq C(\ell)$ *and* $I(\ell) \subseteq I(f(\ell))$*.*

The judgements defining the canonic transfer function are shown in Table II, using inference rules which should be read as follows: boxed premises amount to checks on Γ, τ_1, τ_2, determining the domain of the transfer function, while premises not included in boxes define the value of the new labels $\vec{\ell}$. If any of the boxed premises fails, the transfer function is undefined and the browser does not process the event. Observe that that the necessary entries of the transfer function

can be generated "on the fly" upon event processing in an implementation of our theory.

We briefly comment on the rules in Table II as follows. In (G-LOAD), assuming that we load the URL u, we check *evt_label*$(u) \subseteq \Gamma_C(u)$, since a request is sent to u and this request is visible to any network attacker or to any web attacker controlling u. We use $\Gamma_C(u)$ as the confidentiality label of the new network connection to ensure that the presence of that connection in the browser may only be visible to an attacker ℓ such that $\Gamma_C(u) \cap C(\ell) \neq \emptyset$. We use *msg_label*$(u)$ as the integrity label of the new network connection, since an attacker controlling u may be able to compromise the integrity of any response received over that connection. Finally, when accessing the cookie jar to retrieve the cookies to be sent in the request to u, we set *msg_label*(u) as the confidentiality component of the label ℓ_{co} passed to the cookie jar, which ensures that only cookies intended to be disclosed to u will be retrieved. We use $\top_s$ as integrity label for retrieving cookies, so that we get cookies irrespective of their integrity label.

(G-DOCRESP) and (G-XHRRESP) propagate the label ℓ_n of the network connection to the cookie jar when setting new cookies included in a network response received over that connection. In terms of integrity, this implies that a network connection can only set cookies with lower integrity than itself. More subtly, in terms of confidentiality, this also implies that a network connection can only set cookies with higher confidentiality than itself: this is needed to ensure that the attacker cannot detect the occurrence of private network responses from the value (or the existence) of public cookies set in those responses. The rules assign the label $f(\ell_n)$ to the new scripts running after response processing; here, the well-formation of f is crucial to ensure that the confidentiality and integrity restrictions of the script are as least as strong as those of the network connection where they have been downloaded.

In (G-DOCREDIR) and (G-XHRREDIR), when redirecting to a URL u, we check *evt_label*$(u) \subseteq C(\ell_n)$, since a network request is sent to u upon redirection and it may reveal the existence of the network connection. We also have to check $I(\ell_n) \subseteq \Gamma_I(u)$ to ensure that no low integrity connection sends a request to a high integrity URL. We preserve the confidentiality label of the existing network connection, so that the existence of the connection cannot be revealed even after the redirection. Instead, we update the integrity label of the connection to the original integrity label of the connection extended with *msg_label*(u): this formalizes the intuition that the integrity of a network connection gets downgraded through cross-origin redirects. The label used for writing cookies is ℓ_n, just as in (G-DOCRESP), while the label used for fetching cookies is (*msg_label*$(u), \top_s$), just as in (G-LOAD).

(G-GET) ensures that no high confidentiality cookie is read by a low confidentiality context and that no low integrity cookie is read by a high integrity context. (G-SET) is the writing counterpart of (G-GET). Finally, (G-SEND) is similar to (G-XHRREDIR), with the role of the incoming network connection taken by a running script (and no cookie set).

C. Reactive Non-Interference

Having defined a canonic transfer function, we now analyze which non-interference properties are supported by it. Let $\mathsf{FF}^\tau(\Gamma, f)$ be the instantiation of FF^τ with the transfer function derived from Γ and f using the judgements in Table II.

We first discuss confidentiality. The next definition of erasure removes from input events any cookie which must not be visible to the attacker according to its label. By making similar input events that are identical after such erasure, reactive non-interference ensures that the value (and even the presence) of the confidential cookies has no visible import to the attacker.

Definition 9 (Confidentiality Erasure)**.** *Given a set of cookies CK, let ck-erase$^C_\ell(CK)$ be defined as:*

$$\{\mathsf{ck}(s, s')^{\ell'} \in CK \mid C(\ell') \cap C(\ell) \neq \emptyset\}.$$

We then define erase$^C_\ell : \mathcal{I} \to \mathcal{I}$ by applying ck-erase$^C_\ell$ to each CK syntactically occurring in an input event.

The confidentiality theorem combines cookie confidentiality with session confidentiality, i.e., the occurrence of a $\mathsf{load}(u)$ event which must not be visible to the attacker according to the label of u has indeed no visible import on the outputs produced by the browser.

Theorem 1 (Confidentiality)**.** *Let $\pi_C = \langle rel_\ell, \sim_\ell \rangle$ be the confidentiality policy such that:*

1) $\forall i : \neg rel_\ell(i) \triangleq i = \mathsf{load}(u) \wedge \Gamma_C(u) \cap C(\ell) = \emptyset$;
2) $\forall i, i' : i \sim_\ell i' \Leftrightarrow erase^C_\ell(i) = erase^C_\ell(i')$.

Then, $\mathsf{FF}^\tau(\Gamma, f)$ is non-interferent under π_C.

We now focus on integrity. The next definition of erasure removes from output events any cookie which can be set by the attacker according to its label. By making similar output events that are identical after such erasure, reactive non-interference ensures that only low-integrity cookies can be affected by a manipulation of the input stream performed by the attacker.

Definition 10 (Integrity Erasure)**.** *Given a set of cookies CK, let ck-erase$^I_\ell(CK)$ be defined as:*

$$\{\mathsf{ck}(s, s')^{\ell'} \in CK \mid I(\ell') \cap I(\ell) = \emptyset\}.$$

We then define erase$^I_\ell : \mathcal{O} \to \mathcal{O}$ by applying ck-erase$^I_\ell$ to each CK syntactically occurring in an output event.

The integrity theorem combines cookie integrity with session integrity, i.e., the attacker can force the browser into sending network requests to u only if the label of u has a low integrity component.

Theorem 2 (Integrity)**.** *Let $\pi_I = \langle rel_\ell, \sim_\ell \rangle$ be the integrity policy such that:*

1) $\forall o : rel_\ell(o) \triangleq o = \mathsf{net\_req}(u : CK) \wedge \Gamma_I(u) \cap I(\ell) = \emptyset$;
2) $\forall o, o' : o \sim_\ell o' \Leftrightarrow erase^I_\ell(o) = erase^I_\ell(o')$.

Then, $\mathsf{FF}^\tau(\Gamma, f)$ is non-interferent under π_I.

To prove these two theorems, we define a set of syntactic constraints over the structure of the transfer function and we show that the non-interference statements hold for any transfer function satisfying the constraints. We then prove that

TABLE II Generation of a canonic transfer function from Γ

(G-LOAD)
$$\frac{\boxed{\mathit{evt\_label}(u) \subseteq \Gamma_C(u)} \quad \ell_n = (\Gamma_C(u), \mathit{msg\_label}(u)) \quad \ell_{co} = (\mathit{msg\_label}(u), \top_s)}{\Gamma, f \rhd \mathit{transfer}(\mathsf{load}, u, -) \rightsquigarrow (\ell_n, -, \ell_{co}, -)}$$

(G-DOCRESP)
$$\Gamma, f \rhd \mathit{transfer}(\mathsf{doc\_resp}, \ell_n, -) \rightsquigarrow (-, \ell_n, -, f(\ell_n))$$

(G-DOCREDIR)
$$\frac{\boxed{\mathit{evt\_label}(u) \subseteq C(\ell_n)} \quad \boxed{I(\ell_n) \subseteq \Gamma_I(u)} \quad \ell_m = (C(\ell_n), I(\ell_n) \cup \mathit{msg\_label}(u)) \quad \ell_{co} = (\mathit{msg\_label}(u), \top_s)}{\Gamma, f \rhd \mathit{transfer}(\mathsf{doc\_redir}, \ell_n, u) \rightsquigarrow (\ell_m, \ell_n, \ell_{co}, -)}$$

(G-XHRRESP)
$$\Gamma, f \rhd \mathit{transfer}(\mathsf{xhr\_resp}, \ell_n, -) \rightsquigarrow (-, \ell_n, -, f(\ell_n))$$

(G-XHRREDIR)
$$\frac{\boxed{\mathit{evt\_label}(u) \subseteq C(\ell_n)} \quad \boxed{I(\ell_n) \subseteq \Gamma_I(u)} \quad \ell_m = (C(\ell_n), I(\ell_n) \cup \mathit{msg\_label}(u)) \quad \ell_{co} = (\mathit{msg\_label}(u), \top_s)}{\Gamma, f \rhd \mathit{transfer}(\mathsf{xhr\_redir}, \ell_n, u) \rightsquigarrow (\ell_m, \ell_n, \ell_{co}, -)}$$

(G-GET)
$$\frac{\boxed{C(\ell_r) \subseteq C(\ell_t)} \quad \boxed{I(\ell_t) \subseteq I(\ell_r)}}{\Gamma, f \rhd \mathit{transfer}(\mathsf{get}, \ell_r, \ell_t) \rightsquigarrow (-, -, -, -)}$$

(G-SET)
$$\frac{\boxed{C(\ell_t) \subseteq C(\ell_w)} \quad \boxed{I(\ell_w) \subseteq I(\ell_t)}}{\Gamma, f \rhd \mathit{transfer}(\mathsf{set}, \ell_w, \ell_t) \rightsquigarrow (-, -, -, -)}$$

(G-SEND)
$$\frac{\boxed{\mathit{evt\_label}(u) \subseteq C(\ell_s)} \quad \boxed{I(\ell_s) \subseteq \Gamma_I(u)} \quad \ell_n = (C(\ell_s), I(\ell_s) \cup \mathit{msg\_label}(u)) \quad \ell_{co} = (\mathit{msg\_label}(u), \top_s)}{\Gamma, f \rhd \mathit{transfer}(\mathsf{send}, \ell_s, u) \rightsquigarrow (\ell_n, -, \ell_{co}, -)}$$

the canonic transfer function we defined always satisfies the constraints. Note that this approach allows one to syntactically prove non-interference for transfer functions different from the canonic one, which is a useful and interesting result by itself. For space reasons, we refer the interested reader to Appendix B for the presentation of the full set of constraints.

Another interesting property of the canonic transfer function is that it ensures *compatibility* for websites not implementing the security mechanisms proposed in this paper. Intuitively, it is possible to identify a "weak" URL labelling which does not improve security with respect to standard web browsers, but ensures that no runtime security check performed by the transfer function will ever stop a website from working as originally intended. Formally, we extend the set of output events of FF^τ with a new event $\star$, called *failure*. We then define a variant of FF^τ which is parametric with respect to a URL labelling Γ and explicitly models failures due to the security enforcement performed by the canonic transfer function derived from Γ. This is done by including the event $\star$ in the output stream generated by the reactive system whenever the transfer function is undefined. An excerpt of the failure semantics is presented in Appendix C.

Let $\Gamma_\top$ be the URL labelling assigning the $\top$ label to each URL, let id be the identity function on labels and let $\mathsf{FF}^\tau_\star(\Gamma_\top, id)$ be the failure-aware variant of FF^τ implementing the canonic transfer function derived from $\Gamma_\top$ and id. We can state the following compatibility theorem.

Theorem 3 (Compatibility)**.** *Let C_0 be the initial state of $\mathsf{FF}^\tau_\star(\Gamma_\top, id)$ and assume that the function $\kappa : \mathcal{D} \times \mathcal{S} \to \mathit{Tags}$ assigns the top label $\top$ to all the elements of its domain. If $C_0(I) \Downarrow O$, then $\star$ does not occur in O.*

VI. CASE STUDIES

A. Cookie Protection Against Web Attackers

The HttpOnly attribute has been proposed as an in-depth defense mechanism for authentication cookies against web attackers [3]. If a cookie is marked as HttpOnly, the browser forbids any access to it by JavaScript, thus preventing its theft through a successful XSS exploitation. The HttpOnly attribute also provides some integrity guarantees, since JavaScript cannot set or overwrite HttpOnly cookies.

Intuitively, a first attempt at representing HttpOnly cookies in our model can be done by giving cookies set by the domain d the label $\ell_c = (\{\mathsf{http}(d), \mathsf{https}(d)\}, \{\mathsf{http}(d), \mathsf{https}(d)\})$, and by ensuring that scripts are assigned the top label $\top$. The label ℓ_c allows the browser to send and set these cookies over both HTTP and HTTPS connections to d. The label $\top$ assigned to scripts, instead, ensures that JavaScript cannot read or write these cookies, as enforced by rules (G-GET) and (G-SET).

As it turns out, however, this labelling forces the implementation of stricter security checks than those performed by standard web browsers on HttpOnly cookies. This is not a limitation of our model, but rather a consequence of the fact that scripts are actually able to compromise the integrity of HttpOnly cookies in current web browsers. Indeed, even if an attacker-controlled script cannot directly set an HttpOnly cookie by accessing the `document.cookie` property, it can still force HttpOnly cookies into the browser by exploiting network communication. For instance, assume that a trusted website `a.com` uses HttpOnly cookies for authentication purposes: a malicious script could run a login CSRF attack by

submitting the attacker's credentials to `a.com`, thus effectively forcing fresh HttpOnly cookies into the user's browser.

To prevent this class of attacks, the canonic transfer function enforces two further invariants: (1) by rule (G-SEND), all the network connections which are opened by a script are tagged with label $\top$, which enforces that no cookie with integrity label $\{\mathsf{http}(d), \mathsf{https}(d)\}$ can be set over these connections; and (2) by rules (G-DOCREDIR) and (G-XHRREDIR), when a redirect is performed, the integrity label of the network connection receiving the redirect is downgraded to the union of the original integrity label and the message label of the redirect URL. Hence, if a cross-domain redirect is performed, no cookie with integrity label $\{\mathsf{http}(d), \mathsf{https}(d)\}$ can be set over the network connection. This ensures that web attackers cannot exploit malicious scripts or redirects to set cookies with label ℓ_c in the browser, unless they control the domain d.

In the end, standard HttpOnly cookies cannot be accurately modelled in our framework, since the integrity guarantees they provide cannot be expressed by a non-interference policy. Indeed, HttpOnly cookies cannot be set by a script using the `document.cookie` property, but scripts can still set them by exploiting network communication, so it is not clear which label should be assigned to scripts to have non-interference. As we discussed, this asymmetry leaves room for attacks.

Clearly, one can represent in our framework cookies which cannot be read by scripts, but can be set by them, by replacing the cookie label ℓ_c with $\ell'_c = (\{\mathsf{http}(d), \mathsf{https}(d)\}, \top_s)$. These cookies cannot be read by scripts running with the $\top$ label, but they can be liberally set by them. Another plausible design choice would be changing the label given to scripts to let them access cookies labelled ℓ'_c, at the cost of limiting their cross-origin communication. For instance, by giving scripts the label of the connection where they have been downloaded, scripts from the domain d would be allowed to read cookies labelled ℓ'_c, but any cross-domain communication from these scripts will be forbidden by rule (G-SEND) to prevent cookie leakage. This may be a better solution for web applications like e-banking services, which may need to access session state at the client side, but do not interact with untrusted third-parties.

B. Protection Against Gadget Attackers

The gadget attacker has been first introduced in [5] as a realistic threat model for mashup security. A gadget attacker is just a web attacker with an additional capability: a trusted website deliberately embeds a gadget (script) chosen by the attacker as part of its standard functionalities. The embedded gadget may be useful, e.g., for advertisement purposes or for the computation of site-wide popularity metrics. It is well-known that this kind of operation is dangerous on the Web, since the embedded script may be entitled to run in the same origin of the embedding page [24]. For instance, the embedded script may be able to read the authentication cookies of the embedding page. This is largely accepted, however, as long as the author of the embedding page trusts the gadget. But what if the gadget is compromised by the attacker?

Consider a web page hosted at the HTTPS URL u on domain d and loading a gadget from the HTTPS URL u' on domain d'. We can define a labelling Γ such that $\Gamma_C(u) = \{\mathsf{https}(d), \mathsf{https}(d')\}$, which would allow the web page at u to only communicate with HTTPS URLs hosted at d and d' by rule (G-SEND). This may be fine, for instance, if the gadget loaded from u' only computes some local statistics shown in the web page at u. Pick now the following input stream:

$$\begin{aligned} I \;=\; & [\mathsf{load}(u), \\ & \mathsf{doc\_resp}_n(u : \{\mathsf{ck}(k, v)^{\ell}\}, \mathsf{xhr}(u', \lambda x.x\,\mathsf{unit})), \\ & \mathsf{xhr\_resp}_m(u' : \emptyset, \lambda y.\mathsf{let}\ z = \mathsf{get\text{-}ck}(k)\ \mathsf{in}\ \mathit{leak}(z))] \end{aligned}$$

The input stream I models a scenario where the normally harmless gadget on u' has been somehow compromised by the attacker, so that it will read the value of the cookie k set by the response from u and leak it to the attacker's website, which we assume to be hosted outside the domains d and d'. This attack is prevented by the labelling above, since the XHR request leaking the cookie value is stopped by rule (G-SEND), given that this request would still originate from u.

C. Strengthening PayPal

In 2014 a severe CSRF vulnerability on the online payment system PayPal was disclosed, despite existing server-side protection mechanisms [1]. PayPal employs authentication tokens in order to prevent CSRF attacks, but these tokens could be used multiple times for the same user and, through another vulnerability, it was possible for an attacker to obtain such a valid token for any user. The combination of these two flaws could be used to mount arbitrary CSRF attacks against any PayPal user, e.g., to authorize payments on the user's behalf.

Figure 1 represents a typical payment scenario [25] for a user that has already logged into her PayPal account. The protocol starts with the user clicking on the "buy now" button in an online shop. After being redirected to PayPal, she confirms the payment and the article is successfully purchased. One important detail for the present discussion is that the initial request to PayPal (step 2) is explicitly triggered by the user (step 1) in this scenario. Our goal is to enforce a policy that prevents CSRF attacks against PayPal, while still allowing benign payments. This can be done by setting $\Gamma(u) = (\top_s, \{\mathsf{https}(\texttt{paypal.com})\})$ for all URLs u of PayPal, while letting $\Gamma(u') = \top$ for all URLs u' of the online shop (as we are only interested in protecting PayPal here).

We first explain why this policy does not block benign payments via PayPal, like in Figure 1: since we have an explicit user action that triggers the initial request to PayPal, we assimilate steps 1-2 of the protocol to the processing of a load event in our formal model. Since $\Gamma_C(u) = \top_s$ for all URLs u of PayPal, the request at step 2 is successfully sent according to rule (G-LOAD). Steps 3-12 of the protocol are always in the domain of PayPal and thus the label used for scripts and network connection is always $(\top_s, \{\mathsf{https}(\texttt{paypal.com})\})$. This allows the browser to perform arbitrary redirects and XHR request to URLs on PayPal, hence all these steps succeed. Finally, since we have $\Gamma_C(u) = \top_s$ for all URLs u at PayPal and we have $\Gamma_I(u') = \top_s$ for all URLs at the shop domain, also steps 13-15 can be performed successfully, since the cross-origin redirect is permitted by rule (G-DOCREDIR).

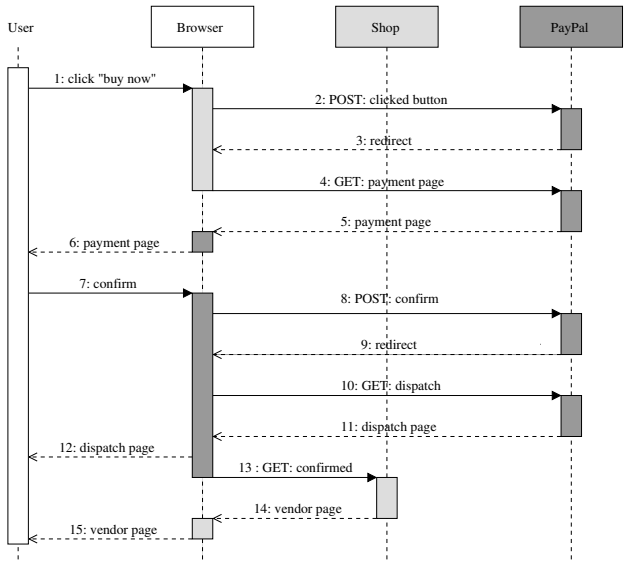

Fig. 1. A typical payment scenario on PayPal

If we now consider a CSRF attack, then we do not have a message that is triggered directly by the user at step 1. This means that we do not have a load event to process, but rather a xhr_req, doc_redir or xhr_redir event coming from a domain different from `paypal.com`. By the definition of the canonic transfer function, we then always have to show either $I(\ell_n) \subseteq \Gamma_I(u)$ or $I(\ell_s) \subseteq \Gamma_I(u)$ in these cases, where u is the URL loaded at step 1. One can then observe that we always have $I(\ell_n) \not\subseteq \{\mathsf{https}(\texttt{paypal.com})\}$ and $I(\ell_s) \not\subseteq \{\mathsf{https}(\texttt{paypal.com})\}$, hence the integrity checks fail and the message is not sent. Thus, our technique effectively prevents the aforementioned CRSF attack against PayPal, even if vulnerabilities are not fixed at the server side.

D. Additional Examples

We also developed two more examples to show our framework at work: cookie protection against network attackers improving on Secure cookies [3] and protection against CSRF in the spirit of Allowed Referrer Lists [13]. For space reasons, these examples are only included in Appendix D.

VII. Implementation

We developed a proof-of-concept implementation of our proposal as a Google Chrome extension, Michrome, which we make available online [2].

A. Michrome Implementation

Michrome changes the behaviour of Google Chrome by mimicking the operational semantics of FF^τ, assuming the deployment of the canonic transfer function in Section V-B. The prototype leverages the standard Google Chrome extension APIs, which allow for a rather direct implementation of the semantics. We discuss below the main differences with respect to the formal model and some important details.

1) User Inputs: FF^τ uniformly treats user inputs as load events. In practice, however, users have different ways to interact with their web browser, most notably by typing in the address bar and by clicking buttons or links. Assimilating all user inputs to load events in Michrome would be a poor design choice, since many of these inputs, e.g., button clicks, can be triggered by malicious JavaScript code, but load events have high integrity in our model. Unfortunately, the Google Chrome extension APIs do not allow one to discriminate between user clicks and clicks performed by JavaScript; similarly, they do not provide any way to distinguish between the user writing in the address bar and a navigation attempt by JavaScript.

Our choice is then to only endorse the first request which is fired from an empty tab and to deem it as the result of a load event, since the only way to trigger a network request from an existing empty tab is by typing in its address bar. All the other network requests are assimilated to less trusted xhr_req events and hence subject to stricter security checks. This policy can be relaxed by defining in Michrome a white-list of trusted entry points, i.e., URLs which are known to be controlled by trusted companies and have a very high assurance of being protected against CSRF attacks: a similar approach has already been advocated in App Isolation [12]. Relaxing the standard behaviour of Michrome is occasionally helpful in practice, for instance to support the PayPal case study (see below).

2) Tagging Scripts: in normal web browsing, many scripts run in the same page (and hence in the same origin) at the same time. The Google Chrome extension APIs do not allow one to detect which script is performing a given operation when more than one script is included in the same page, so Michrome cannot assign labels to individual scripts (unlike the FF^τ model). This issue is solved by giving a label to the entire tab displaying the page rather than to individual scripts. Intuitively, this label represents an upper bound for each label which would be assigned to a script running in the tab.

When a remote content is included from a URL u, the label of the including tab is downgraded and joined with $\Gamma(u)$. There is only one simple exception to this rule: if the included content is passive, i.e., if it is an image, the integrity label of the tab does not get downgraded. This prevents label creeping for integrity and simplifies the specification of information flow policies for web developers.

3) Policy Granularity: In the current prototype, security labels are assigned to domain names rather than to URLs. This choice is mainly dictated by the practical need of testing our extension on existing websites: having a more coarse-grained security enforcement simplifies the process of writing information flow policies for websites we do not know and control. There is no real mismatch from the formal model here: we just implicitly assume that $\Gamma(u) = \Gamma(u')$ for all u, u' such that $\mathit{host}(u) = \mathit{host}(u')$.

4) Default Behaviour: Formally, the labelling Γ is defined as a (total) function from URLs to labels. In practice, however, one cannot assign a label to all the URLs in the Web. Our choice is to implicitly assume the $\top$ label for all the URLs without an explicit entry in Γ. This solution is suggested by the choice of preserving compatibility with existing websites:

since the $\top$ label does not constrain cross-origin communication, the browser behaviour is unchanged when interacting with URLs not included in the labelling.

5) Cookies: At the time of writing, Michrome does not have full support for cookie labels. We plan to implement support for arbitrary labels in the next future, but the current prototype always assumes that a cookie received from a URL u has confidentiality label $\Gamma_C(u)$ and integrity label $\top_s$. This choice is mainly done for the sake of simplicity: by implicitly inferring cookie labels from Γ, we reduce the amount of information which we must specify for the websites we test. The confidentiality label $\Gamma_C(u)$ is justified by the fact that we assume that all URLs on the same domain have the same Γ, hence all the cookies set by them cannot be communicated outside $\Gamma_C(u)$[1]. The integrity label $\top_s$, instead, is motivated by backward compatibility: since standard cookies do not provide good integrity guarantees, we do not try to enforce additional protection in order to avoid breaking websites.

B. Securing the University Website

We performed a first test of Michrome by securing a university website, call it U. Since this website does not include many third-party contents and does not expect to process cross-domain requests, we first assigned U the label: $(\{\mathsf{http}(U), \mathsf{https}(U)\}, \{\mathsf{http}(U), \mathsf{https}(U)\})$. This label states that any session established with U should only be visible to U itself and that only local web pages are allowed to send requests to U. We then realized that this labelling modifies the browser behaviour when navigating U, since the homepage of U silently includes scripts from Google Analytics (*GA*) over HTTP and the extension blocks any request for these scripts, since *GA* should not be aware of the loading of U.

We tried to make Google Analytics work again by adding $\mathsf{http}(GA)$ to the confidentiality label of U. This indeed allowed the browser to send the request for the analytic scripts, but it also prevented the correct rendering and navigation of U later on. The reason is that, when a script from *GA* is included into a page on U, the integrity label of the tab displaying the page is downgraded to include $\mathsf{https}(GA)$. Since the integrity label of U does not mention *GA*, further requests to U from the page are dropped by Michrome: indeed, these requests may be fired by a malicious script mounting a CSRF attack. To recover functionality, we thus had to relax the integrity label of U to also include $\mathsf{https}(GA)$.

Another small change we had to perform to seamlessly navigate U was to extend its confidentiality label to include the sub-domain where the private area of the university is hosted. We also realised the need to include Google (G) in the integrity label of U, otherwise Michrome would prevent the browser from accessing U from the Google search page. Perhaps surprisingly, though Google is entirely deployed over HTTPS, extending the integrity label of U with just $\mathsf{https}(G)$ does not suffice to fully preserve functionality. The reason is that, just like most users, we often omit the protocol and just type `www.google.com` in the address bar to access Google: the browser then tries HTTP by default and then gets automatically redirected to HTTPS by Google. Hence, the integrity label of the tab after the redirect becomes $\{\mathsf{http}(G), \mathsf{https}(G)\}$, which is not good enough to access U. This problem can be solved by using HSTS [17] and preventing any communication attempt to Google over HTTP.

An alternative, simpler solution to this problem is listing the home page of U as a trusted entry point in Michrome, so as to avoid listing all the most popular search engines in the integrity label of U. This is a safe choice in practice, since the homepage of U, like most homepages, is static and does not expect any parameter or untrusted input to sanitize. All in all, we found it pretty easy to come up with an accurate security policy for the website and we think that most web developers should find this process quite intuitive to carry out.

C. Testing on PayPal

We also tested Michrome by placing an order on a well-known digital distribution platform, call it D, and by performing the payment using PayPal. We first built an entry for D in the URL labelling, ensuring that both the confidentiality and the integrity components of its label only included D and PayPal. We then set the confidentiality label of PayPal to $\top_s$ and its integrity label to PayPal itself (over HTTPS), thus reconstructing the scenario in the Section VI-C. The payment process worked seamlessly, confirming the result we expected from the formal model.

D. Compatibility and Perceived Performances

Besides the experiments detailed above, we also wrote information flow policies for a small set of national websites and we left Michrome activated in our web browsers while routinely browsing the Web for a few days. We never encountered any visible compatibility issue, even when interacting with websites without an explicit label in the URL labelling, which confirms that the $\top$ label given to them is a sensible default. Clearly, we occasionally broke websites when trying to come up with a correct label to assign to them, but this operation only needs to be done once per website (and only if additional protection is desired for that website). We envision a collaborative effort by security experts and web developers to write down policies for the major security-relevant websites, as it already happens for HTTPS Everywhere [15]. We did not observe any perceivable performance degradation in any of the visited websites, which we do not find surprising, given that the security enforcement ultimately boils down to a few (light-weight) checks on labels.

VIII. Related Work

Browser-enforced security policies have already been proposed in the past, following two main lines of research. The first research line proposed *purely client-side* defenses like ZAN [28], SessionShield [22], CookiExt [9], [10], CsFire [25] and SessInt [9], which automatically mitigate web applications vulnerabilities by changing the browser behaviour. We improve over these works by giving web developers a tool to

[1]This is only true if no cookie sharing between sub-domains is possible. Indeed, the current prototype does not protect domain cookies [3], but we plan to include support for them in future releases.

express their own browser-enforced security policies. Involving web developers in the security process is crucial for the usability and the large-scale adoption of a defensive solution, since purely client-side defenses like the ones we mentioned must implement heuristics to "guess" when their security policy should be applied. These heuristics are bound to (at least occasionally) fail: for instance, CookiExt sometimes breaks the Facebook chat [10], while CsFire prevents certain uses of OpenId [13]. The second research line, instead, focused on *hybrid* solutions similar to our approach, where the browser enforces a security policy specified by the server [18], [21], [26], [29], [13]. These proposals, however, target very specific attacks like XSS [18], [21], [26] or CSRF [13] and have not been proved correct. Conversely, in this paper we formalize a rather general micro-policy framework for web browsers and we prove it is expressive enough to support a broad class of useful information flow policies, subsuming existing low-level security mechanisms for web sessions. We think that other useful security properties beyond non-interference can be enforced using micro-policies in the browser: we leave this study for future work.

The present paper was also inspired by previous work on information flow control for web browsers. FlowFox [16] was the first web browser enforcing a sound and precise information flow control on JavaScript by using secure multi-execution. There are many important differences between that approach and the one proposed in this paper. First, FlowFox exclusively prevents attacks posed by malicious scripts, while our proposal covers more common web threats, including malicious HTTP(S) redirects and network attacks. Second, FlowFox does not address integrity threats, though an extension explicitly aimed at thwarting CSRF attacks via scripts has been proposed [20]. Third, FlowFox requires profound changes to the JavaScript engine and has a quite significant impact on browsing performances, while we advocate a more lightweight approach deployed as a browser extension. FlowFox, however, allows the specification of fine-grained information flow policies on JavaScript which are beyond the scope of this work.

More recently, a research paper reported on the extension of Chromium with support for information flow control based on a lightweight, coarse-grained form of taint tracking [6]. This proposal complements previous work on information flow control for JavaScript by focusing on the entire browser and embracing a wider range of web threats. It might be interesting to explore if the security mechanisms we advocate in this paper can be implemented using the security labels discussed in [6]. The scope of the two works, however, is different: we focus on web session security, while [6] targets intra-browser information flow policies. We model a smaller fragment of the browser, but our proposal requires way less changes to existing web browsers, as testified by a prototype implementation of our framework as a Google Chrome extension. Indeed, our approach deliberately targets a good balance between strong web session security guarantees and minimal browser changes.

Our proposal also shares similar design goals with coarse-grained information flow control frameworks for JavaScript like BFlow [31] and COWL [27]. These frameworks divide scripts in compartments and assign security labels to the latter, to then constrain communication across compartments based on label checks. An important difference with respect to these works is that the scope of the present paper is not limited to JavaScript. Moreover, we carry out our technical development in a formal model and prove security with respect to this model, while neither BFlow nor COWL have been formalized. However, both BFlow and COWL support the enforcement of general information flow policies on JavaScript code, which is beyond the scope of the present work.

Finally, we observe that our label-based policies for confidentiality and integrity are reminiscent of the Same Origin Mutual Approval (SOMA) proposal [23]. SOMA extends the browser with stricter access control checks on content inclusion: both the site operator of the including page and the third party content provider must approve a content inclusion before any communication is allowed by the browser. SOMA is shown to be effective in particular against CSRF attacks and malicious data exfiltration through XSS attacks, which are threats considered also in our work. There are two relevant differences, however, which make our proposal strictly more expressive than SOMA. First and most importantly, SOMA defines an access control mechanism and not an information flow framework: all the security checks performed by SOMA only depend on the including page and the embedded contents, and there is no way to allow or deny a content inclusion based on whether, e.g., the including page has been retrieved by a redirect from the attacker website. Second, SOMA only focuses on network communication and does not support security policies for cookies.

IX. Conclusion

This work explores the usage of micro-policies for the specification and enforcement of confidentiality and integrity properties of web sessions. Micro-policies are specified in terms of tags (here, information flow labels) and a transfer function, which is responsible for monitoring security-relevant operations based on these tags. We modelled the browser as a reactive system and information flow security for web sessions as a non-interference property. We designed a synthesis technique for the transfer function, which allows the end user to specify the expected security policies as simple confidentiality and integrity labels. We demonstrated how our framework uniformly captures a broad spectrum of security policies (e.g., cookie protection, CSRF prevention, and gadget security), improving over existing ad-hoc solutions in terms of soundness and flexibility. Despite the flexibility of micro-policies, we managed to implement a prototype of our proposal as a simple and efficient extension for Google Chrome.

As a future work, we plan to extend our formal model by considering additional browser and webpage components, striving for a good balance between flexibility and ease of deployment. We also plan to formalize our development in a theorem prover in order to provide machine-checked security proofs. Furthermore, we plan to design micro-policies tailored to other popular web applications, such as single sign-on

protocols, conducting a systematic security analysis. While performing experiments we realized that Michrome naturally acts as a *learning tool* that collects the integrity level of any web resource that is accessed by a web application. More specifically, when we do not assign security labels to a website, any access is allowed and the integrity level of the browser tab is populated by the security labels of accessed URLs. This information is useful to have an immediate idea of the "trusted computing base" of the web application and, in many cases, to discover potential vulnerabilities such as importing scripts via HTTP. We will complement this learning feature with information about violation of the transfer function, so to automatically derive confidentiality and integrity labels for a whole web application.

REFERENCES

[1] Y. Ali, "Hacking paypal accounts with one click (patched)," 2014, available at http://yasserali.com/hacking-paypal-accounts-with-one-click.

[2] Anonymus, "Micro-policies for web session security," 2016, available at https://sites.google.com/site/micropolwebsese.

[3] A. Barth, "Http state management mechanism," 2011, available at https://tools.ietf.org/html/rfc6265.

[4] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, 2008, pp. 75–88.

[5] ——, "Securing frame communication in browsers," in *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, 2008, pp. 17–30.

[6] L. Bauer, S. Cai, L. Jia, T. Passaro, M. Stroucken, and Y. Tian, "Run-time monitoring and formal analysis of information flows in chromium," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[7] A. Bohannon and B. C. Pierce, "Featherweight firefox: Formalizing the core of a web browser," in *USENIX Conference on Web Application Development, WebApps'10, Boston, Massachusetts, USA, June 23-24, 2010*, 2010.

[8] A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic, "Reactive noninterference," in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, 2009, pp. 79–90.

[9] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "Automatic and robust client-side protection for cookie-based sessions," in *Engineering Secure Software and Systems - 6th International Symposium, ESSoS 2014, Munich, Germany, February 26-28, 2014, Proceedings*, 2014, pp. 161–178.

[10] ——, "Cookiext: Patching the browser against session hijacking attacks," *Journal of Computer Security*, vol. 23, no. 4, pp. 509–537, 2015.

[11] M. Bugliesi, S. Calzavara, R. Focardi, W. Khan, and M. Tempesta, "Provably sound browser-based enforcement of web session integrity," in *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, 2014, pp. 366–380.

[12] E. Y. Chen, J. Bau, C. Reis, A. Barth, and C. Jackson, "App isolation: get the security of multiple browsers with just one," in *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, 2011, pp. 227–238.

[13] A. Czeskis, A. Moshchuk, T. Kohno, and H. J. Wang, "Lightweight server support for browser-based CSRF protection," in *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, 2013, pp. 273–284.

[14] A. A. de Amorim, M. Dénès, N. Giannarakis, C. Hritcu, B. C. Pierce, A. Spector-Zabusky, and A. Tolmach, "Micro-policies: Formally verified, tag-based security monitors," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, 2015, pp. 813–830.

[15] Electronic Frontier Foundation, "HTTPS Everywhere," 2015, available at https://www.eff.org/https-everywhere.

[16] W. D. Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "Flowfox: a web browser with flexible and precise information flow control," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, 2012, pp. 748–759.

[17] J. Hodges, C. Jackson, and A. Barth, "Http strict transport security (hsts)," 2012, available at https://tools.ietf.org/html/rfc6797.

[18] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, 2007, pp. 601–610.

[19] M. Johns, B. Braun, M. Schrank, and J. Posegga, "Reliable protection against session fixation attacks," in *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, 2011, pp. 1531–1537.

[20] W. Khan, S. Calzavara, M. Bugliesi, W. D. Groef, and F. Piessens, "Client side web session integrity as a non-interference property," in *Information Systems Security - 10th International Conference, ICISS 2014, Hyderabad, India, December 16-20, 2014, Proceedings*, 2014, pp. 89–108.

[21] M. T. Louw and V. N. Venkatakrishnan, "Blueprint: Robust prevention of cross-site scripting attacks for existing browsers," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, 2009, pp. 331–346.

[22] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen, "Sessionshield: Lightweight protection against session hijacking," in *Engineering Secure Software and Systems - Third International Symposium, ESSoS 2011, Madrid, Spain, February 9-10, 2011. Proceedings*, 2011, pp. 87–100.

[23] T. Oda, G. Wurster, P. C. van Oorschot, and A. Somayaji, "SOMA: mutual approval for included content in web pages," in *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, 2008, pp. 89–98.

[24] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, 2012, pp. 155–168.

[25] P. D. Ryck, L. Desmet, W. Joosen, and F. Piessens, "Automatic and precise client-side protection against CSRF attacks," in *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, 2011, pp. 100–116.

[26] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, 2010, pp. 921–930.

[27] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazières, "Protecting users by confining javascript with COWL," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, 2014, pp. 131–146.

[28] S. Tang, N. Dautenhahn, and S. T. King, "Fortifying web-based applications automatically," in *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, 2011, pp. 615–626.

[29] J. Weinberger, A. Barth, and D. Song, "Towards client-side HTML security policies," in *6th USENIX Workshop on Hot Topics in Security, HotSec'11, San Francisco, CA, USA, August 9, 2011*, 2011.

[30] M. West, A. Barth, and D. Veditz, "Content security policy (csp)," 2015, available at http://www.w3.org/TR/CSP/.

[31] A. Yip, N. Narula, M. N. Krohn, and R. Morris, "Privacy-preserving browser-side scripting with bflow," in *Proceedings of the 2009 EuroSys Conference, Nuremberg, Germany, April 1-3, 2009*, 2009, pp. 233–246.

[32] X. Zheng, J. Jiang, J. Liang, H. Duan, S. Chen, T. Wan, and N. Weaver, "Cookies lack integrity: Real-world implications," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, 2015, pp. 707–721.

APPENDIX

A. Formal Definition of the Reactive Semantics

A little bit of additional notation is useful before presenting the semantics. We write $\mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2)\downarrow$ when *transfer* is defined for an event of type event_type and a pair of tags τ_1, τ_2, otherwise we write $\mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2)\uparrow$. Notice that $\mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2)\downarrow$ is just a shorthand for $\mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) = (-, -, -, -)$.

The function $!K(u)^{\tau_r}$ retrieves from the cookie jar K all the cookies which should be read by the origin u, based on the tag τ_r passed to the cookie jar. Intuitively, the tag τ_r models the security assumptions on the reader: when cookies are fetched by the browser for inclusion in a HTTP(S) request to u, it may simply be the protocol of u; when cookies are accessed by a script running in the origin u, it may reflect the amount of trust we place on the script. The tag τ_r is indirectly passed to the transfer function in the definition of $!K(u)^{\tau_r}$ for an event of type get: only cookies set by the domain of u with a tag τ such that $\mathit{transfer}(\mathsf{get}, \tau_r, \tau)$ is defined are returned. Formally, we have:

$$!K(u)^{\tau_r} = \{\mathsf{ck}(k, v)^{\tau} \in K(\mathit{host}(u)) \mid \mathit{transfer}(\mathsf{get}, \tau_r, \tau)\downarrow\}$$

We let $K \xleftarrow[\tau_w]{CK} u$ be the cookie jar obtained as the result of updating the cookie jar K with the cookies CK set by the origin u; here, the tag τ_w represents the security assumptions on the writer. Again, this tag may correspond to the protocol of u for cookies set via HTTP(S) responses from u or it may reflect the trust we place in a script setting cookies while running in the origin u. Formally, we let $K \xleftarrow[\tau_w]{CK} u$ be the map coinciding with K, but for the entry $d = \mathit{host}(u)$, defined as follows:

$$\{\mathsf{ck}(k, v)^{\tau} \in CK \mid \mathit{transfer}(\mathsf{set}, \tau_w, \tau)\downarrow\} \cup \{\mathsf{ck}(k, v)^{\tau} \in K(d) \mid \forall\ \mathsf{ck}(k', v')^{\tau} \in CK : k' \neq k \vee \mathit{transfer}(\mathsf{set}, \tau_w, \tau)\uparrow\}$$

The definition includes two cases. In words, it states that, after the update, the cookie jar includes: (1) all the cookies in CK which can be legitimately set based on the transfer function for an event of type set, and (2) all the cookies in K which are not overwritten by cookies in CK, either because no cookie with the same key is included in CK or because the transfer function is undefined.

The transitions $C \xrightarrow{i} P$ in Table III describe how the consumer state C reacts to the input event i by evolving into a producer state P. The definition of $C \xrightarrow{i} P$ consists only of two rules: (I-MIRROR) and (I-COMPLETE), which are needed to make FF^{τ} a reactive system. The bulk of the semantics is defined by the auxiliary relation $C \overset{i}{\mapsto} P$.

TABLE III Reactive semantics of FF^{τ} - Input events

(I-LOAD)

$$\frac{\mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) = (\tau_n, -, \tau_{co}, -)}{\langle K, N, H, \mathsf{wait}, [\,]\rangle \xmapsto{\mathsf{load}(u)} \langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, \mathsf{doc\_req}(u : !K(u)^{\tau_{co}})\rangle}$$

(I-DOCRESP)

$$\frac{\mathit{transfer}(\mathsf{doc\_resp}, \tau_n, -) = (-, \tau_{ci}, -, \tau_s)}{\langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, [\,]\rangle \xmapsto{\mathsf{doc\_resp}_n(u:CK,e)} \langle K \xleftarrow[\tau_{ci}]{CK} u, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,]\rangle}$$

(I-DOCREDIR)

$$\frac{\mathit{transfer}(\mathsf{doc\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -) \qquad K' = K \xleftarrow[\tau_{ci}]{CK} u}{\langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, [\,]\rangle \xmapsto{\mathsf{doc\_redir}_n(u:CK,u')} \langle K', N \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K'(u)^{\tau_{co}})\rangle}$$

(I-XHRRESP)

$$\frac{\mathit{transfer}(\mathsf{xhr\_resp}, \tau_n, -) = (-, \tau_{ci}, -, \tau_s)}{\langle K, N, H \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}, \mathsf{wait}, [\,]\rangle \xmapsto{\mathsf{xhr\_resp}_n(u:CK,v)} \langle K \xleftarrow[\tau_{ci}]{CK} u, N, H, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,]\rangle}$$

(I-XHRREDIR)

$$\frac{\mathit{transfer}(\mathsf{xhr\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -) \qquad K' = K \xleftarrow[\tau_{ci}]{CK} u}{\langle K, N, H \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, [\,]\rangle \xmapsto{\mathsf{xhr\_redir}_n(u:CK,u')} \langle K', N, H \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K'(u')^{\tau_{co}})\rangle}$$

(I-MIRROR)

$$\frac{C \overset{i}{\mapsto} P}{C \xrightarrow{i} P}$$

(I-COMPLETE)

$$\frac{\nexists P : \langle K, N, H, \mathsf{wait}, [\,]\rangle \overset{i}{\mapsto} P}{\langle K, N, H, \mathsf{wait}, [\,]\rangle \xrightarrow{i} \langle K, N, H, \mathsf{wait}, \bullet\rangle}$$

When the user navigates the browser to a URL u, a new network connection n is created and is assigned a new tag τ_n by rule (I-LOAD), based on the tag of the URL u. Also, a new tag τ_{co} is computed and passed to the cookie jar, to determine

the set of cookies to be sent to u, and a new document request event including these cookies is put into the output buffer. If a HTTP(S) redirect to a URL u' is received over n, the tag of the network connection may be updated to a new tag τ_m as the result of rule (I-DocRedir). The update is determined by the tag τ_n of the network connection and the tag of the redirect URL u'. Notice that rule (I-DocRedir) also computes two tags which are needed to set and get cookies upon redirection respectively. When a document response is eventually received over a network connection, the connection is closed and a new script is run in the browser by rule (I-DocResp). The tag assigned to the script is computed by rule (I-DocResp), based on the tag of the network connection where the script is received. Observe that the tag of the network connection may actually keep track of the entire redirection chain which is followed before the script is downloaded. The treatment for XHR requests and responses is analogous to what we discussed above. The only difference is that XHR connections are not opened as the result of processing a $\mathsf{load}(u)$ event, but rather by script execution, which is explained in the next sub-section.

The transitions $P \xrightarrow{o} Q$ in Table IV describe how a producer state P generates an output event o and evolves into another state Q. Like for inputs, the semantics is split in two layers, with the rules (O-Mirror) and (O-Complete) accommodating the requirements of the reactive system definition and an auxiliary relation $P \overset{o}{\mapsto} Q$ formalizing the non-trivial moves. The first three rules for this relation are standard, so we just comment on the remaining rules.

TABLE IV Reactive semantics of FF^{τ} - Output events

(O-App)
$$\langle K, N, H, \lceil (\lambda x.e)\, v \rfloor^{\tau_s}_{@u}, [] \rangle \overset{\bullet}{\mapsto} \langle K, N, H, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u}, [] \rangle$$

(O-LetCtx)
$$\frac{\langle K, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [] \rangle \overset{o}{\mapsto} \langle K', N', H', \lceil e'' \rfloor^{\tau_s}_{@u}, [] \rangle}{\langle K, N, H, \lceil \mathsf{let}\ x = e\ \mathsf{in}\ e' \rfloor^{\tau_s}_{@u}, [] \rangle \overset{o}{\mapsto} \langle K', N', H', \lceil \mathsf{let}\ x = e''\ \mathsf{in}\ e' \rfloor^{\tau_s}_{@u}, [] \rangle}$$

(O-Let)
$$\langle K, N, H, \lceil \mathsf{let}\ x = v\ \mathsf{in}\ e \rfloor^{\tau_s}_{@u}, [] \rangle \overset{\bullet}{\mapsto} \langle K, N, H, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u}, [] \rangle$$

(O-GetCookie)
$$\frac{\exists \tau, v : \mathsf{ck}(k, v)^{\tau} \in\, !K(u)^{\tau_s}}{\langle K, N, H, \lceil \mathsf{get\text{-}ck}(k) \rfloor^{\tau_s}_{@u}, [] \rangle \overset{\bullet}{\mapsto} \langle K, N, H, \lceil v \rfloor^{\tau_s}_{@u}, [] \rangle}$$

(O-SetCookie)
$$\frac{CK = \{\mathsf{ck}(k, v)^{\tau} \mid \tau = \kappa(\mathit{host}(u), k)\}}{\langle K, N, H, \lceil \mathsf{set\text{-}ck}(k, v) \rfloor^{\tau_s}_{@u}, [] \rangle \overset{\bullet}{\mapsto} \langle K \xleftarrow{CK}_{\tau_s} u, N, H, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [] \rangle}$$

(O-Xhr)
$$\frac{\mathit{transfer}(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)}{\langle K, N, H, \lceil \mathsf{xhr}(u', \lambda x.e) \rfloor^{\tau_s}_{@u}, [] \rangle \xrightarrow{\mathsf{xhr\_req}(u' : !K(u')^{\tau_{co}})} \langle K, N, H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [] \rangle}$$

(O-Flush)
$$\langle K, N, H, T, o :: O \rangle \overset{o}{\mapsto} \langle K, N, H, T, O \rangle$$

(O-Mirror)
$$\frac{P \overset{o}{\mapsto} Q}{P \overset{o}{\rightarrow} Q}$$

(O-Complete)
$$\frac{\nexists o, Q : \langle K, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [] \rangle \overset{o}{\mapsto} Q}{\langle K, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [] \rangle \overset{\bullet}{\rightarrow} \langle K, N, H, \mathsf{wait}, [] \rangle}$$

Rules (O-GetCookie) and (O-SetCookie) model cookie access and cookie setting via JavaScript respectively. These rules make use of the function for getting and setting cookies we already discussed. The only point worth mentioning here is that the outcome of these two operations is determined by both the tag assigned to the script and the tag assigned to the cookie which is read/set. Notice that if a cookie with key k is set by a script running in the origin u, it always gets the tag $\kappa(\mathit{host}(u), k)$ intended for cookies with key k set by the domain owning the script. Hence, existing JavaScript code does not need to be adapted to include the tags intended for the cookies it sets.

Rule (O-Xhr) models the sending of an AJAX request by a script. Based on the tag of the script τ_s and the tag of the destination URL u', new tags τ_n and τ_{co} are computed for the new network connection and for accessing the cookie jar to fetch the cookies to be attached to the request. When the AJAX request is sent, its asynchronous behaviour is captured by binding the continuation $\lambda x.e$ to the new network connection where a response is expected and by immediately returning the dummy value unit, so that the script can immediately proceed in its execution. The continuation $\lambda x.e$ is later executed in the origin where the request was sent as the result of an application of rule (I-XhrResp), processing the corresponding response.

Finally, rule (O-Flush) is used to flush the output buffer when it is populated by an application of rule (I-Load): this is technically needed to move from a producer state back into a consumer state.

B. Constraint System

We propose a set of constraints on the transfer function, which ensure that a number of useful non-interference properties are enforced at runtime by FF^{τ}. Specifically, we are able to support:

1) confidentiality of cookies: the actual value of a high-confidentiality cookie has no visible import for the attacker (no cookie theft);
2) integrity of cookies: a high-integrity cookie cannot be set or modified by the attacker (no session fixation);
3) confidentiality of web sessions: the attacker is unaware of the presence of any on-going session between the browser and a trusted website;
4) integrity of web sessions: the attacker cannot force the browser into establishing new sessions with a trusted website (no login CSRF), or into introducing additional messages into existing sessions (no CSRF).

Constraint-checks are formulated as follows:

$$\Gamma \vdash \textit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) = (\ell_n, \ell_{ci}, \ell_{co}, \ell_s),$$

reading as: the entry of the transfer function fulfills the constraints in an environment Γ. We write $\Gamma \vdash \textit{transfer}$ when each entry of the transfer function fulfills the constraints. If the transfer function fulfills the constraints, it provably enforces meaningful non-interference policies on cookies and network communication. Specifically, it ensures that:

1) if a cookie is given label ℓ, its value can only be disclosed to an attacker ℓ' such that $C(\ell) \cap C(\ell') \neq \emptyset$ (*cookie confidentiality*);
2) if a cookie is given label ℓ, it can only be set or modified by an attacker ℓ' such that $I(\ell) \cap I(\ell') \neq \emptyset$ (*cookie integrity*);
3) if a URL u is given label ℓ, an attacker ℓ' can notice that the browser is loading u only if $C(\ell) \cap C(\ell') \neq \emptyset$ (*session confidentiality*);
4) if a URL u is given label ℓ, an attacker ℓ' can force the browser to send requests to u only if $I(\ell) \cap I(\ell') \neq \emptyset$ (*session integrity*)

These properties are enforced by assigning labels to different browser elements, e.g., network connections and scripts, and by ensuring that these labels constitute upper bounds for the security relevant side-effects which can be triggered by these browser elements.

TABLE V A non-interference constraint system for transfer functions

(T-LOAD)
$$\frac{\textit{evt\_label}(u) \cup C(\ell_n) \subseteq \Gamma_C(u) \qquad \textit{msg\_label}(u) \subseteq C(\ell_{co}) \qquad \textit{msg\_label}(u) \subseteq I(\ell_n)}{\Gamma \vdash \textit{transfer}(\mathsf{load}, u, -) = (\ell_n, -, \ell_{co}, -)}$$

(T-DOCRESP)
$$\frac{C(\ell_{ci}) \cup C(\ell_s) \subseteq C(\ell_n) \qquad I(\ell_n) \subseteq I(\ell_{ci}) \cap I(\ell_s)}{\Gamma \vdash \textit{transfer}(\mathsf{doc\_resp}, \ell_n, u) = (-, \ell_{ci}, -, \ell_s)}$$

(T-DOCREDIR)
$$\frac{\textit{evt\_label}(u) \cup C(\ell_{ci}) \cup C(\ell_m) \subseteq C(\ell_n) \qquad \textit{msg\_label}(u) \subseteq C(\ell_{co}) \qquad I(\ell_n) \cup \textit{msg\_label}(u) \subseteq I(\ell_m) \qquad I(\ell_n) \subseteq I(\ell_{ci}) \cap \Gamma_I(u)}{\Gamma \vdash \textit{transfer}(\mathsf{doc\_redir}, \ell_n, u) = (\ell_m, \ell_{ci}, \ell_{co}, -)}$$

(T-XHRRESP)
$$\frac{C(\ell_{ci}) \cup C(\ell_s) \subseteq C(\ell_n) \qquad I(\ell_n) \subseteq I(\ell_{ci}) \cap I(\ell_s)}{\Gamma \vdash \textit{transfer}(\mathsf{xhr\_resp}, \ell_n, u) = (-, \ell_{ci}, -, \ell_s)}$$

(T-XHRREDIR)
$$\frac{\textit{evt\_label}(u) \cup C(\ell_{ci}) \cup C(\ell_m) \subseteq C(\ell_n) \qquad \textit{msg\_label}(u) \subseteq C(\ell_{co}) \qquad I(\ell_n) \cup \textit{msg\_label}(u) \subseteq I(\ell_m) \qquad I(\ell_n) \subseteq I(\ell_{ci}) \cap \Gamma_I(u)}{\Gamma \vdash \textit{transfer}(\mathsf{xhr\_redir}, \ell_n, u) = (\ell_m, \ell_{ci}, \ell_{co}, -)}$$

(T-GET)
$$\frac{C(\ell_r) \subseteq C(\ell_t) \qquad I(\ell_t) \subseteq I(\ell_r)}{\Gamma \vdash \textit{transfer}(\mathsf{get}, \ell_r, \ell_t) = (-, -, -, -)}$$

(T-SET)
$$\frac{C(\ell_t) \subseteq C(\ell_w) \qquad I(\ell_w) \subseteq I(\ell_t)}{\Gamma \vdash \textit{transfer}(\mathsf{set}, \ell_w, \ell_t) = (-, -, -, -)}$$

(T-SEND)
$$\frac{\textit{evt\_label}(u) \cup C(\ell_n) \subseteq C(\ell_s) \qquad \textit{msg\_label}(u) \subseteq C(\ell_{co}) \qquad I(\ell_s) \cup \textit{msg\_label}(u) \subseteq I(\ell_n) \qquad I(\ell_s) \subseteq \Gamma_I(u)}{\Gamma \vdash \textit{transfer}(\mathsf{send}, \ell_s, u) = (\ell_n, -, \ell_{co}, -)}$$

The constraints are given in Table V, we comment on them in the following. Before diving into the details, however, it is worth discussing how the functions *evt_label* and *msg_label* are consistently used in the rules: this is not obvious, since the attacker capabilities discriminate between message presence (*evt_label*) and message contents (*msg_label*). Intuitively, in the constraint system we only use *msg_label* when determining the set of cookies to be attached to a HTTP(S) request, since we only care about whether the remote end-point of a network connection should be entitled to get some cookies; in all the

other cases, we use *evt_label* to ensure the absence of implicit information flows based on message presence. Recall that *msg_label*$(u) \subseteq C(\ell)$ implies *evt_label*$(u) \subseteq C(\ell)$ for all URLs u and all the well-formed attackers ℓ.

Rule (T-LOAD) ensures that, when a URL u is loaded, the information $\Gamma_C(u)$ is an upper bound for both *evt_label*(u) and the confidentiality label $C(\ell_n)$ of the new network connection. Having *evt_label*$(u) \subseteq \Gamma_C(u)$ implies that the load event is always visible to any network attacker or any web attacker sitting at *host*(u), while having $C(\ell_n) \subseteq \Gamma_C(u)$ guarantees that the side-effects produced by a response received over the network connection are only visible to $\Gamma_C(u)$. The rule also checks two other conditions: *msg_label*$(u) \subseteq C(\ell_{co})$ is needed to ensure that the cookies attached to the document request sent to u can actually be disclosed to it, while *msg_label*$(u) \subseteq I(\ell_n)$ formalizes that an attacker who controls u may be able to compromise the integrity of any response received over the new network connection.

Rules (T-DOCRESP) and (T-XHRRESP) check that the confidentiality label $C(\ell_n)$ of the network connection where a response is received is an upper bound for the confidentiality label $C(\ell_{ci})$ of the cookies which are set in the HTTP(S) headers of the response, so that the attacker cannot infer the occurrence of private input events from the value of public cookies possibly set in such events. Also, the rules check that $C(\ell_n)$ is an upper bound for the confidentiality label $C(\ell_s)$ of the script downloaded over the network connection, so that the script cannot produce unexpected visible side-effects. The conditions on integrity are dual: a network connection can only be used to set cookies and spawn scripts with lower integrity than the integrity label of the connection itself.

Rules (T-DOCREDIR) and (T-XHRREDIR) are more complex, since HTTP(S) redirects have a number of side-effects: a new network connection is opened, new cookies are set into the browser, and cookies available in the cookie jar are fetched to compose an HTTP(S) request. Enforcing non-interference in presence of a redirect to u thus requires some care. The confidentiality label $C(\ell_n)$ of the network connection where the redirect is received must be an upper bound for: (1) *evt_label*(u), since a network request is sent to u and made visible to any attacker who controls the URL; (2) the label $C(\ell_{ci})$ of the new cookies set in the browser, otherwise these cookies could be exploited to reveal the occurrence of a redirect over a high-confidentiality connection, and (3) the label $C(\ell_m)$ of the new network connection, which otherwise may leak the existence of a previously opened connection. To ensure the confidentiality of the cookies sent in the redirect, the condition *msg_label*$(u) \subseteq C(\ell_{co})$ must be checked, just like in the case of rule (T-LOAD). As to integrity, two conditions must be verified. First, the integrity label $I(\ell_m)$ of the new network connection must be an upper bound for the integrity label of the original connection $I(\ell_n)$ and for *msg_label*(u), so that low-integrity connections cannot ever be endorsed to high integrity and all the origins involved in a redirection chain are actually tracked in the integrity label of the connection. Second, we check that the integrity label $I(\ell_n)$ of the original network connection is a lower bound for both the integrity label $I(\ell_{ci})$ of the new cookies set in the browser and the information $\Gamma_I(u)$, so that low-integrity connections cannot be used to set high-integrity cookies or to send requests to security-sensitive URLs.

Rule (T-GET) ensures that a cookie with confidentiality label $C(\ell_t)$ can only be accessed by a reader with confidentiality label $C(\ell_r) \subseteq C(\ell_t)$; dually, a cookie with integrity label $I(\ell_t)$ can only be accessed by a reader with integrity label $I(\ell_r) \supseteq I(\ell_t)$. The first condition guarantees that high-confidentiality cookies are only disclosed to their intended readers, while the second condition ensures that low-integrity cookies cannot affect the execution of high-integrity scripts.

Rule (T-SET) is dual to rule (T-GET). It checks that a cookie with confidentiality label $C(\ell_t)$ can only be set by a writer with confidentiality label $C(\ell_w) \supseteq C(\ell_t)$; dually, a cookie with integrity label $I(\ell_t)$ can only be set by a reader with integrity label $I(\ell_w) \subseteq I(\ell_t)$. The first condition guarantees that high-confidentiality scripts cannot reveal their secrets by setting low-confidentiality cookies, while the second condition ensures that high-integrity cookies can only be set or modified by their intended writers.

Rule (T-SEND) enforces similar invariants to the rules for redirections (T-DOCREDIR) and (T-XHRREDIR). The only difference with respect to these rules is that, in contrast to HTTP(S) redirects, the sending of an AJAX request does not directly set new cookies in the browser, so the conditions to check are slightly more concise.

Having defined a constraint system for transfer functions, we now formalize which non-interference properties are supported by it. The following theorems are similar to the results presented in Section V-C, but instead of talking about the canonical transfer function we now talk of all the transfer functions that satisfy the constraints.

Theorem 4 (Confidentiality)**.** *Assume that* $\Gamma \vdash$ *transfer and let* $\pi_C = \langle rel_\ell, \sim_\ell \rangle$ *be the confidentiality policy such that:*

1) $\forall i : \neg rel_\ell(i) \triangleq i = \mathsf{load}(u) \wedge \Gamma_C(u) \cap C(\ell) = \emptyset$;
2) $\forall i, i' : i \sim_\ell i' \Leftrightarrow erase^C_\ell(i) = erase^C_\ell(i')$.

Then, FF^τ *is non-interferent under* π_C *when implementing transfer.*

Theorem 5 (Integrity)**.** *Assume that* $\Gamma \vdash$ *transfer and let* $\pi_I = \langle rel_\ell, \sim_\ell \rangle$ *be the integrity policy such that:*

1) $\forall o : rel_\ell(o) \triangleq o = \mathsf{net\_req}(u : CK) \wedge \Gamma_I(u) \cap I(\ell) = \emptyset$;
2) $\forall o, o' : o \sim_\ell o' \Leftrightarrow erase^I_\ell(o) = erase^I_\ell(o')$.

Then, FF^τ *is non-interferent under* π_I *when implementing transfer.*

As explained in the proof sketch in Section V-C, we show that the canonical transfer function always satisfies the constraints, in order to obtain the non-interference results for the canonical transfer function.

Theorem 6. *If* $\Gamma, f \rhd \mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) \rightsquigarrow \vec{\ell}$, *then* $\Gamma \vdash \mathit{transfer}(\mathsf{event\_type}, \tau_1, \tau_2) = \vec{\ell}$.

Proof. By a case analysis on the rule applied to prove the judgement in the premise and an application of the corresponding constraint-check, using the well-formedness of f. □

C. Failure Semantics

Table VI shows an excerpt of the failure semantics. It is used in the compatibility theorem to make failing runtime checks in the transfer function explicit in the output stream.

TABLE VI Failure semantics of FF^τ (excerpt)

(I-LOAD1)
$$\frac{\Gamma, f \rhd \mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) \rightsquigarrow (\tau_n, -, \tau_{co}, -) \qquad \forall \mathsf{ck}(k,v)^\tau \in K(\mathit{host}(u)) : \Gamma, f \rhd \mathit{transfer}(\mathsf{get}, \tau_{co}, \tau) \rightsquigarrow (-,-,-,-)}{\langle K, N, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{load}(u)}{\longmapsto}_\Gamma \langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, \mathsf{doc\_req}(u : !K(u)^{\tau_{co}})\rangle}$$

(I-LOAD2)
$$\frac{\nexists \tau_n, \tau_{co} : \Gamma, f \rhd \mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) \rightsquigarrow (\tau_n, -, \tau_{co}, -)}{\langle K, N, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{load}(u)}{\longmapsto}_\Gamma \langle K, N, H, \mathsf{wait}, \star\rangle}$$

(I-LOAD3)
$$\frac{\Gamma, f \rhd \mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) \rightsquigarrow (\tau_n, -, \tau_{co}, -) \qquad \exists \mathsf{ck}(k,v)^\tau \in K(\mathit{host}(u)) : \nexists t : \Gamma, f \rhd \mathit{transfer}(\mathsf{get}, \tau, \tau_{co}) \rightsquigarrow t}{\langle K, N, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{load}(u)}{\longmapsto}_\Gamma \langle K, N, H, \mathsf{wait}, \star\rangle}$$

(I-DOCRESP1)
$$\frac{\Gamma, f \rhd \mathit{transfer}(\mathsf{doc\_resp}, \tau_n, -) \rightsquigarrow (-, \tau_{ci}, -, \tau_s) \qquad \forall \mathsf{ck}(k,v)^\tau \in CK : \Gamma, f \rhd \mathit{transfer}(\mathsf{set}, \tau_{co}, \tau) \rightsquigarrow (-,-,-,-)}{\langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{doc\_resp}_n(u:CK,e)}{\longmapsto}_\Gamma \langle K \overset{CK}{\leftarrow}_{\tau_{ci}} u, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,]\rangle}$$

(I-DOCRESP2)
$$\frac{\nexists \tau_{ci}, \tau_s : \Gamma, f \rhd \mathit{transfer}(\mathsf{doc\_resp}, \tau_n, -) \rightsquigarrow (-, \tau_{ci}, -, \tau_s)}{\langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{doc\_resp}_n(u:CK,e)}{\longmapsto}_\Gamma \langle K, N, H, \mathsf{wait}, \star\rangle}$$

(I-DOCRESP3)
$$\frac{\Gamma, f \rhd \mathit{transfer}(\mathsf{doc\_resp}, \tau_n, -) \rightsquigarrow (-, \tau_{ci}, -, \tau_s) \qquad \exists \mathsf{ck}(k,v)^\tau \in CK : \nexists t : \Gamma, f \rhd \mathit{transfer}(\mathsf{set}, \tau_{co}, \tau) \rightsquigarrow t}{\langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, [\,]\rangle \overset{\mathsf{doc\_resp}_n(u:CK,e)}{\longmapsto}_\Gamma \langle K, N, H, \mathsf{wait}, \star\rangle}$$

(O-GETCOOKIE1)
$$\frac{\exists \tau, v : \mathsf{ck}(k,v)^\tau \in K(\mathit{host}(u)) \qquad \Gamma, f \rhd \mathit{transfer}(\mathsf{get}, \tau_{co}, \tau) \rightsquigarrow (-,-,-,-)}{\langle K, N, H, \lceil \mathsf{get\text{-}ck}(k) \rfloor^{\tau_s}_{@u}, [\,]\rangle \overset{\bullet}{\mapsto}_\Gamma \langle K, N, H, \lceil v \rfloor^{\tau_s}_{@u}, [\,]\rangle}$$

(O-GETCOOKIE2)
$$\frac{\exists \tau, v : \mathsf{ck}(k,v)^\tau \in K(\mathit{host}(u)) \qquad \nexists t : \Gamma, f \rhd \mathit{transfer}(\mathsf{get}, \tau_{co}, \tau) \rightsquigarrow t}{\langle K, N, H, \lceil \mathsf{get\text{-}ck}(k) \rfloor^{\tau_s}_{@u}, [\,]\rangle \overset{\star}{\mapsto}_\Gamma \langle K, N, H, \mathsf{wait}, [\,]\rangle}$$

(O-SETCOOKIE1)
$$\frac{\tau = \kappa(\mathit{host}(u), k) \qquad CK = \{\mathsf{ck}(k,v)^\tau\} \qquad \Gamma, f \rhd \mathit{transfer}(\mathsf{set}, \tau_s, \tau) \rightsquigarrow (-,-,-,-)}{\langle K, N, H, \lceil \mathsf{set\text{-}ck}(k,v) \rfloor^{\tau_s}_{@u}, [\,]\rangle \overset{\bullet}{\mapsto}_\Gamma \langle K \overset{CK}{\leftarrow}_{\tau_s} u, N, H, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,]\rangle}$$

(O-SETCOOKIE2)
$$\frac{\tau = \kappa(\mathit{host}(u), k) \qquad \nexists t : \Gamma, f \rhd \mathit{transfer}(\mathsf{set}, \tau_s, \tau) \rightsquigarrow t}{\langle K, N, H, \lceil \mathsf{set\text{-}ck}(k,v) \rfloor^{\tau_s}_{@u}, [\,]\rangle \overset{\star}{\mapsto}_\Gamma \langle K, N, H, \mathsf{wait}, [\,]\rangle}$$

As opposed to the original semantics of FF^τ, the failure semantics contains multiple rules per event - one rule for the case in which no failure occurs and one rule for each possible failure of the transfer function. We elaborate this on the example of a load-event. For a load-event there are two possible sources of failure:

1) The transfer function can fail for the load-event, because the runtime check in rule (G-LOAD) fails.

2) The transfer function can fail for the get-event, because one of the runtime checks in rule (G-GET) fails for one cookie while retrieving the cookies from the cookie jar.

The rule (I-LOAD1) treats the case in which none of the failures occurs (this is ensured by the premise). The rule is then equivalent to the original rule (I-LOAD). The rule (I-LOAD2) covers the case of the failure described in 1) while the rule (I-LOAD3) covers the case of the failure described in 2). The design of the rules for other events is analogous.

We now give the proof to Theorem 3.

Theorem 3 (Compatibility)**.** *Let C_0 be the initial state of $\mathsf{FF}^{\tau}_{\star}(\Gamma_{\top}, id)$ and assume that the function $\kappa : \mathcal{D} \times \mathcal{S} \to$ Tags assigns the top label $\top$ to all the elements of its domain. If $C_0(I) \Downarrow O$, then $\star$ does not occur in O.*

Proof. By a case analysis of the reduction rules, one can easily verify that $C(\ell_n) = C(\ell_s) = \top_s$ for all labels of network connections and scripts, so the confidentiality checks in rule (G-LOAD), (G-DOCREDIR), (G-XHRREDIR) and (G-SEND) trivially succeed. We can also observe $C(\ell_{ci}) = \top_s$ for all cookie writes, so the confidentiality condition on (G-SET) is always true. Because of κ, we know $C(\tau) = \top_s$ for all cookies $\mathsf{ck}(k, v)^{\tau}$ in the cookie jar, and hence the confidentiality condition on (G-GET) holds true. As we know $\Gamma_I(u) = \top_s$ for all u, the integrity checks in (G-DOCREDIR), (G-XHRREDI) and (G-SEND) succeed. One can observe that we have $I(\ell_{co}) = \top_s$ for cookie reads, hence the integrity check in (G-GET) succeeds as well. Because of κ, we know $I(\tau) = \top_s$ for all cookies $\mathsf{ck}(k, v)^{\tau}$ and hence the integrity condition in (G-SET) holds true. □

D. Additional Examples

1) Cookie Protection Against Network Attackers: The Secure attribute is the standard defense mechanism for authentication cookies against network attackers [3]. If a cookie is marked as Secure, the browser will only attach it to HTTPS requests, thus ensuring that the cookie is never sent in clear. The Secure attribute does not provide strong integrity guarantees, since Secure cookies set over HTTPS can be overwritten by non-Secure cookies set over HTTP [32].

It is indeed much harder to protect cookies against powerful attackers, like network attackers. Based on the previous informal explanation, it seems natural to model the Secure attribute in our framework by giving cookies set by the domain d the following label: $\ell_c = (\{\mathsf{https}(d)\}, \{\mathsf{http}(d), \mathsf{https}(d)\})$. However, if we want to let scripts access these cookies, as it normally happens for Secure cookies, the price to pay for non-interference is high: in particular, scripts downloaded from the domain d cannot perform *any* network communication at all. Technically, this is a consequence of the observation that scripts accessing these cookies must have a label ℓ such that $C(\ell) \subseteq \{\mathsf{https}(d)\}$ by rule (G-GET), but any attempt to communicate with a URL u by these scripts would fail by rule (G-SEND), given that the rule requires *evt_label*$(u) \subseteq C(\ell)$, but *evt_label*$(u) \not\subseteq \{\mathsf{https}(d)\}$ for all u. Though this looks restrictive, it is actually correct, since the *presence* of any network communication is visible to a network attacker, hence it may act as a side-channel to leak the cookie value. As an extreme but simple example, a script may branch over a conditional, checking whether the value of the cookie is equal to a given string, and only send a bit over the network if this is true, thus revealing the cookie value.

If scripts do not need to access authentication cookies, which is the most common case for web applications, protection against network attackers can be enforced by raising the label of scripts to $\top$. This solution would not constrain network communication and would mimic the behaviour of standard cookies marked as both HttpOnly and Secure, though with the notable caveats on integrity discussed for HttpOnly cookies. Better protection against active network attackers can be implemented by changing the integrity label of cookies to just $\{\mathsf{https}(d)\}$, thus preventing cookies from being set or overwritten over HTTP.

2) Preventing Cross-Site Request Forgery: It is easy to provide protection against CSRF attacks in our framework, since the format of our labelling immediately supports specifications in the spirit of Allowed Referrer Lists (ARLs) [13], but with better security guarantees. ARLs allow individual websites to specify which URLs are legitimately entitled to send authenticated requests to them: if the browser attempts to contact a website from an address which is not included in the ARL specified by the website, the authentication cookies are not attached.

Moving to our framework, assume that a URL u should only accept authenticated requests from its own domain d and another domain d'. We can specify a labelling Γ such that $\Gamma_I(u) = \{\mathsf{http}(d), \mathsf{https}(d), \mathsf{http}(d'), \mathsf{https}(d')\}$. Notice that the protection granted by this labelling is quite strong: not only it ensures that authenticated requests to u can only be sent by pages hosted on d or d', but it also guarantees that the attacker cannot force d or d' into abusing their privileges by exploiting reflected XSS attacks enabled by HTTP(S) redirects.

To exemplify, let u_a be a HTTP URL pointing to the attacker website and u_d be a HTTP URL on the domain d, and pick the following input stream:

$$I \quad = \quad [\mathsf{load}(u_a), \mathsf{doc\_redir}_n(u_a : \emptyset, u_d), \mathsf{doc\_resp}_n(u_d : \emptyset, \mathsf{xhr}(u, \lambda x.\mathsf{unit}))].$$

The input stream I models the behaviour of an attacker abusing an HTTP(S) redirection from u_a to u_d to mount a reflected XSS attack on u_d, which then attempts to force the page hosted at u_d into sending an authenticated request to u. This request, however, will not be sent under the labelling above, though it is fired from the domain d. The reason is that the network connection n instantiated when processing the $\mathsf{load}(u_a)$ event is initially given an integrity label $\{\mathsf{http}(d_a)\}$, where d_a is the attacker-controlled domain hosting the malicious page at u_a. The redirection then changes the integrity label of n to

$\{\mathsf{http}(d_a), \mathsf{http}(d)\}$ and this label is inherited by the script running in the origin u_d. Since $\{\mathsf{http}(d_a), \mathsf{http}(d)\} \not\subseteq \Gamma_I(u)$, the XHR request is dropped. Protecting from this class of attacks is beyond the capabilities of ARLs, since the last malicious request would be sent by an allowed referrer.

E. Preliminaries

The proof of these results is given in [8].

Definition 11 (Unwinding Relation)**.** *An* unwinding relation *is a label-indexed family of binary relations $\mathcal{R}_\ell$ on states of a reactive system with the following properties:*

1) *if $Q \mathcal{R}_\ell Q'$, then $Q' \mathcal{R}_\ell Q$;*
2) *if $C \mathcal{R}_\ell C'$ and $C \xrightarrow{i} P$ and $C' \xrightarrow{i'} P'$ and $i \sim_\ell i'$ with $rel_\ell(i)$ and $rel_\ell(i')$, then $P \mathcal{R}_\ell P'$;*
3) *if $C \mathcal{R}_\ell C'$ and $C \xrightarrow{i} P$ with $\neg rel_\ell(i)$, then $P \mathcal{R}_\ell C'$;*
4) *if $P \mathcal{R}_\ell C$ and $P \xrightarrow{o} Q$, then $\neg rel_\ell(o)$ and $Q \mathcal{R}_\ell C$;*
5) *if $P \mathcal{R}_\ell P'$, then either of the following conditions hold true:*
   a) *$P \xrightarrow{o} Q$ and $P' \xrightarrow{o'} Q'$ with $o \sim_\ell o'$ and $Q \mathcal{R}_\ell Q'$;*
   b) *$P \xrightarrow{o} Q$ with $\neg rel_\ell(o)$ and $Q \mathcal{R}_\ell P'$;*
   c) *$P' \xrightarrow{o'} Q'$ with $\neg rel_\ell(o')$ and $P \mathcal{R}_\ell Q'$.*

Theorem 7. *Let C_0 be the initial state of a reactive system R. If $C_0 \mathcal{R}_\ell C_0$ for some unwinding relation $\mathcal{R}$, then R satisfies non-interference.*

F. Proof of Confidentiality

Definition 12 (Low Equivalence)**.** *We define an* erasure *operator $erase_\ell^C(\cdot)$ on different browser data structures:*

- *$erase_\ell^C(K)$ is the map with domain $dom(k)$ such that for all $d \in dom(K)$: $(erase_\ell^C(K))(d) = ck\text{-}erase_\ell^C(K(d))$*
- *$erase_\ell^C(N)$ is the map obtained from N by erasing all the entries $n^\tau : u$ such that $C(\tau) \cap C(\ell) = \emptyset$;*
- *$erase_\ell^C(H)$ is the map obtained from H by erasing all the entries $n^\tau : (u, \lceil \lambda x.e \rfloor_{@u'})$' such that $C(\tau) \cap C(\ell) = \emptyset$;*
- *$erase_\ell^C(T) = \mathsf{wait}$ whenever $T = \lceil e \rfloor^\tau_{@u}$ with $C(\tau) \cap C(\ell) = \emptyset$, while $erase_\ell^C(T) = T$ otherwise.*

We then define a binary low equivalence *relation $\simeq_\ell^C$ between data structures coinciding after applying the erasure $erase_\ell^C(\cdot)$.*

Definition 13 (Candidate for Confidentiality)**.** *Let $Q = \langle K, N, H, T, O\rangle$ and $Q' = \langle K', N', H', T', O'\rangle$, we write $Q \mathcal{R}_\ell^C Q'$ if and only if: (1) $K \simeq_\ell^C K'$; (2) $N \simeq_\ell^C N'$; (3) $H \simeq_\ell^C H'$; (4) $T \simeq_\ell^C T'$; (5) $O \approx_\ell O'$.*

Lemma 1. *For the initial state C_0 we have $C_0 \mathcal{R}_\ell^C C_0$.*

Proof. This follows directly from the reflexivity of $\simeq_\ell^C$ and $\approx_\ell$. □

Lemma 2. *$\mathcal{R}_\ell^C$ satisfies the first condition of Definition 11.*

Proof. A straightforward syntactic check on the definition of $\mathcal{R}_\ell^C$. □

Lemma 3 (Visible Cookies)**.** *Let $\Gamma \vdash$ transfer and $K \simeq_\ell^C K'$. If $C(\tau) \cap C(\ell) \neq \emptyset$, then $!K(u)^\tau = !K'(u)^\tau$ for any u.*

Proof. We only show $!K(u)^\tau \subseteq !K'(u)^\tau$, the proof of the other direction is analogous.

Let $\mathsf{ck}(k, v)^{\tau'} \in !K(u)^\tau$. Then we know that $transfer(\mathsf{get}, \tau, \tau')\downarrow$. By the constraints of the transfer function we know from the rule (T-GET) that $C(\tau) \subseteq C(\tau')$. It follows that $C(\tau') \cap C(\ell) \neq \emptyset$ and hence $\mathsf{ck}(k, v)^{\tau'} \in !(erase_\ell^C(K'))(u)^\tau$ by the definition of $erase_\ell^C(K)$. Because of $K \simeq_\ell^C K'$, we also have $\mathsf{ck}(k, v)^{\tau'} \in !(erase_\ell^C(K'))(u)^\tau$ and hence $\mathsf{ck}(k, v)^{\tau'} \in !K'(u)^\tau$. □

Lemma 4 (Visible Updates)**.** *Let $K \simeq_\ell^C K'$. If $ck\text{-}erase_\ell^C(CK) = ck\text{-}erase_\ell^C(CK')$, then $K \xleftarrow{CK}_\tau u \simeq_\ell^C K' \xleftarrow{CK'}_\tau u$ for any u, τ.*

Proof. To show the claim we have to show that $ck\text{-}erase_\ell^C((K \xleftarrow{CK}_\tau u)(d)) = ck\text{-}erase_\ell^C((K' \xleftarrow{CK'}_\tau u)(d))$ for all domains d. We distinguish two cases:

- If $d \neq host(u)$ then we have $K(d) = (K \xleftarrow{CK}_\tau u)(d)$ and $K'(d) = (K' \xleftarrow{CK'}_\tau u)(d)$. As we know $ck\text{-}erase_\ell^C(K(d)) = ck\text{-}erase_\ell^C(K'(d))$ by the definition of $K \simeq_\ell^C K'$ we can conclude $ck\text{-}erase_\ell^C((K \xleftarrow{CK}_\tau u)(d)) = ck\text{-}erase_\ell^C((K' \xleftarrow{CK'}_\tau u)(d))$.
- If $d = host(u)$ then let $\mathsf{ck}(k_1, v_1)^{\tau_1} \in ck\text{-}erase_\ell^C((K \xleftarrow{CK}_\tau u)(d))$. This implies $\mathsf{ck}(k_1, v_1)^{\tau_1} \in (K \xleftarrow{CK}_\tau u)(d)$ and $C(\tau_1) \cap C(\ell) \neq \emptyset$.

  We do a case distinction following the definition of $(K \xleftarrow{CK}_\tau u)(d)$:

- If $\mathsf{ck}(k_1, v_1)^{\tau_1} \in CK$ with $\mathit{transfer}(\mathsf{set}, \tau, \tau_1)\downarrow$, then because of $\mathit{ck\text{-}erase}^C_\ell(CK) = \mathit{ck\text{-}erase}^C_\ell(CK')$ and $C(\tau_1) \cap C(\ell) \neq \emptyset$ we know that $\mathsf{ck}(k_1, v_1)^{\tau_1} \in CK'$, hence $\mathsf{ck}(k_1, v_1)^{\tau_1} \in (K' \xleftarrow{CK'}_\tau u)(d)$.
- If $\mathsf{ck}(k_1, v_1)^{\tau_1} \in K(d)$ and for all $\mathsf{ck}(k_2, v_2)^{\tau_2} \in CK$ we have $k_1 \neq k_2$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$ then let $\mathsf{ck}(k_3, v_3)^{\tau_3} \in CK'$. We have to show that $k_3 \neq k_1$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_3)\uparrow$. We perform a case distinction:
  - If $k_3 \neq k_1$ then the claim follows trivially.
  - If $k_3 = k_1$ then we observe that $\tau_3 = \kappa(d, k_3) = \kappa(d, k_1) = \tau_1$. This implies $C(\tau_3) \cap C(\ell) \neq \emptyset$ and hence $\mathsf{ck}(k_3, v_3)^{\tau_3} \in CK$ because $\mathit{ck\text{-}erase}^C_\ell(CK) = \mathit{ck\text{-}erase}^C_\ell(CK')$. We then know $\mathit{transfer}(\mathsf{set}, \tau, \tau_3)\uparrow$, which concludes the proof.

□

Lemma 5 (Invisible Updates)**.** *Let* $\Gamma \vdash$ *transfer and* $C(\tau) \cap C(\ell) = \emptyset$*. Then* $K \simeq^C_\ell K \xleftarrow{CK}_\tau u$ *for all* K, u, CK*.*

Proof. We show both directions of set inclusions:

- Let $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^C_\ell(K(d))$. This implies $C(\tau_1) \cap C(\ell) \neq \emptyset$. Let $\mathsf{ck}(k_2, v_2)^{\tau_2} \in CK$. To show $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^C_\ell((K \xleftarrow{CK}_\tau u)(d))$ we have to show $k_2 \neq k_1$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$. We perform a case distinction:
  - if $k_2 \neq k_1$ then the claim follows trivially.
  - If $k_2 = k_1$ then we observe that $\tau_2 = \kappa(d, k_2) = \kappa(d, k_1) = \tau_1$. This implies $C(\tau_2) \cap C(\ell) \neq \emptyset$, hence $C(\tau_2) \not\subseteq C(\tau)$ and by the constraints we have $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$.
- Let $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^C_\ell((K \xleftarrow{CK}_\tau u)(d))$. This implies $C(\tau_1) \cap C(\ell) \neq \emptyset$. We show that for all $\mathsf{ck}(k_2, v_2)^{\tau_2} \in CK$ we have $k_2 \neq k_1$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$. We do a case distinction:
  - If $k_2 \neq k_1$ then the claim follows trivially.
  - If $k_2 = k_1$ then we observe that $\tau_2 = \kappa(d, k_2) = \kappa(d, k_1) = \tau_1$. This implies $C(\tau_2) \cap C(\ell) \neq \emptyset$, hence $C(\tau_2) \not\subseteq C(\tau)$ and we have $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$.

□

Lemma 6. *$\mathcal{R}^C_\ell$ satisfies the second condition of Definition 11.*

Proof. Let $C = \langle K, N, H, \mathsf{wait}, [\,]\rangle$ and $C' = \langle K', N', H', \mathsf{wait}, [\,]\rangle$ with $C\,\mathcal{R}^C_\ell\,C'$. By definition of $\mathcal{R}^C_\ell$, we have:

(1) $K \simeq^C_\ell K'$;
(2) $N \simeq^C_\ell N'$;
(3) $H \simeq^C_\ell H'$.

Assume that $C \xrightarrow{i} P$ and $C' \xrightarrow{i'} P'$ with $i \sim_\ell i'$ and $\mathit{rel}_\ell(i)$ and $\mathit{rel}_\ell(i')$. We perform a case distinction on i:

- if $i = \mathsf{load}(u)$, then $i' = \mathsf{load}(u)$ by definition of $\sim_\ell$. Given that we assume $\mathit{rel}_\ell(i)$, we have $\Gamma_C(u) \cap C(\ell) \neq \emptyset$. By the reduction rules, assuming that $\mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) = (\ell_n, -, \ell_{co}, -)$, we then have:

$$\begin{array}{rcl} P & = & \langle K, N \uplus \{n^{\ell_n} : u\}, H, \mathsf{wait}, \mathsf{doc\_req}(u : !K(u)^{\ell_{co}})\rangle \\ P' & = & \langle K', N' \uplus \{n^{\ell_n} : u\}, H', \mathsf{wait}, \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})\rangle \end{array}$$

  To prove the desired conclusion, we need to show:
  (a) $N \uplus \{n^{\ell_n} : u\} \simeq^C_\ell N' \uplus \{n^{\ell_n} : u\}$
  (b) $\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}) \approx_\ell \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})$

  Point (a) is an immediate consequence of (2), while point (b) is more complicated. We distinguish three sub-cases:
  - if $\mathit{evt\_label}(u) \not\subseteq C(\ell)$, then $\neg\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}))$ and $\neg\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K'(u)^{\ell_{co}}))$, hence we conclude by applying rules (S-LEFT), (S-RIGHT) and (S-EMPTY);
  - if $\mathit{evt\_label}(u) \subseteq C(\ell)$ and $\mathit{msg\_label}(u) \not\subseteq C(\ell)$, then $\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}))$ and $\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K'(u)^{\ell_{co}}))$. However, we also know that $\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}) \sim_\ell \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})$, hence we conclude by applying rule (S-MATCH);
  - if $\mathit{evt\_label}(u) \subseteq C(\ell)$ and $\mathit{msg\_label}(u) \subseteq C(\ell)$, then $\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}))$ and $\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K'(u)^{\ell_{co}}))$. By the constraints of the transfer function, we know that $\mathit{msg\_label}(u) \subseteq C(\ell_{co})$. This implies $C(\ell_{co}) \cap C(\ell) \neq \emptyset$ and hence $!K(u)^{\ell_{co}} = !K'(u)^{\ell_{co}}$ by using (1) and Lemma 3, which is enough to conclude by rule (S-MATCH), using the reflexivity of the $\sim_\ell$ relation;
- if $i = \mathsf{doc\_resp}_n(u : CK, e)$, then $i' = \mathsf{doc\_resp}_n(u : CK', e)$ with $\mathit{ck\text{-}erase}^C_\ell(CK) = \mathit{ck\text{-}erase}^C_\ell(CK')$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:

– assume that rule (I-DocResp) is used on both C and C', then we know that $N = N_0 \uplus \{n^{\tau_n} : u\}$ and $N' = N_1 \uplus \{n^{\tau'_n} : u\}$ for some N_0, N_1 such that $N_0 \simeq_\ell^C N_1$. Assuming that *transfer*(doc_resp, τ_n, *tag*(u)) $= (-, \tau_{ci}, -, \tau_s)$ and *transfer*(doc_resp, τ'_n, *tag*(u)) $= (-, \tau'_{ci}, -, \tau'_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \overset{CK}{\longleftarrow}_{\tau_{ci}} u, N_0, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ P' & = & \langle K' \overset{CK'}{\longleftarrow}_{\tau'_{ci}} u, N_1, H', \lceil e \rfloor^{\tau'_s}_{@u}, [\,] \rangle \end{array}$$

To prove the desired conclusion, we need to show:

(a) $K \overset{CK}{\longleftarrow}_{\tau_{ci}} u \simeq_\ell^C K' \overset{CK'}{\longleftarrow}_{\tau'_{ci}} u$.

(b) $\lceil e \rfloor^{\tau_s}_{@u} \simeq_\ell^C \lceil e \rfloor^{\tau'_s}_{@u}$

We distinguish two subcases:

∗ if $C(\tau_n) \cap C(\ell) \neq \emptyset$ then we know by (2) and the definition of $\simeq_\ell^C$ that $\tau_n = \tau'_n$. Hence we also have $\tau_{ci} = \tau'_{ci}$ and $\tau_s = \tau'_s$. Point (a) follows directly from Lemma 4 and point (b) follows from the reflexivity of $\simeq_\ell^C$.

∗ if $C(\tau_n) \cap C(\ell) = \emptyset$ then by (2) we also know that $C(\tau'_n) \cap C(\ell) = \emptyset$. By the constraints we know that $C(\tau) \cap C(\ell) = \emptyset$ for all $\tau \in \{\tau_s, \tau'_s, \tau_{ci}, \tau'_{ci}\}$. Point (a) then follows from Lemma 5, (1) and transitivity and point (b) follows from the definition of $\simeq_\ell^C$.

– assume that rule (I-DocResp) is used on C, while C' fails using rule (I-Complete). This implies that $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq_\ell^C N'$ and $C(\tau_n) \cap C(\ell) = \emptyset$. Assuming that *transfer*(doc_resp, τ_n, *tag*(u)) $= (-, \tau_{ci}, -, \tau_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \overset{CK}{\longleftarrow}_{\tau_{ci}} u, N_0, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

To prove the desired conclusion we need to show:

(a) $K \overset{CK}{\longleftarrow}_{\tau_{ci}} u \simeq_\ell^C K'$

(b) $\lceil e \rfloor^{\tau_s}_{@u} \simeq_\ell^C \mathsf{wait}$

(c) $[\,] \approx_\ell \bullet$

By the constraints we know that $C(\tau) \cap C(\ell) = \emptyset$ for all $\tau \in \{\tau_s, \tau_{ci}, \}$. Point (a) then follows from Lemma 5, (1) and transitivity and point (b) follows from the definition of $\simeq_\ell^C$. Point (c) follows from the invisibility of $\bullet$ and using (S-Right) and (S-Empty).

– assume that rule (I-DocResp) is used on C', while C fails using rule (I-Complete). The case is analogous to the previous one;

– assume that rule (I-Complete) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

• if $i = \mathsf{doc\_redir}_n(u : CK, u')$, then $i' = \mathsf{doc\_redir}_n(u : CK', u')$ with *ck-erase*$_\ell^C(CK)$ = *ck-erase*$_\ell^C(CK')$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:

– assume that rule (I-DocRedir) is used on both C and C', then we know that $N = N_0 \uplus \{n^{\tau_n} : u\}$ and $N' = N_1 \uplus \{n^{\tau'_n} : u\}$ for some N_0, N_1 such that $N_0 \simeq_\ell^C N_1$. Assuming that *transfer*(doc_redir, τ_n, *tag*(u'), $-$) $= (\tau_m, \tau_{ci}, \tau_{co}, -)$, *transfer*(doc_redir, τ'_n, *tag*(u')) $= (\tau'_m, \tau'_{ci}, \tau'_{co}, -)$, $K_0 = K \overset{CK}{\longleftarrow}_{\tau_{ci}} u$ and $K_1 = K' \overset{CK'}{\longleftarrow}_{\tau'_{ci}} u$, we have

$$\begin{array}{lcl} P & = & \langle K_0, N_0 \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle \\ P' & = & \langle K_1, N_1 \uplus \{n^{\tau'_m} : u'\}, H', \mathsf{wait}, \mathsf{doc\_req}(u' : !K_1(u')^{\tau'_{co}}) \rangle \end{array}$$

To prove the desired conclusion, we need to show:

(a) $K_0 \simeq_\ell^C K_1$

(b) $N_0 \uplus \{n^{\tau_m} : u'\} \simeq_\ell^C N_1 \uplus \{n^{\tau'_m} : u'\}$

(c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \mathsf{doc\_req}(u' : !K_1(u')^{\tau'_{co}})$

We distinguish two subcases:

∗ if $C(\tau_n) \cap C(\ell) \neq \emptyset$ then we know by (2) and the definition of $\simeq_\ell^C$ that $\tau_n = \tau'_n$. Hence we also have $\tau_m = \tau'_m$, $\tau_{ci} = \tau'_{ci}$ and $\tau_{co} = \tau'_{co}$. Point (a) follows directly from Lemma 4 and point (b) follows from (2) and the definition of $\simeq_\ell^C$. To show point (c) we do an additional case distinction:

· if *evt_label*$(u') \not\sqsubseteq C(\ell)$ then $\neg$*rel*(doc_req$(u' : !K_0(u')^{\tau_{co}})$) and $\neg$*rel*(doc_req$(u' : !K_1(u')^{\tau_{co}})$), hence we conclude by applying rules (S-Left) and (S-Right).

· If *evt_label*$(u') \subseteq C(\ell)$ and *msg_label*$(u') \not\subseteq C(\ell)$ then the claim follows from the definition of $\sim_\ell^C$ and the rule (S-MATCH).
· If *evt_label*$(u') \subseteq C(\ell)$ and *msg_label*$(u') \subseteq C(\ell)$ then we get *msg_label*$(u') \subseteq C(\tau_{co})$ by the constraints of the transfer-function and the rule (T-DOCREDIR). This implies $C(\tau_{co}) \cap C(\ell) \neq \emptyset$ and hence the we know $!K_0(u')^{\tau_{co}} = !K_1(u')^{\tau_{co}}$ by Lemma 3. Point (c) then follows from the definition of $\sim_\ell$ and rule (S-MATCH).

∗ if $C(\tau_n) \cap C(\ell) = \emptyset$ then by (2) we know that also $C(\tau'_n) \cap C(\ell) = \emptyset$.
By the constraints of the transfer-function we get $C(\tau) \cap C(\ell)$ for all $\tau \in \{\tau_m, \tau'_m, \tau_{ci}, \tau'_{ci}\}$ by rule (T-DOCREDIR). Point (a) follows from Lemma 5, (1) and transitivity and point (b) follows from (2) and the definition of $\simeq_\ell^C$. The constraints also give us *evt_label*$(u') \cap C(\ell) = \emptyset$ and hence we know $\neg$*rel*(doc_req$(u' : !K_0(u')^{\tau_{co}})$) and $\neg$*rel*(doc_req$(u' : !K_1(u')^{\tau'_{co}})$). We can conclude (c) using rules (S-LEFT) and (S-RIGHT) and (S-EMPTY).

– assume that rule (I-DOCREDIR) is used on C, while C' fails using rule (I-COMPLETE). This implies that $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq_\ell^C N'$ and $C(\tau_n) \cap C(\ell) = \emptyset$. Assuming that *transfer*(doc_redir, τ_n, *tag*(u')) $= (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have:

$$\begin{array}{lcl} P & = & \langle K_0, N_0 \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}})\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

To prove the desired conclusion we need to show:
(a) $K_0 \simeq_\ell^C K'$
(b) $N_0 \uplus \{n^{\tau_m} : u'\} \simeq_\ell^C N'$
(c) doc_req$(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \bullet$

By the constraints of the transfer-function we get $C(\tau) \cap C(\ell)$ for all $\tau \in \{\tau_m, \tau_{ci}\}$ by rule (T-DOCREDIR). Point (a) follows from Lemma 5, (1) and transitivity and point (b) follows from (2) and the definition of $\simeq_\ell^C$. The constraints also give us *evt_label*$(u') \cap C(\ell) = \emptyset$ and hence we know $\neg$*rel*(doc_req$(u' : !K_0(u')^{\tau_{co}})$) and $\neg$*rel*$(\bullet)$. We can conclude (c) using rules (S-LEFT) and (S-RIGHT) and (S-EMPTY).

– assume that rule (I-DOCREDIR) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
– assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

• if $i =$ xhr_resp$_n(u : CK, v)$, then $i' =$ xhr_resp$_n(u : CK', v)$ with *ck-erase*$_\ell^C(CK) =$ *ck-erase*$_\ell^C(CK')$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:
– assume that rule (I-XHRRESP) is used on both C and C', then we know that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}$ and $H' = H_1 \uplus \{n^{\tau'_n} : (u, \lceil \lambda x.e' \rfloor_{@u''})\}$ for some H_0, H_1 such that $H_0 \simeq_\ell^C H_1$. Assuming that *transfer*(xhr_resp, τ_n, *tag*(u)) $= (-, \tau_{ci}, -, \tau_s)$ and *transfer*(xhr_resp, τ'_n, *tag*(u)) $= (-, \tau'_{ci}, -, \tau'_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \xleftarrow{CK}_{\tau_{ci}} u, N, H_0, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,]\rangle \\ P' & = & \langle K' \xleftarrow{CK'}_{\tau'_{ci}} u, N', H_1, \lceil e\{v/x\} \rfloor^{\tau'_s}_{@u''}, [\,]\rangle \end{array}$$

To prove the desired conclusion, we need to show:
(a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq_\ell^C K' \xleftarrow{CK'}_{\tau'_{ci}} u$.
(b) $\lceil e\{v/x\} \rfloor^{\tau_s}_{@u'} \simeq_\ell^C \lceil e'\{v/x\} \rfloor^{\tau'_s}_{@u''}$
We distinguish two subcases:

∗ if $C(\tau_n) \cap C(\ell) \neq \emptyset$ then we know by (3) and the definition of $\simeq_\ell^C$ that $\tau_n = \tau'_n$, $u' = u''$ and $e = e'$. Hence we also have $\tau_{ci} = \tau'_{ci}$ and $\tau_s = \tau'_s$. Point (a) follows directly from Lemma 4 and point (b) follows from the definition of $\simeq_\ell^C$.
∗ if $C(\tau_n) \cap C(\ell) = \emptyset$ then by (3) we know that also $C(\tau_n) \cap C(\ell) = \emptyset$. By the constraints we know that $C(\tau) \cap C(\ell) = \emptyset$ for all $\tau \in \{\tau_s, \tau'_s, \tau_{ci}, \tau'_{ci}\}$. Point (a) then follows from Lemma 5, (1) and transitivity and point (b) follows from the definition of $\simeq_\ell^C$.

– assume that rule (I-XHRRESP) is used on C, while C' fails using rule (I-COMPLETE). This implies that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}$ for some H_0 with $H_0 \simeq_\ell^C H'$ and $C(\tau_n) \cap C(\ell) = \emptyset$. Assuming that *transfer*(xhr_resp, τ_n, $-$) $= (-, \tau_{ci}, -, \tau_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \xleftarrow{CK}_{\tau_{ci}} u, N, H_0, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,]\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

To prove the desired conclusion we need to show:

(a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq^C_\ell K'$
(b) $\lceil e\{v/x\} \rfloor^{\tau_s}_{@u'} \simeq^C_\ell \mathsf{wait}$
(c) $[\,] \approx_\ell \bullet$

By the constraints we know that $C(\tau) \cap C(\ell) = \emptyset$ for all $\tau \in \{\tau_s, \tau_{ci}\}$. Point (a) then follows from Lemma 5, (1) and transitivity and point (b) follows from the definition of $\simeq^C_\ell$. Point (c) follows from $\neg \mathit{rel}(\bullet)$ and using rules (S-RIGHT) and (S-EMPTY).

- assume that rule (I-XHRRESP) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
- assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{rcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

- if $i = \mathsf{xhr\_redir}_n(u : CK, u')$, then $i' = \mathsf{xhr\_redir}_n(u : CK', u')$ with $\mathit{ck\text{-}erase}^C_\ell(CK) = \mathit{ck\text{-}erase}^C_\ell(CK')$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:
  - assume that rule (I-XHRREDIR) is used on both C and C', then we know that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}$ and $H' = H_1 \uplus \{n^{\tau'_n} : (u, \lceil \lambda x.e' \rfloor_{@u'''})\}$ for some H_0, H_1 such that $H_0 \simeq^C_\ell H_1$. Assuming that $\mathit{transfer}(\mathsf{xhr\_redir}, \tau_n, \mathit{tag}(u'), -) = (\tau_m, \tau_{ci}, \tau_{co}, -)$, $\mathit{transfer}(\mathsf{xhr\_redir}, \tau'_n, \mathit{tag}(u')) = (\tau'_m, \tau'_{ci}, \tau'_{co}, -)$, $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ and $K_1 = K' \xleftarrow{CK'}_{\tau'_{ci}} u$, we have

$$\begin{array}{rcl} P & = & \langle K_0, N, H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle \\ P' & = & \langle K_1, N', H_1 \uplus \{n^{\tau'_m} : (u', \lceil \lambda x.e' \rfloor_{@u'''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}}) \rangle \end{array}$$

    To prove the desired conclusion, we need to show:

    (a) $K_0 \simeq^C_\ell K_1$
    (b) $H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\} \simeq^C_\ell H_1 \uplus \{n^{\tau'_m} : (u', \lceil \lambda x.e' \rfloor_{@u'''})\}$
    (c) $\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \sim_\ell \mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}})$

    We distinguish two subcases:

    * if $C(\tau_n) \cap C(\ell) \neq \emptyset$ then we know by (3) and the definition of erase^C_ℓ that $\tau_n = \tau'_n$, $e = e'$ and $u'' = u'''$. Hence we also have $\tau_m = \tau'_m$, $\tau_{ci} = \tau'_{ci}$ and $\tau_{co} = \tau'_{co}$. Point (a) follows directly from Lemma 4 and point (b) follows from (3) and the definition of $\simeq^C_\ell$. To show point (c) we do an additional case distinction:
      · if $\mathit{evt\_label}(u') \not\subseteq C(\ell)$ then $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}))$ and $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_1(u')^{\tau_{co}}))$, hence we conclude by applying rules (S-LEFT) and (S-RIGHT).
      · If $\mathit{evt\_label}(u') \subseteq C(\ell)$ and $\mathit{msg\_label}(u') \not\subseteq C(\ell)$ then the claim follows from the definition of $\sim^C_\ell$ and the rule (S-MATCH).
      · If $\mathit{evt\_label}(u') \subseteq C(\ell)$ and $\mathit{msg\_label}(u') \subseteq C(\ell)$ then we get $\mathit{msg\_label}(u') \subseteq C(\tau_{co})$ by the constraints of the transfer-function and the rule (T-XHRREDIR). This implies $C(\tau_{co}) \cap C(\ell) \neq \emptyset$ and hence the we know $!K_0(u')^{\tau_{co}} = !K_1(u')^{\tau_{co}}$ by Lemma 3. Point (c) then follows from the definition of $\sim_\ell$ and rule (S-MATCH).
    * if $C(\tau_n) \cap C(\ell) = \emptyset$ then we know by (3) and the definition of $\simeq^C_\ell$ that $C(\tau_n) \cap C(\ell) = \emptyset$. By the constraints of the transfer-function we get $C(\tau) \cap C(\ell)$ for all $\tau \in \{\tau_m, \tau'_m, \tau_{ci}, \tau'_{ci}\}$ by rule (T-XHRREDIR). Point (a) follows from Lemma 5, (1) and transitivity and point (b) follows from (3) and the definition of $\simeq^C_\ell$. The constraints also give us $\mathit{evt\_label}(u') \cap C(\ell) = \emptyset$ and hence we know $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}))$ and $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}}))$. We can conclude (c) using rules (S-LEFT) and (S-RIGHT) and (S-EMPTY).
  - assume that rule (I-XHRREDIR) is used on C, while C' fails using rule (I-COMPLETE). This implies that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}$ for some H_0 with $H_0 \simeq^C_\ell H'$ and $C(\tau_n) \cap C(\ell) = \emptyset$. Assuming that $\mathit{transfer}(\mathsf{xhr\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have

$$\begin{array}{rcl} P & = & \langle K_0, N, H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

    To prove the desired conclusion we need to show:

    (a) $K_0 \simeq^C_\ell K'$
    (b) $H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\} \simeq^C_\ell H'$
    (c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \bullet$

    By the constraints of the transfer-function we get $C(\tau) \cap C(\ell)$ for all $\tau \in \{\tau_m, \tau_{ci}\}$ by rule (T-XHRREDIR). Point (a) follows from Lemma 5, (1) and transitivity and point (b) follows from (3) and the definition of $\simeq^C_\ell$. The constraints

also give us $\mathit{evt\_label}(u') \cap C(\ell) = \emptyset$ and hence we know $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}))$ and $\neg \mathit{rel}(\bullet)$. We can conclude (c) using rules (S-LEFT) and (S-RIGHT) and (S-EMPTY).

– assume that rule (I-XHRREDIR) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
– assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

□

Lemma 7. *$\mathcal{R}_\ell^C$ satisfies the third condition of Definition 11.*

Proof. Let $C = \langle K, N, H, \mathsf{wait}, [\,] \rangle$ and $C' = \langle K', N', H', \mathsf{wait}, [\,] \rangle$ with $C \, \mathcal{R}_\ell^C \, C'$. By definition of $\mathcal{R}_\ell^C$, we have:
(1) $K \simeq_\ell^C K'$;
(2) $N \simeq_\ell^C N'$;
(3) $H \simeq_\ell^C H'$.

Assume that $C \xrightarrow{i} P$ with $\neg \mathit{rel}_\ell(i)$. The only case when $\neg \mathit{rel}_\ell(i)$ holds true is when $i = \mathsf{load}(u)$ with $\Gamma_C(u) \cap C(\ell) = \emptyset$. By the reduction rules, assuming that $\mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) = (\tau_n, -, \tau_{co}, -)$, we then have:

$$P = \langle K, N \uplus \{n^{\tau_n} : u\}, H, \mathsf{wait}, \mathsf{doc\_req}(u : !K(u)^{\tau_{co}}) \rangle$$

To prove the desired conclusion, we need to show:
(a) $N \uplus \{n^{\tau_n} : u\} \simeq_\ell^C N'$
(b) $\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}) \approx_\ell [\,]$

By the constraints we know $C(\tau_n) \cap C(\ell) = \emptyset$ and point (a) follows from (2) and the definition of $\simeq_\ell^C$. By the constraints we also get $\mathit{evt\_label}(u) \cap C(\ell) = \emptyset$ and hence $\neg \mathit{rel}(\mathsf{doc\_req}(u : !K(u)^{\tau_{co}}))$. We can conclude (b) using (S-LEFT) and (S-EMPTY). □

Lemma 8 (Invisible Scripts)**.** *Let $P = \langle K, N, H, \lceil e \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and let $P \, \mathcal{R}_\ell^C \, Q$ for some state Q. If $C(\tau_s) \cap C(\ell) = \emptyset$ and $P \xrightarrow{o} Q'$ for some o, Q', then $\neg \mathit{rel}_\ell(o)$ and $Q' \, \mathcal{R}_\ell^C \, Q$.*

Proof. Let $Q = \langle K', N', H', T', O' \rangle$. By definition of $\mathcal{R}_\ell^C$ we have
(1) $K \simeq_\ell^C K'$
(2) $N \simeq_\ell^C N'$
(3) $H \simeq_\ell^C H'$
(4) $\lceil e \rfloor_{@u}^{\tau_s} \simeq_\ell^C T'$
(5) $[\,] \simeq_\ell^C O'$

We do an induction on the term structure of e:
Induction Hypothesis: The claim holds for all $P = \langle K, N, H, \lceil e' \rfloor_{@u}^{\tau_s}, [\,] \rangle$ where e' is a subterm of e.

In the following subcases we will say that the claim follows trivially if the claim follows directly from $(1) - (5)$ and the fact that $\lceil e' \rfloor_{@u}^{\tau_s} \simeq_\ell^C T'$ for all e' because $C(\tau_s) \cap C(\ell) = \emptyset$.

- If $e = \lambda x.e'\ v$ then (O-APP) is used and $Q' = \langle K, N, H, \lceil e'\{v/x\} \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
- If $e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ then we distinguish two cases:
  – If there exist K^*, N^*, H^*, e^* and o^* with $\langle K, N, H, \lceil e_1 \rfloor_{@u}^{\tau_s}, [\,] \rangle \xrightarrow{o^*} \langle K^*, N^*, H^*, \lceil e^* \rfloor_{@u}^{\tau_s}, [\,] \rangle$ then (O-LETCTX) is used and $Q' = \langle K^*, N^*, H^*, \lceil \mathsf{let}\ x = e^*\ \mathsf{in}\ e_2 \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and $o = o^*$. As e_1 is a subterm of e we get $K^* \simeq_\ell^C K'$, $N^* \simeq_\ell^C N'$, $H^* \simeq_\ell^C H'$ and $\neg \mathit{rel}(o^*)$ by the induction hypothesis. The claim then follows directly.
  – Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
- If $e = \mathsf{let}\ x = v\ \mathsf{in}\ e'$ then (O-LET) is used and $Q' = \langle K, N, H, \lceil e'\{v/x\} \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
- If $e = \mathsf{get\text{-}ck}(k)$ then we distinguish two cases:
  – If there exist τ, v with $\mathsf{ck}(k, v)^\tau \in !K(u)^{\tau_s}$ then rule (O-GETCOOKIE) is used and $Q' = \langle K, N, H, \lceil v \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
  – Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
- If $e = \mathsf{set\text{-}ck}(k, v)$ then rule (O-SETCOOKIE) is used. Let $CK = \{\mathsf{ck}(k, v)^\tau \mid \tau = \kappa(\mathit{host}(u), k)\}$, then $Q' = \langle K \xleftarrow{CK}_{\tau_s} u, N, H, \lceil \mathit{unit} \rfloor_{@u}^{\tau_s}, [\,] \rangle$ and $o = \bullet$. Using (1) and Lemma 5 we get $K \xleftarrow{CK}_{\tau_s} u \simeq_\ell^C K'$. The claim then follows trivially.
- If $e = \mathsf{xhr}(u', \lambda x.e')$ then we distinguish two cases:

- If $\mathit{transfer}(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)$ for some τ_n, τ_{co} then rule (O-XHR) is used and $Q' = \langle K, N, H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \mathsf{xhr\_req}(u' : !K(u')^{\tau_{co}})$.
  By the constraints we get $C(\tau_n) \cap C(\ell) = \emptyset$ and $\mathit{evt\_label}(u) \cap C(\ell) = \emptyset$. We get $H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\} \simeq^C_\ell H'$ directly from the definition of $\simeq^C_\ell$ and (3). Since we have $\neg \mathit{rel}(o)$, the claim then follows trivially. .
- Otherwise rule (O-COMPLETE) is used and the claim follows trivially.

- For all other forms of e (O-COMPLETE) is used and the claim follows trivially.

□

Lemma 9. *$\mathcal{R}^C_\ell$ satisfies the fourth condition of Definition 11.*

Proof. Let $P = \langle K, N, H, T, O \rangle$ and $C = \langle K', N', H', \mathsf{wait}, [\,] \rangle$, where either $T \neq \mathsf{wait}$ or $O = o :: O'$ for some o, O' by definition of producer state. Assume that $P \, \mathcal{R}^C_\ell \, C$, by definition we have:

(1) $K \simeq^C_\ell K'$
(2) $N \simeq^C_\ell N'$
(3) $H \simeq^C_\ell H'$
(4) $T \simeq^C_\ell \mathsf{wait}$
(5) $O \approx_\ell [\,]$

We now distinguish two sub-cases:

- if $O = o :: O'$ for some o, O', the only available reduction rule for P is rule (O-FLUSH), hence:

$$P \xrightarrow{o} \langle K, N, H, T, O' \rangle.$$

  Using (5) and the definition of stream similarity, we have $\neg \mathit{rel}_\ell(o)$ and $O' \approx_\ell [\,]$, which is enough to close the case;
- otherwise, assume without loss of generality that $O = [\,]$, then $T = \lceil e \rfloor^{\tau_s}_{@u}$ for some e, u, τ_s. We observe that point (4) implies that $C(\tau_s) \cap C(\ell) = \emptyset$, hence we conclude by Lemma 8;

□

Lemma 10. *$\mathcal{R}^C_\ell$ satisfies the fifth condition of Definition 11.*

Proof. Let $P = \langle K, N, H, T, O \rangle$ and $P' = \langle K', N', H', T', O' \rangle$ with $P \, \mathcal{R}^C_\ell \, P'$. By definition of $\mathcal{R}^C_\ell$, we have:

1) $K \simeq^C_\ell K'$;
2) $N \simeq^C_\ell N'$;
3) $H \simeq^C_\ell H'$;
4) $T \simeq^C_\ell T'$;
5) $O \approx_\ell O'$.

We distinguish four sub-cases:

- if $O = o :: O_0$ and $O' = o' :: O_1$, the only available reduction rule is (O-FLUSH) for both P and P'. We distinguish three sub-cases:
  - If $\neg \mathit{rel}(o)$ then $O \approx_\ell O'$ is shown by the rule (S-LEFT) and we know $O_0 \approx_\ell O'$. We have $P \xrightarrow{o} Q$ for $Q = \langle K, N, H, T, O_0 \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case b).
  - If $\mathit{rel}(o)$ and $\neg \mathit{rel}(o')$ then $O \approx_\ell O'$ is shown by the rule (S-RIGHT) and we know $O \approx_\ell O_1$. We have $P' \xrightarrow{o} Q'$ for $Q' = \langle K', N', H', T', O_1 \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case c).
  - If $\mathit{rel}(o)$ and $\mathit{rel}(o')$ then $O \approx_\ell O'$ is shown by the rule (S-MATCH) and we know $O_0 \approx_\ell O_1$ and $o \sim_\ell o'$. By the rule (O-FLUSH) we have $P \xrightarrow{o} Q$ and $P' \xrightarrow{o'} Q'$ for $Q = \langle K, N, H, T, O_1 \rangle$ and $Q' = \langle K', N', H', T', O_0 \rangle$. Using $(1), (2), (3)$ and (4), we can match case a).
- if $O = o :: O''$ and $O' = [\,]$, we know that $O \approx_\ell O'$ can only be shown using rule (S-LEFT), hence we have $\neg \mathit{rel}(o)$ and $O'' \approx_\ell [\,]$. We have $P \xrightarrow{o} Q$ for $Q = \langle K, N, H, T, O'' \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case b).
- if $O = [\,]$ and $O' = o' :: O''$, we know that $O \approx_\ell O'$ can only be shown using rule (S-RIGHT), hence we have $\neg \mathit{rel}(o')$ and $[\,] \approx_\ell O''$. We have $P' \xrightarrow{o'} Q'$ for $Q' = \langle K', N', H', T', O'' \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case c).
- if $O = O' = [\,]$, then by definition of producer state we have $T = \lceil e \rfloor^{\tau_s}_{@u}$ and $T' = \lceil e' \rfloor^{\tau'_s}_{@u'}$. We distinguish two sub-cases:
  - If $C(\tau_s) \cap C(\ell) = \emptyset$ and $P \xrightarrow{o} Q$ then we get $\neg \mathit{rel}(o)$ and $Q \, \mathcal{R}^C_\ell \, P'$ by Lemma 8 and we can match case b).
  - If $C(\tau_s) \cap C(\ell) \neq \emptyset$ then point (4) implies $e = e'$, $u = u'$ and $\tau_s = \tau'_s$.
    Assume $P \xrightarrow{o} Q$ and $P' \xrightarrow{o'} Q'$. We show the following stronger claim, that is directly implying a): $o \sim_\ell o'$ and $Q \, \mathcal{R}^C_\ell \, Q'$ and $P \overset{o}{\mapsto} Q \iff P \overset{o'}{\mapsto} Q'$

We do an induction on the term structure of e:
Induction Hypothesis: The claim holds for all $P = \langle K, N, H, \lceil e'' \rfloor^{\tau_s}_{@u}, O\rangle$ and $P' = \langle K', N', H', \lceil e'' \rfloor^{\tau_s}_{@u}, O'\rangle$ where e'' is a subterm of e.

$*$ If $e = \lambda x.e_1\, v$ then (O-APP) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ Q' & = & \langle K', N', H', \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$*$ If $e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ then we distinguish two cases:

$\cdot$ If there exist K^*, N^*, H^*, e^* and o^* with $\langle K, N, H, \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,]\rangle \xrightarrow{o^*} \langle K^*, N^*, H^*, \lceil e^* \rfloor^{\tau_s}_{@u}, [\,]\rangle$ then as e_1 is a subterm of e by the induction hypothesis we know that there exist $K^{**}, N^{**}, H^{**}, e^{**}$ and o^{**} with $\langle K', N', H', \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,]\rangle \xrightarrow{o^{**}} \langle K^{**}, N^{**}, H^{**}, \lceil e^{**} \rfloor^{\tau_s}_{@u}, [\,]\rangle$ such that $o^* \sim_\ell o^{**}$, $K^* \simeq^C_\ell K^{**}$, $N^* \simeq^C_\ell N^{**}$, $H^* \simeq^C_\ell H^{**}$ and $\lceil e^* \rfloor^{\tau_s}_{@u} \simeq^C_\ell \lceil e^{**} \rfloor^{\tau_s}_{@u}$, hence $e^* = e^{**}$. By rule (O-LETCTX) we get

$$\begin{array}{lcl} Q & = & \langle K^*, N^*, H^*, \lceil \mathsf{let}\ x = e^*\ \mathsf{in}\ e_2 \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ Q' & = & \langle K^{**}, N^{**}, H^{**}, \lceil \mathsf{let}\ x = e^*\ \mathsf{in}\ e_2 \rfloor^{\tau_s}_{@u}, [\,]\rangle \end{array}$$

and $o = o^*$ and $o' = o^{**}$ and the claim follows.

$\cdot$ Otherwise we know by the induction hypothesis that both P and P' use rule (O-COMPLETE). We get

$$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,]\rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$*$ If $e = \mathsf{let}\ x = v\ \mathsf{in}\ e_1$ then (O-LET) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ Q' & = & \langle K', N', H', \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$*$ If $e = \mathsf{get\text{-}ck}(k)$ then we distinguish two cases:

$\cdot$ If there exist τ, v with $\mathsf{ck}(k, v)^\tau \in\, !K(u)^{\tau_s}$ then we know that $\mathsf{ck}(k, v)^\tau \in\, !K'(u)^{\tau_s}$ by Lemma 3. Hence rule (O-GETCOOKIE) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil v \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ Q' & = & \langle K', N', H', \lceil v \rfloor^{\tau_s}_{@u}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$\cdot$ If there exist no τ, v with $\mathsf{ck}(k, v)^\tau \in\, !K(u)^{\tau_s}$ then we know that there are no τ', v' with $\mathsf{ck}(k, v')^{\tau'} \in\, !K(u)^{\tau_s}$ by Lemma 3. Hence rule (O-COMPLETE) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,]\rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$*$ If $e = \mathsf{set\text{-}ck}(k, v)$ then rule (O-SETCOOKIE) is used on . Let $CK = \{\mathsf{ck}(k, v)^\tau \mid \tau = \kappa(\mathit{host}(u), k)\}$, then

$$\begin{array}{lcl} Q & = & \langle K \xleftarrow{CK}_{\tau_s} u, N, H, \mathsf{wait}, [\,]\rangle \\ Q' & = & \langle K' \xleftarrow{CK}_{\tau_s} u, N', H', \mathsf{wait}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. We get $K \xleftarrow{CK}_{\tau_s} u \simeq^C_\ell K' \xleftarrow{CK}_{\tau_s} u$ using (1) and Lemma 4. The claim then follows using $(2), (3)$ and reflexivity.

$*$ If $e = \mathsf{xhr}(u', \lambda x.e')$ then we distinguish two cases:

$\cdot$ If $\mathit{transfer}(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)$ for some τ_n, τ_{co} then rule (O-XHR) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ Q' & = & \langle K', N', H' \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ o & = & \mathsf{xhr\_req}(u' : !K(u')^{\tau_{co}}) \\ o' & = & \mathsf{xhr\_req}(u' : !K'(u')^{\tau_{co}}) \end{array}$$

We get $H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\} \simeq^C_\ell H' \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}$ using (3) and the definition of $\simeq^C_\ell$.
We distinguish two subcases:

(a) If $C(\tau_{co}) \cap C(\ell) \neq \emptyset$ then we get $!K(u')^{\tau_{co}} = !K'(u')^{\tau_{co}}$ and hence $o = o'$ by Lemma 3. The claim follows using $(1), (2)$ and reflexivity.

(b) If $C(\tau_{co}) \cap C(\ell) = \emptyset$ then we know *msg_label*$(u) \subseteq C(l_{co})$ by the constraints of the transfer-function and can hence infer *msg_label*$(u) \cap C(\ell) = \emptyset$. We then get $o \sim_\ell o'$ by the definition of $\sim_\ell$ and the claim follows using $(1), (2)$ and reflexivity.

· Otherwise rule (O-COMPLETE) is used on P and P' and

$$\begin{array}{rcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,]\rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

∗ For all other forms of e the rule (O-COMPLETE) is used on P and P' and

$$\begin{array}{rcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,]\rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,]\rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

□

G. Proof of Integrity

Definition 14 (High Equivalence)**.** *We define an* erasure *operator* $erase^I_\ell(\cdot)$ *on different browser data structures:*

- $erase^I_\ell(K)$ *is the map with domain* $dom(K)$ *such that for all* $d \in dom(K) : (erase^I_\ell(K))(d) = ck\text{-}erase^I_\ell(K(d))$
- $erase^I_\ell(N)$ *is the map obtained from* N *by erasing all the entries* $n^\tau : u$ *such that* $I(\tau) \cap I(\ell) \neq \emptyset$*;*
- $erase^I_\ell(H)$ *is the map obtained from* H *by erasing all the entries* $n^\tau : (u, \lceil \lambda x.e \rfloor_{@u'})$*' such that* $I(\tau) \cap I(\ell) \neq \emptyset$*;*
- $erase^I_\ell(T) = \mathsf{wait}$ *whenever* $T = \lceil e \rfloor^\tau_{@u}$ *with* $I(\tau) \cap C(\ell) \neq \emptyset$*, while* $erase^I_\ell(T) = T$ *otherwise.*

We then define a binary high equivalence *relation* $\simeq^I_\ell$ *between data structures coinciding after applying the erasure* $erase^I_\ell(\cdot)$*.*

Definition 15 (Well-formedness)**.** *Let* $Q = \langle K, N, H, T, O\rangle$*. Then we say that* Q *is* well-formed*, if we have*

1) if $\{n^{\tau_n} : u\} \subseteq N$ *then* *msg_label*$(u) \subseteq I(\tau_n)$
2) if $\{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\} \subseteq H$ *then* *msg_label*$(u) \subseteq I(\tau_n)$

Definition 16 (Candidate for Integrity)**.** *Let* $Q = \langle K, N, H, T, O\rangle$ *and* $Q' = \langle K', N', H', T', O'\rangle$*, we write* $Q \,\mathcal{R}^I_\ell\, Q'$ *if and only if: (1)* $K \simeq^I_\ell K'$*; (2)* $N \simeq^I_\ell N'$*; (3)* $H \simeq^I_\ell H'$*; (4)* $T \simeq^I_\ell T'$*; (5)* $O \approx_\ell O'$*; (6); Q and* Q' *are well-formed.*

Lemma 11. *For the initial state* C_0 *we have* $C_0 \,\mathcal{R}^I_\ell\, C_0$

Proof. (1) - (5) follow directly from the reflexivity of $\simeq^I_\ell$ and $\approx_\ell$. Well-formedness for the initial state is trivial since for $C_0 = \langle K, N, H, T, O\rangle$, we have $N = \{\}$ and $M = \{\}$. □

Lemma 12 (invariant)**.** *If* Q *is well-formed and* $Q \xrightarrow{a} Q'$ *then* Q' *is well-formed.*

Proof. Let $Q = \langle K, N, H, T, O\rangle$ and $Q' = \langle K', N', H', T', O'\rangle$. We do a case distinction on the form of Q:

- If Q is a consumer state then we do a case distinction on the used reduction rule.
  - If rule (I-LOAD) is used, Then we have $N' = N \uplus \{n^{\tau_n} : u\}$ with *msg_label*$(u) \subseteq I(\tau_n)$ by rule (T-LOAD), so $1)$ holds. Because of $H = H'$ and $T = T'$ we can also conclude $2)$.
  - If rule (I-DOCRESP) is used, then we have $N' \subseteq N'$ and $H' = H$ and we can conclude $1)$ and $2)$.
  - If rule (I-DOCREDIR) is used, then we have $N' = (N \setminus \{n^{\tau_n} : u\}) \uplus \{n^{\tau_m} : u'\}$ with *msg_label*$(u') \subseteq I(\tau_m)$ by rule (T-DOCREDIR), so $1)$ follows from $1)$. Because of $H = H'$ we can also conclude $2)$.
  - If rule (I-XHRRESP) is used, then we $H' \subseteq H$ and $N' = N$ and we can conclude $1)$ and $2)$
  - If rule (I-XHRREDIR) is used, then we have $H' = (H \setminus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}) \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}$ with *msg_label*$(u') \subseteq I(\tau_m)$ by rule (T-XHRREDIR) and $2)$ follows from $2)$. Because of $N = N'$ we can also conclude $1)$.
  - If rule (I-COMPLETE) is used, then we have $N' = N$ and $H' = H$, so $1)$ and $2)$ hold.
- If Q is a producer sate, then we have $O \neq [\,]$ or $T \neq \mathsf{wait}$. We perform a case distinction on these two possibilities.
  - If $O = o :: O''$ then rule (O-FLUSH) is used and $N = N'$ and $H = H'$, so $1)$ and $2)$ hold.
  - if $O = [\,]$ and $T = \lceil e \rfloor^\tau_{@u}$ then we do an induction on the term structure of e.
    Induction Hypothesis: The claim holds for all $Q = \langle K, N, H, \lceil e' \rfloor^\tau_{@u}, O\rangle$ where e' is a subterm of e.
    ∗ If $e = \lambda x.e'\, v$ then (O-APP) and $N' = N$ and $H = H'$. The claim follows trivially.
    ∗ If $e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ then we distinguish two cases:

· If there exist K^*, N^*, H^*, e^* and o^* with $\langle K, N, H, \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,]\rangle \xrightarrow{o^*} \langle K^*, N^*, H^*, \lceil e^* \rfloor^{\tau_s}_{@u}, [\,]\rangle$ then (O-LETCTX) is used and $N' = N^*$ and $H' = H^*$. We get the claim by the induction hypothesis.
· Otherwise rule (O-COMPLETE) is used and the claim follows trivially.

∗ If $e = \mathsf{let}\ x = v\ \mathsf{in}\ e'$ then (O-LET) is used and $N' = N$ and $H = H'$. The claim follows trivially.
∗ If $e = \mathsf{get\text{-}ck}(k)$ then we distinguish two cases:
· If there exist τ, v with $\mathsf{ck}(k, v)^\tau \in\, !K(u)^{\tau_s}$ then rule (O-GETCOOKIE) is used and $N' = N$ and $H = H'$. The claim follows trivially.
· Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
∗ If $e = \mathsf{set\text{-}ck}(k, v)$ then rule (O-SETCOOKIE) is used and $N' = N$ and $H = H'$. The claim follows trivially.
∗ If $e = \mathsf{xhr}(u', \lambda x.e')$ then we distinguish two cases:
· If $\mathit{transfer}(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)$ for some τ_n, τ_{co} then rule (O-XHR) is used and $H' = H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}$
By the constraints we get $\mathit{msg\_label}(u') \subseteq I(\tau_n)$ and we get 2). Because of $N' = N$ we also have 1)
· Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
∗ For all other forms of e (O-COMPLETE) is used and the claim follows trivially.

□

Lemma 13. *$\mathcal{R}^I_\ell$ satisfies the first condition of Definition 11.*

Proof. A straightforward syntactic check on the definition of $\mathcal{R}^I_\ell$. □

Lemma 14 (Equality of high-integrity cookies)**.** *Let $K \simeq^I_\ell K'$. Then we have $\mathit{ck\text{-}erase}^I_\ell(!K(u)^\tau) = \mathit{ck\text{-}erase}^I_\ell(!K'(u)^\tau)$ for all u, τ.*

Proof. We only show $\mathit{ck\text{-}erase}^I_\ell(!K(u)^\tau) \subseteq \mathit{ck\text{-}erase}^I_\ell(!K'(u)^\tau)$, the proof of the other direction is analogous.

Let $\mathsf{ck}(k, v)^{\tau'} \in \mathit{ck\text{-}erase}^I_\ell(!K(u)^\tau)$. Then we know that $\mathsf{ck}(k, v)^{\tau'} \in K(d)$ with $I(\tau') \cap I(\ell) = \emptyset$ and $\mathit{transfer}(\mathsf{get}, \tau, \tau')\downarrow$. By the definition of $\simeq^I_\ell$ we know that $\mathsf{ck}(k, v)^{\tau'} \in K'(d)$ and because of $I(\tau') \cap I(\ell) = \emptyset$ and $\mathit{transfer}(\mathsf{get}, \tau, \tau')\downarrow$ we know $\mathsf{ck}(k, v)^\tau \in \mathit{ck\text{-}erase}^I_\ell(!K'(u)^{\tau'})$. □

Lemma 15 (High Equivalence after Cookie Updates)**.** *Let $\Gamma \vdash$ transfer and $K \simeq^I_\ell K'$. Then we have $K \xleftarrow{CK}_\tau u \simeq^I_\ell K' \xleftarrow{CK}_\tau u$ for all CK, u, τ.*

Proof. We have to show $\mathit{erase}^I_\ell((K \xleftarrow{CK}_\tau u)(d)) = \mathit{erase}^I_\ell((K' \xleftarrow{CK}_\tau u)(d))$ for all domains d. We only show $\mathit{erase}^I_\ell((K \xleftarrow{CK}_\tau u)(d)) \subseteq \mathit{erase}^I_\ell((K' \xleftarrow{CK}_\tau u)(d))$, the proof for the other direction is analogous. We distinguish two cases:

- If $d \neq \mathit{host}(u)$ then $(K \xleftarrow{CK}_\tau u)(d) = K(d)$ and $(K' \xleftarrow{CK}_\tau u)(d) = K'(d)$ and the claim follows directly from $K \simeq^I_\ell K'$.
- If $d = \mathit{host}(u)$ then assume that $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^I_\ell((K \xleftarrow{CK}_\tau u)(d)$. This implies that $I(\tau_1) \cap I(\ell) = \emptyset$ and $\mathsf{ck}(k_1, v_1)^{\tau_1} \in K \xleftarrow{CK}_\tau u$. We do a case distinction following the definition of $K \xleftarrow{CK}_\tau u$.
  - If $\mathsf{ck}(k_1, v_1)^{\tau_1} \in CK$ and $\mathit{transfer}(\mathsf{set}, \tau, \tau_1)\downarrow$ then we can immediately conclude that $\mathsf{ck}(k_1, v_1)^{\tau_1} \in CK'$ and hence $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^I_\ell((K' \xleftarrow{CK}_\tau u)(d))$.
  - If $\mathsf{ck}(k_1, v_1)^{\tau_1} \in K(d)$ and for all $\mathsf{ck}(k_2, v_2)^{\tau_2} \in CK$ we have $k_2 \neq k_1$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$ then let $\mathsf{ck}(k_3, v_3)^{\tau_3} \in CK'$. We have to show $k_3 \neq k_1$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_3)\uparrow$. We distinguish two cases:
    ∗ If $k_3 \neq k_1$ then the claim follows trivially.
    ∗ if $k_3 = k_1$ then we observe that $\tau_3 = \kappa(d, k_3) = \kappa(d, k_1) = \tau_1$. This implies $I(\tau_3) \cap I(\ell) = \emptyset$ and hence $\mathsf{ck}(k_3, v_3)^{\tau_3} \in CK$ because of $\mathit{ck\text{-}erase}^I_\ell(CK) = \mathit{ck\text{-}erase}^I_\ell(CK')$. We then know $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$.

□

Lemma 16 (Low Integrity Updates)**.** *Let $\Gamma \vdash$ transfer and $I(\tau) \cap I(\ell) \neq \emptyset$. Then we have $K \simeq^I_\ell K \xleftarrow{CK}_\tau u$ for all CK, u.*

Proof. We show both directions of set inclusions:

- Assume that $\mathsf{ck}(k, v)^{\tau'} \in \mathit{erase}^I_\ell((K \xleftarrow{CK}_\tau u)(d))$. This implies that $I(\tau') \cap I(\ell) = \emptyset$ and $\mathsf{ck}(k, v)^{\tau'} \in (K \xleftarrow{CK}_\tau u)(d)$. We can deduce that $I(\tau) \not\subseteq I(\tau')$, hence we know $\mathit{transfer}(\mathsf{set}, \tau, \tau')\uparrow$. By the definition of $(K \xleftarrow{CK}_\tau u)(d)$ we then know that $\mathsf{ck}(k, v)^{\tau'} \in K(d)$. Because of $I(\tau') \cap I(\ell) = \emptyset$ we get $\mathsf{ck}(k, v)^{\tau'} \in \mathit{erase}^I_\ell(K(d))$.
- Assume that $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^I_\ell(K(d))$. This implies that $I(\tau_1) \cap I(\ell) = \emptyset$ and $\mathsf{ck}(k_1, v_1)^{\tau_1} \in K(d)$. To show that $\mathsf{ck}(k_1, v_1)^{\tau_1} \in \mathit{erase}^I_\ell((K \xleftarrow{CK}_\tau u)(d))$ it suffices to show that for all we have $\mathsf{ck}(k_2, v_2)^{\tau_2} \in CK$ $k_1 \neq k_2$ or $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$. We distinguish two cases:
  - If $k_2 \neq k_1$ then the claim follows trivially

– If $k_2 = k_1$ then we observe that $\tau_2 = \kappa(\mathit{host}(u), k_2) = \kappa(\mathit{host}(u), k_1) = \tau_1$. As we know $I(\tau_1) \not\subseteq I(\tau)$, we can conclude $\mathit{transfer}(\mathsf{set}, \tau, \tau_2)\uparrow$

□

Lemma 17. *$\mathcal{R}^I_\ell$ satisfies the second condition of Definition 11.*

Proof. Let $C = \langle K, N, H, \mathsf{wait}, [\,]\rangle$ and $C' = \langle K', N', H', \mathsf{wait}, [\,]\rangle$ with $C\,\mathcal{R}^I_\ell\,C'$. By definition of $\mathcal{R}^C_\ell$, we have:
(1) $K \simeq^I_\ell K'$;
(2) $N \simeq^I_\ell N'$;
(3) $H \simeq^I_\ell H'$;
(4) C and C' are well-formed.

Assume that $C \xrightarrow{i} P$ and $C' \xrightarrow{i'} P'$ with $i \sim_\ell i'$ and $\mathit{rel}_\ell(i)$ and $\mathit{rel}_\ell(i')$. Well-formedness of P and P' follows directly from Lemma 12. We perform a case distinction on i:

- if $i = \mathsf{load}(u)$, then $i' = \mathsf{load}(u)$ by definition of $\sim_\ell$. By the reduction rules, assuming that $\mathit{transfer}(\mathsf{load}, \mathit{tag}(u), -) = (\ell_n, -, \ell_{co}, -)$, we then have:

$$\begin{array}{lcl} P & = & \langle K, N \uplus \{n^{\ell_n} : u\}, H, \mathsf{wait}, \mathsf{doc\_req}(u : !K(u)^{\ell_{co}})\rangle \\ P' & = & \langle K', N' \uplus \{n^{\ell_n} : u\}, H', \mathsf{wait}, \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})\rangle \end{array}$$

  To prove the desired conclusion, we need to show:
  (a) $N \uplus \{n^{\ell_n} : u\} \simeq^C_\ell N' \uplus \{n^{\ell_n} : u\}$
  (b) $\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}) \approx_\ell \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})$

  Point (a) is an immediate consequence of (2), while point (b) is more complicated. We distinguish two sub-cases:
  – if $\Gamma_I(u) \cap I(\ell) \neq \emptyset$ then we have $\neg\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}))$ and $\neg\mathit{rel}_\ell(\mathsf{doc\_req}(u : !K'(u)^{\ell_{co}}))$, hence we conclude by applying rules (S-LEFT) and (S-RIGHT);
  – if $\Gamma_I(u) \cap I(\ell) = \emptyset$ then we get $\mathit{ck\text{-}erase}^I_\ell(!K(u)^{\tau_{co}}) = \mathit{ck\text{-}erase}^I_\ell(!K'(u)^{\tau_{co}})$ by Lemma 14 and hence $\mathsf{doc\_req}(u : !K(u)^{\ell_{co}}) \sim_\ell \mathsf{doc\_req}(u : !K'(u)^{\ell_{co}})$ by the definition of $\sim_\ell$.

- if $i = \mathsf{doc\_resp}_n(u : CK, e)$, then $i' = i$ with $\mathit{msg\_label}(u) \not\subseteq I(\ell)$ by the definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:
  – assume that rule (I-DOCRESP) is used on both C and C', then we know that $N = N_0 \uplus \{n^{\tau_n} : u\}$ and $N' = N_1 \uplus \{n^{\tau'_n} : u\}$ for some N_0, N_1 such that $N_0 \simeq^I_\ell N_1$. Assuming that $\mathit{transfer}(\mathsf{doc\_resp}, \tau_n, \mathit{tag}(u)) = (-, \tau_{ci}, -, \tau_s)$ and $\mathit{transfer}(\mathsf{doc\_resp}, \tau'_n, \mathit{tag}(u)) = (-, \tau'_{ci}, -, \tau'_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \xleftarrow{CK}_{\tau_{ci}} u, N_0, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ P' & = & \langle K' \xleftarrow{CK}_{\tau'_{ci}} u, N_1, H', \lceil e \rfloor^{\tau'_s}_{@u}, [\,]\rangle \end{array}$$

    To prove the desired conclusion, we need to show:
    (a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq^I_\ell K' \xleftarrow{CK}_{\tau'_{ci}} u$.
    (b) $\lceil e \rfloor^{\tau_s}_{@u} \simeq^I_\ell \lceil e \rfloor^{\tau'_s}_{@u}$
    We distinguish two subcases:
    * if $I(\tau_n) \cap I(\ell) = \emptyset$ then we know by (2) and the definition of $\simeq^I_\ell$ that $\tau_n = \tau'_n$. Hence we also have $\tau_{ci} = \tau'_{ci}$ and $\tau_s = \tau'_s$. Point (a) follows directly from Lemma 15 and point (b) follows from the definition of $\simeq^I_\ell$.
    * if $I(\tau_n) \cap I(\ell) \neq \emptyset$ then we know by (2) also that $I(\tau'_n) \cap I(\ell) \neq \emptyset$. By the constraints that $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau' ci, \tau_s, \tau'_s\}$. Then (a) follows from Lemma 16, (1) and transitivity and (b) follows directly from the definition of $\simeq^I_\ell$.
  – assume that rule (I-DOCRESP) is used on C, while C' fails using rule (I-COMPLETE). This implies that $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq^I_\ell N'$ and $I(\tau_n) \cap I(\ell) \neq \emptyset$. Assuming that $\mathit{transfer}(\mathsf{doc\_resp}, \tau_n, \mathit{tag}(u)) = (-, \tau_{ci}, -, \tau_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \xleftarrow{CK}_{\tau_{ci}} u, N_0, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,]\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

    To prove the desired conclusion we need to show:
    (a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq^I_\ell K'$
    (b) $\lceil e \rfloor^{\tau_s}_{@u} \simeq^I_\ell \mathsf{wait}$
    (c) $[\,] \approx_\ell \bullet$

We know by the constraints that $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau_s\}$. Then (a) follows from Lemma 16, (1) and transitivity and (b) follows directly from the definition of $\simeq^I_\ell$. Point (c) follows from $\neg$*rel*($\bullet$) and using rules (S-RIGHT) and (S-EMPTY).

- assume that rule (I-DOCRESP) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
- assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

- if $i = \mathsf{doc\_redir}_n(u : CK, u')$, then $i' = i$ with *msg_label*$(u) \not\sqsubseteq I(\ell)$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:
  - assume that rule (I-DOCREDIR) is used on both C and C', then we know that $N = N_0 \uplus \{n^{\tau_n} : u\}$ and $N' = N_1 \uplus \{n^{\tau'_n} : u\}$ for some N_0, N_1 such that $N_0 \simeq^I_\ell N_1$. Assuming that *transfer*($\mathsf{doc\_redir}, \tau_n, \mathit{tag}(u'), -) = (\tau_m, \tau_{ci}, \tau_{co}, -)$, *transfer*($\mathsf{doc\_redir}, \tau'_n, \mathit{tag}(u')) = (\tau'_m, \tau'_{ci}, \tau'_{co}, -)$, $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ and $K_1 = K' \xleftarrow{CK}_{\tau'_{ci}} u$, we have

$$\begin{array}{lcl} P & = & \langle K_0, N_0 \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}})\rangle \\ P' & = & \langle K_1, N_1 \uplus \{n^{\tau'_m} : u'\}, H', \mathsf{wait}, \mathsf{doc\_req}(u' : !K_1(u')^{\tau'_{co}})\rangle \end{array}$$

    To prove the desired conclusion, we need to show:
    (a) $K_0 \simeq^I_\ell K_1$
    (b) $N_0 \uplus \{n^{\tau_m} : u'\} \simeq^I_\ell N_1 \uplus \{n^{\tau'_m} : u'\}$
    (c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \mathsf{doc\_req}(u' : !K_1(u')^{\tau'_{co}})$

    We distinguish two subcases:
    - if $I(\tau_n) \cap I(\ell) = \emptyset$ then we know by (2) and the definition of $\simeq^I_\ell$ that $\tau_n = \tau'_n$. Hence we also have $\tau_m = \tau'_m$, $\tau_{ci} = \tau'_{ci}$ and $\tau_{co} = \tau'_{co}$. Point (a) follows directly from Lemma 15 and point (b) follows from (2) and the definition of $\simeq^I_\ell$. Using (a) and Lemma 14 we can conclude (c) using the definition of $\sim_\ell$ and rule (S-MATCH).
    - if $I(\tau_n) \cap I(\ell) \neq \emptyset$ then by (2) we know that $I(\tau'_n) \cap I(\ell) \neq \emptyset$. By the constraints of the transfer-function we get $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau'_{ci}, \tau_m, \tau'_m\}$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. Point (a) follows from Lemma 16, (1) and transitivity and point (b) follows from (2) and the definition of $\simeq^I_\ell$. Because of $\Gamma_I(u') \cap I(\ell) \neq \emptyset$ we get $\neg$*rel*($\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}})$) and $\neg$*rel*($\mathsf{doc\_req}(u' : !K_1(u')^{\tau'_{co}})$). We can conclude (c) by applying rules (S-LEFT) and (S-RIGHT).
  - assume that rule (I-DOCREDIR) is used on C, while C' fails using rule (I-COMPLETE). This implies that $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq^I_\ell N'$ and $I(\tau_n) \cap I(\ell) \neq \emptyset$. Assuming that *transfer*($\mathsf{doc\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have:

$$\begin{array}{lcl} P & = & \langle K_0, N_0 \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}})\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

    To prove the desired conclusion we need to show:
    (a) $K_0 \simeq^I_\ell K'$
    (b) $N_0 \uplus \{n^{\tau_m} : u'\} \simeq^I_\ell N'$
    (c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \bullet$

    By the constraints of the transfer-function we get $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau_s\}$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$.. Point (a) follows from Lemma 16, (1) and transitivity and point (b) follows from (2) and the definition of $\simeq^I_\ell$. Because of $\Gamma_I(u') \cap I(\ell) \neq \emptyset$ we get $\neg$*rel*($\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}})$) and $\neg$*rel*($\bullet$). We can conclude (c) by applying rules (S-LEFT) and (S-RIGHT).
  - assume that rule (I-DOCREDIR) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
  - assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet\rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet\rangle \end{array}$$

    The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

- if $i = \mathsf{xhr\_resp}_n(u : CK, v)$, then $i' = i$ with *msg_label*$(u) \not\sqsubseteq I(\ell)$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:

– assume that rule (I-XHRRESP) is used on both C and C', then we know that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}$ and $H' = H_1 \uplus \{n^{\tau'_n} : (u, \lceil \lambda x.e' \rfloor_{@u''})\}$ for some H_0, H_1 such that $H_0 \simeq^I_\ell H_1$. Assuming that *transfer*(xhr_resp, τ_n, *tag*(u)) $= (-, \tau_{ci}, -, \tau_s)$ and *transfer*(xhr_resp, τ'_n, *tag*(u)) $= (-, \tau'_{ci}, -, \tau'_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \overleftarrow{}^{CK}_{\tau_{ci}} u, N, H_0, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,] \rangle \\ P' & = & \langle K' \overleftarrow{}^{CK}_{\tau'_{ci}} u, N', H_1, \lceil e\{v/x\} \rfloor^{\tau'_s}_{@u''}, [\,] \rangle \end{array}$$

To prove the desired conclusion, we need to show:

(a) $K \overleftarrow{}^{CK}_{\tau_{ci}} u \simeq^I_\ell K' \overleftarrow{}^{CK}_{\tau'_{ci}} u$.

(b) $\lceil e\{v/x\} \rfloor^{\tau_s}_{@u'} \simeq^I_\ell \lceil e'\{v/x\} \rfloor^{\tau'_s}_{@u''}$

We distinguish two subcases:

∗ if $I(\tau_n) \cap I(\ell) = \emptyset$ then we know by (3) and the definition of $\simeq^I_\ell$ that $\tau_n = \tau'_n$, $u' = u''$ and $e = e'$. Hence we also have $\tau_{ci} = \tau'_{ci}$ and $\tau_s = \tau'_s$. Point (a) follows directly from Lemma 15 and point (b) follows from the definition of $\simeq^I_\ell$.

∗ if $I(\tau_n) \cap I(\ell) \neq \emptyset$ then we know by (3) that $I(\tau'_n) \cap I(\ell) \neq \emptyset$. By the constraints we get $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau'_{ci}, \tau_s, \tau'_s\}$. Point (a) then follows from Lemma 16, (1) and transitivity and point (b) follows from the definition of $\simeq^I_\ell$.

– assume that rule (I-XHRRESP) is used on C, while C' fails using rule (I-COMPLETE). This implies that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}$ for some H_0 with $H_0 \simeq^I_\ell H'$ and $I(\tau_n) \cap I(\ell) \neq \emptyset$. Assuming that *transfer*(xhr_resp, τ_n, $-$) $= (-, \tau_{ci}, -, \tau_s)$, we have:

$$\begin{array}{lcl} P & = & \langle K \overleftarrow{}^{CK}_{\tau_{ci}} u, N, H_0, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,] \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

To prove the desired conclusion we need to show:

(a) $K \overleftarrow{}^{CK}_{\tau_{ci}} u \simeq^I_\ell K'$

(b) $\lceil e\{v/x\} \rfloor^{\tau_s}_{@u'} \simeq^I_\ell \mathsf{wait}$

(c) $[\,] \approx_\ell \bullet$

we know $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau_s\}$ by the constraints. Point (a) then follows from Lemma 16, (1) and transitivity and point (b) follows from the definition of $\simeq^I_\ell$. Point (c) follows form $\neg$*rel*($\bullet$) and using rule (S-RIGHT).

– assume that rule (I-XHRRESP) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;

– assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

• if $i = \mathsf{xhr\_redir}_n(u : CK, u')$, then $i' = i$ with *msg_label*(u) $\not\subseteq I(\ell)$ by definition of $\sim_\ell$. We have four different sub-cases, based on the applied reduction rules:

– assume that rule (I-XHRREDIR) is used on both C and C', then we know that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}$ and $H' = H_1 \uplus \{n^{\tau'_n} : (u, \lceil \lambda x.e' \rfloor_{@u'''})\}$ for some H_0, H_1 such that $H_0 \simeq^I_\ell H_1$. Assuming that *transfer*(xhr_redir, τ_n, *tag*(u'), $-$) $= (\tau_m, \tau_{ci}, \tau_{co}, -)$, *transfer*(xhr_redir, τ'_n, *tag*(u')) $= (\tau'_m, \tau'_{ci}, \tau'_{co}, -)$, $K_0 = K \overleftarrow{}^{CK}_{\tau_{ci}} u$ and $K_1 = K' \overleftarrow{}^{CK'}_{\tau'_{ci}} u$, we have

$$\begin{array}{lcl} P & = & \langle K_0, N, H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle \\ P' & = & \langle K_1, N', H_1 \uplus \{n^{\tau'_m} : (u', \lceil \lambda x.e' \rfloor_{@u'''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}}) \rangle \end{array}$$

To prove the desired conclusion, we need to show:

(a) $K_0 \simeq^I_\ell K_1$

(b) $H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\} \simeq^I_\ell H_1 \uplus \{n^{\tau'_m} : (u', \lceil \lambda x.e' \rfloor_{@u'''})\}$

(c) $\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \sim_\ell \mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}})$

We distinguish two subcases:

∗ if $I(\tau_n) \cap I(\ell) = \emptyset$ then we know by (3) and the definition of $\simeq^I_\ell$ that $\tau_n = \tau'_n$, $e = e'$ and $u'' = u'''$. Hence we also have $\tau_m = \tau'_m$, $\tau_{ci} = \tau'_{ci}$ and $\tau_{co} = \tau'_{co}$. Point (a) follows directly from Lemma 15 and point (b) follows from (3) and the definition of $\simeq^I_\ell$. Using (a) and Lemma 14 we can conclude (c) using the definition of $\sim_\ell$ and rule (S-MATCH).

$*$ if $I(\tau_n) \cap I(\ell) \neq \emptyset$ then we know by (3) that also $I(\tau'_n) \cap I(\ell) \neq \emptyset$. By the constraints we get $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau'_{ci}, \tau_s, \tau'_s\}$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. Point (a) follows from Lemma 16, (1) and transitivity and point (b) follows directly from the definition of $\simeq^I_\ell$. Because of $\Gamma_I(u') \cap I(\ell) \neq \emptyset$ we get $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}))$ and $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_1(u')^{\tau'_{co}}))$. We can conclude (c) by applying rules (S-LEFT) and (S-RIGHT).

– assume that rule (I-XHRREDIR) is used on C, while C' fails using rule (I-COMPLETE). This implies that $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}$ for some H_0 with $H_0 \simeq^I_\ell H'$ and $I(\tau_n) \cap C(\ell) \neq \emptyset$. Assuming that $\mathit{transfer}(\mathsf{xhr\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have

$$\begin{array}{lcl} P & = & \langle K_0, N, H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

To prove the desired conclusion we need to show:

(a) $K_0 \simeq^I_\ell K'$
(b) $H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\} \simeq^I_\ell H'$
(c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell \bullet$

By the constraints we get $I(\tau) \cap I(\ell) \neq \emptyset$ for all $\tau \in \{\tau_{ci}, \tau_s\}$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. Point (a) follows from Lemma 16, (1) and transitivity and point (b) follows directly from the definition of $\simeq^I_\ell$. Because of $\Gamma_I(u') \cap I(\ell) \neq \emptyset$ we get $\neg \mathit{rel}(\mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}))$ and $\neg \mathit{rel}(\bullet)$. We can conclude (c) by applying rules (S-LEFT) and (S-RIGHT).

– assume that rule (I-XHRREDIR) is used on C', while C fails using rule (I-COMPLETE). The case is analogous to the previous one;
– assume that rule (I-COMPLETE) is used on both C and C', then:

$$\begin{array}{lcl} P & = & \langle K, N, H, \mathsf{wait}, \bullet \rangle \\ P' & = & \langle K', N', H', \mathsf{wait}, \bullet \rangle \end{array}$$

The conclusion is trivial, using (1), (2), (3) and the reflexivity of the stream similarity relation;

□

Lemma 18. *$\mathcal{R}^C_\ell$ satisfies the third condition of Definition 11.*

Proof. Let $C = \langle K, N, H, \mathsf{wait}, [\,] \rangle$ and $C' = \langle K', N', H', \mathsf{wait}, [\,] \rangle$ with $C \, \mathcal{R}^I_\ell \, C'$. By definition of $\mathcal{R}^C_\ell$, we have:
(1) $K \simeq^I_\ell K'$;
(2) $N \simeq^I_\ell N'$;
(3) $H \simeq^I_\ell H'$;
(4) C and C' are well-formed.

Assume that $C \xrightarrow{i} P$ with $\neg \mathit{rel}_\ell(i)$. Well-formedness of P follows directly from Lemma 12. We do a case distinction on the applied reduction rule:

- If rule (I-LOAD) is used then $i = \mathsf{load}(u)$. As we always have $\mathit{rel}(i)$ we do not have to consider this case.
- If rule (I-DOCRESP) is used then $i = \mathsf{doc\_resp}_n(u : CK, e)$. We have $\mathit{msg\_label}(u) \subseteq I(\ell)$ because of $\neg \mathit{rel}(i)$ and $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq^I_\ell N'$. Assuming that $\mathit{transfer}(\mathsf{doc\_resp}, \tau_n, \mathit{tag}(u)) = (-, \tau_{ci}, -, \tau_s)$, we have:

$$P \quad = \quad \langle K \xleftarrow{CK}_{\tau_{ci}} u, N_0, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,] \rangle$$

We have to show:

(a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq^I_\ell K'$
(b) $\lceil e \rfloor^{\tau_s}_{@u} \simeq^I_\ell \mathsf{wait}$

By well-formedness we know that $\mathit{msg\_label}(u) \subseteq I(\tau_n)$, hence $I(\tau_n) \cap I(\ell) \neq \emptyset$. By the constraints we can also derive $I(\tau_{ci}) \cap I(\ell) \neq \emptyset$ and $I(\tau_s) \cap I(\ell) \neq \emptyset$. Point (a) follows from (1) and Lemma 16 and point (b) follows from the definition of $\simeq^I_\ell$.

- If rule (I-DOCREDIR) is used, then $i = \mathsf{doc\_redir}_n(u : CK, u')$. We have $\mathit{msg\_label}(u) \subseteq I(\ell)$ because of $\neg \mathit{rel}(i)$ and $N = N_0 \uplus \{n^{\tau_n} : u\}$ for some N_0 with $N_0 \simeq^I_\ell N'$. Assuming that $\mathit{transfer}(\mathsf{doc\_redir}, \tau_n, \mathit{tag}(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have:

$$P \quad = \quad \langle K_0, N_0 \uplus \{n^{\tau_m} : u'\}, H, \mathsf{wait}, \mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle$$

To prove the desired conclusion we need to show:

(a) $K_0 \simeq^I_\ell K'$
(b) $N_0 \uplus \{n^{\tau_m} : u'\} \simeq^I_\ell N'$
(c) $\mathsf{doc\_req}(u' : !K_0(u')^{\tau_{co}}) \approx_\ell [\,]$

By well-formedness we know that *msg_label*$(u) \subseteq I(\tau_n)$, hence $I(\tau_n) \cap I(\ell) \neq \emptyset$. By the constraints of the transfer-function we also get $I(\ell_m) \cap I(\ell) \neq \emptyset$, $I(\ell_{ci}) \cap I(\ell) \neq \emptyset$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. Point (a) follows from Lemma 16 and point (b) follows from (2) and the definition of $\simeq^I_\ell$. By the definition of *rel* we get $\neg$*rel*(doc_req$(u' : !K_0(u')^{\tau_{co}})$). We can conclude (c) by applying rules (S-LEFT) and (S-EMPTY).

- If rule (I-XHRRESP) is used then $i = \mathsf{xhr\_resp}_n(u : CK, v)$. We have *msg_label*$(u) \subseteq I(\ell)$ because of $\neg$*rel*(i) and $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u'})\}$ for some H_0 with $H_0 \simeq^I_\ell H'$. Assuming that *transfer*(xhr_resp$, \tau_n, -) = (-, \tau_{ci}, -, \tau_s)$, we have:

$$P \quad = \quad \langle K \xleftarrow{CK}_{\tau_{ci}} u, N, H_0, \lceil e\{v/x\} \rfloor^{\tau_s}_{@u'}, [\,] \rangle$$

To prove the desired conclusion we need to show:

(a) $K \xleftarrow{CK}_{\tau_{ci}} u \simeq^I_\ell K'$
(b) $\lceil e\{v/x\} \rfloor^{\tau_s}_{@u'} \simeq^I_\ell$ wait

By well-formedness we know that *msg_label*$(u) \subseteq I(\tau_n)$, hence $I(\tau_n) \cap I(\ell) \neq \emptyset$. By the constraints we can also derive $I(\tau_{ci}) \cap I(\ell) \neq \emptyset$ and $I(\tau_s) \cap I(\ell) \neq \emptyset$. Point (a) follows from (1) and Lemma 16 and point (b) follows from the definition of $\simeq^I_\ell$.

- If rule (I-XHRREDIR) is used then $i = \mathsf{xhr\_redir}_n(u : CK, u')$, We have *msg_label*$(u) \subseteq I(\ell)$ because of $\neg$*rel*(i) and $H = H_0 \uplus \{n^{\tau_n} : (u, \lceil \lambda x.e \rfloor_{@u''})\}$ for some H_0 with $H_0 \simeq^I_\ell H'$ and $I(\tau_n) \cap C(\ell) \neq \emptyset$. Assuming that *transfer*(xhr_redir$, \tau_n, tag(u')) = (\tau_m, \tau_{ci}, \tau_{co}, -)$ and $K_0 = K \xleftarrow{CK}_{\tau_{ci}} u$ we have

$$P \quad = \quad \langle K_0, N, H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\}, \mathsf{wait}, \mathsf{xhr\_req}(u' : !K_0(u')^{\tau_{co}}) \rangle$$

To prove the desired conclusion we need to show:

(a) $K_0 \simeq^I_\ell K'$
(b) $H_0 \uplus \{n^{\tau_m} : (u', \lceil \lambda x.e \rfloor_{@u''})\} \simeq^I_\ell H'$
(c) doc_req$(u' : !K_0(u')^{\tau_{co}}) \approx_\ell [\,]$

By well-formedness we know that *msg_label*$(u) \subseteq I(\tau_n)$, hence $I(\tau_n) \cap I(\ell) \neq \emptyset$. By the constraints of the transfer-function we also get $I(\ell_m) \cap I(\ell) \neq \emptyset$, $I(\ell_{ci}) \cap I(\ell) \neq \emptyset$ and $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. Point (a) follows from Lemma 16 and point (b) follows from (3) and the definition of $\simeq^I_\ell$. By the definition of *rel* we get $\neg$*rel*(xhr_req$(u' : !K_0(u')^{\tau_{co}})$). We can conclude (c) by applying rules (S-LEFT) and (S-EMPTY).

- if rule (I-COMPLETE) is used, then we have

$$P \quad = \quad \langle K, N, H, \mathsf{wait}, \bullet \rangle$$

and the claim follows trivially.

□

Lemma 19 (Low Integrity Scripts)**.** *Let $P = \langle K, N, H, \lceil e \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and let $P \mathcal{R}^I_\ell Q$ for some state Q. If $I(\tau_s) \cap I(\ell) \neq \emptyset$ and $P \xrightarrow{o} Q'$ for some o, Q', then $\neg rel_\ell(o)$ and $Q' \mathcal{R}^I_\ell Q$.*

Proof. Let $Q = \langle K', N', H', T', O' \rangle$. By definition of $\mathcal{R}^I_\ell$ we have
(1) $K \simeq^I_\ell K'$;
(2) $N \simeq^I_\ell N'$;
(3) $H \simeq^I_\ell H'$;
(4) $\lceil e \rfloor^{\tau_s}_{@u} \simeq^I_\ell T'$;
(5) $[\,] \simeq^I_\ell O'$;
(6) P and Q are well-formed.

Well-formedness of Q' follows directly from Lemma 12. We do an induction on the term structure of e:
Induction Hypothesis: The claim holds for all $P = \langle K, N, H, \lceil e' \rfloor^{\tau_s}_{@u}, [\,] \rangle$ where e' is a subterm of e.

In the following subcases we will say that the claim follows trivially if the claim follows directly from $(1) - (5)$ and the fact that

- If $e = \lambda x.e'\ v$ then (O-APP) is used and $Q' = \langle K, N, H, \lceil e'\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
- If $e =$ let $x = e_1$ in e_2 then we distinguish two cases:
  - If there exist K^*, N^*, H^*, e^* and o^* with $\langle K, N, H, \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,] \rangle \xmapsto{o^*} \langle K^*, N^*, H^*, \lceil e^* \rfloor^{\tau_s}_{@u}, [\,] \rangle$ then (O-LETCTX) is used and $Q' = \langle K^*, N^*, H^*, \lceil$let $x = e^*$ in $e_2 \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = o^*$. As e_1 is a subterm of e we get $K^* \simeq^I_\ell K'$, $N^* \simeq^I_\ell N'$, $H^* \simeq^I_\ell H'$ and $\neg$*rel*(o^*) by the induction hypothesis. The claim then follows trivially.
  - Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
- If $e =$ let $x = v$ in e' then (O-LET) is used and $Q' = \langle K, N, H, \lceil e'\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
- If $e =$ get-ck(k) then we distinguish two cases:

  - If there exist τ, v with $\mathsf{ck}(k,v)^{\tau} \in !K(u)^{\tau_s}$ then rule (O-GETCOOKIE) is used and $Q' = \langle K, N, H, \lceil v \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \bullet$. The claim follows trivially.
  - Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
- If $e = \mathsf{set\text{-}ck}(k,v)$ then rule (O-SETCOOKIE) is used. Let $CK = \{\mathsf{ck}(k,v)^{\tau} \mid \tau = \kappa(\mathit{host}(u), k)\}$, then , then $Q' = \langle K \xleftarrow{CK}_{\tau_s} u, N, H, \lceil \mathit{unit} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \bullet$. By Lemma 16 and (1) we get $K \xleftarrow{CK}_{\tau_s} u \simeq^I_\ell K'$ The claim then follows trivially.
- If $e = \mathsf{xhr}(u', \lambda x.e')$ then we distinguish two cases:
  - If $\mathit{transfer}(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)$ for some τ_n, τ_{co} then rule (O-XHR) is used and $Q' = \langle K, N, H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ and $o = \mathsf{xhr\_req}(u' : !K(u')^{\tau_{co}})$.
    By the constraints we get $I(\tau_n) \cap I(\ell) \neq \emptyset$ and we get $H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\} \simeq^I_\ell H'$ directly from the definition of $\simeq^I_\ell$ and (3). We also get $\Gamma_I(u') \cap I(\ell) \neq \emptyset$. so we can conclude $\neg \mathit{rel}(o)$. The claim then follows trivially.
  - Otherwise rule (O-COMPLETE) is used and the claim follows trivially.
- For all other forms of e (O-COMPLETE) is used and the claim follows trivially.

□

Lemma 20. *$\mathcal{R}^I_\ell$ satisfies the fourth condition of Definition 11.*

Proof. Let $P = \langle K, N, H, T, O \rangle$ and $C = \langle K', N', H', \mathsf{wait}, [\,] \rangle$, where either $T \neq \mathsf{wait}$ or $O = o :: O'$ for some o, O' by definition of producer state. Assume that $P \mathcal{R}^I_\ell C$, by definition we have:
(1) $K \simeq^I_\ell K'$;
(2) $N \simeq^I_\ell N'$;
(3) $H \simeq^I_\ell H'$;
(4) $T \simeq^I_\ell \mathsf{wait}$;
(5) $O \approx_\ell [\,]$;
(6) P and C are well-formed.
Let $P \xrightarrow{o} Q$. Well-formedness of Q follows directly from Lemma 12. We now distinguish two sub-cases:

- if $O = o :: O'$ for some o, O', the only available reduction rule for P is rule (O-FLUSH), hence:

$$P \xrightarrow{o} \langle K, N, H, T, O' \rangle.$$

  Using (5) and the definition of stream similarity, we have $\neg \mathit{rel}_\ell(o)$ and $O' \approx_\ell [\,]$, which is enough to close the case;
- otherwise, assume without loss of generality that $O = [\,]$, then $T = \lceil e \rfloor^{\tau_s}_{@u}$ for some e, u, τ_s. We observe that point (4) implies that $I(\tau_s) \cap I(\ell) \neq \emptyset$, hence we conclude by Lemma 19;

□

Lemma 21. *$\mathcal{R}^I_\ell$ satisfies the fifth condition of Definition 11.*

Proof. Let $P = \langle K, N, H, T, O \rangle$ and $P' = \langle K', N', H', T', O' \rangle$ with $P \mathcal{R}^I_\ell P'$. By definition of $\mathcal{R}^I_\ell$, we have:
1) $K \simeq^I_\ell K'$;
2) $N \simeq^I_\ell N'$;
3) $H \simeq^I_\ell H'$;
4) $T \simeq^I_\ell T'$;
5) $O \approx_\ell O'$.
6) P and P' are well-formed.

Assume $P \xrightarrow{o} Q$ and $P' \xrightarrow{o'} Q'$. Well-formedness of Q and Q' follows directly from Lemma 12. We distinguish four sub-cases:

- if $O = o :: O_0$ and $O' = o' :: O_1$, the only available reduction rule is (O-FLUSH) for both P and P'. We distinguish three sub-cases:
  - If $\neg \mathit{rel}(o)$ then $O \approx_\ell O'$ is shown by the rule (S-LEFT) and we know $O_0 \approx_\ell O'$. We have $P \xrightarrow{o} Q$ for $Q = \langle K, N, H, T, O_0 \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case b).
  - If $\mathit{rel}(o)$ and $\neg \mathit{rel}(o')$ then $O \approx_\ell O'$ is shown by the rule (S-RIGHT) and we know $O \approx_\ell O_1$. We have $P' \xrightarrow{o} Q'$ for $Q' = \langle K', N', H', T', O_1 \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case c).
  - If $\mathit{rel}(o)$ and $\mathit{rel}(o')$ then $O \approx_\ell O'$ is shown by the rule (S-MATCH) and we know $O_0 \approx_\ell O_1$ and $o \sim_\ell o'$. By the rule (O-FLUSH) we have $P \xrightarrow{o} Q$ and $P' \xrightarrow{o'} Q'$ for $Q = \langle K, N, H, T, O_1 \rangle$ and $Q' = \langle K', N', H', T', O_0 \rangle$. Using $(1), (2), (3)$ and (4), we can match case a).
- if $O = o :: O''$ and $O' = [\,]$, we know that $O \approx_\ell O'$ can only be shown using rule (S-LEFT), hence we have $\neg \mathit{rel}(o)$ and $O'' \approx_\ell [\,]$. We have $P \xrightarrow{o} Q$ for $Q = \langle K, N, H, T, O'' \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case b).

- if $O = [\,]$ and $O' = o' :: O''$, we know that $O \approx_\ell O'$ can only be shown using rule (S-RIGHT), hence we have $\neg \mathit{rel}(o')$ and $[\,] \approx_\ell O''$. We have $P' \xrightarrow{o'} Q'$ for $Q' = \langle K', N', H', T', O'' \rangle$ by the rule (O-FLUSH). Using $(1), (2), (3)$ and (4), we can match case c).
- if $O = O' = [\,]$, then by definition of producer state we have $T = \lceil e \rfloor^{\tau_s}_{@u}$ and $T' = \lceil e' \rfloor^{\tau'_s}_{@u'}$. We distinguish two sub-cases:
  - If $I(\tau_s) \cap I(\ell) \neq \emptyset$ and $P \xrightarrow{o} Q$ then we get $\neg \mathit{rel}(o)$ and $Q \,\mathcal{R}^I_\ell\, P'$ by Lemma 19 and we can match case b).
  - If $I(\tau_s) \cap I(\ell) = \emptyset$ then point (4) implies $e = e'$, $u = u'$ and $\tau_s = \tau'_s$.

    We show the following stronger claim that directly implies a): $o \sim_\ell o'$ and $Q \,\mathcal{R}^I_\ell\, Q'$ and $P \overset{o}{\mapsto} Q \iff P \overset{o'}{\mapsto} Q'$
    We do an induction on the term structure of e:
    Induction Hypothesis: The claim holds for all $P = \langle K, N, H, \lceil e'' \rfloor^{\tau_s}_{@u}, O \rangle$ and $P' = \langle K', N', H', \lceil e'' \rfloor^{\tau_s}_{@u}, O' \rangle$ where e'' is a subterm of e.
    - If $e = \lambda x.e_1\, v$ then (O-APP) is used on P and P' and

      $$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ Q' & = & \langle K', N', H', \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle \end{array}$$

      and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.
    - If $e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ then we distinguish two cases:
      - If there exist K^*, N^*, H^*, e^* and o^* with $\langle K, N, H, \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,] \rangle \xrightarrow{o^*} \langle K^*, N^*, H^*, \lceil e^* \rfloor^{\tau_s}_{@u}, [\,] \rangle$ then as e_1 is a subterm of e by the induction hypothesis we know that there exist $K^{**}, N^{**}, H^{**}, e^{**}$ and o^{**} with $\langle K', N', H', \lceil e_1 \rfloor^{\tau_s}_{@u}, [\,] \rangle \xrightarrow{o^{**}} \langle K^{**}, N^{**}, H^{**}, \lceil e^{**} \rfloor^{\tau_s}_{@u}, [\,] \rangle$ such that $o^* \sim_\ell o^{**}$, $K^* \simeq^I_\ell K^{**}$, $N^* \simeq^I_\ell N^{**}$, $H^* \simeq^I_\ell H^{**}$ and $\lceil e^* \rfloor^{\tau_s}_{@u} \simeq^I_\ell \lceil e^{**} \rfloor^{\tau_s}_{@u}$, hence $e^* = e^{**}$. By rule (O-LETCTX) we get

        $$\begin{array}{lcl} Q & = & \langle K^*, N^*, H^*, \lceil \mathsf{let}\ x = e^*\ \mathsf{in}\ e_2 \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ Q' & = & \langle K^{**}, N^{**}, H^{**}, \lceil \mathsf{let}\ x = e^*\ \mathsf{in}\ e_2 \rfloor^{\tau_s}_{@u}, [\,] \rangle \end{array}$$

        and $o = o^*$ and $o' = o^{**}$ and the claim follows.
      - Otherwise we know by the induction hypothesis that both P and P' use rule (O-COMPLETE). We get

        $$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,] \rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,] \rangle \end{array}$$

        and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.
    - If $e = \mathsf{let}\ x = v\ \mathsf{in}\ e_1$ then (O-LET) is used on P and P' and

      $$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ Q' & = & \langle K', N', H', \lceil e_1\{v/x\} \rfloor^{\tau_s}_{@u}, [\,] \rangle \end{array}$$

      and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.
    - If $e = \mathsf{get\text{-}ck}(k)$ then we distinguish two cases:
      - If there exist τ, v with $\mathsf{ck}(k, v)^\tau \in\, !K(u)^{\tau_s}$ then we know by the constraints that $I(\tau) \cap I(\ell) = \emptyset$ and hence $\mathsf{ck}(k, v)^\tau \in \mathit{ck\text{-}erase}^I_\ell(!K(u)^{\tau_s})$. By Lemma 14 we know that $\mathsf{ck}(k, v)^\tau \in \mathit{ck\text{-}erase}^I_\ell(!K'(u)^{\tau_s})$ and hence also $\mathsf{ck}(k, v)^\tau \in\, !K'(u)^{\tau_s}$. Hence rule (O-GETCOOKIE) is used on P and P' and

        $$\begin{array}{lcl} Q & = & \langle K, N, H, \lceil v \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ Q' & = & \langle K', N', H', \lceil v \rfloor^{\tau_s}_{@u}, [\,] \rangle \end{array}$$

        and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.
      - If there exist no τ, v with $\mathsf{ck}(k, v)^\tau \in\, !K(u)^{\tau_s}$ then we know that there are no τ', v' with $\mathsf{ck}(k, v')^{\tau'} \in\, !K(u)^{\tau_s}$ by Lemma 14. Hence rule (O-COMPLETE) is used on P and P' and

        $$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,] \rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,] \rangle \end{array}$$

        and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.
    - If $e = \mathsf{set\text{-}ck}(k, v)$ then rule (O-SETCOOKIE) is used on . Let $CK = \{\mathsf{ck}(k, v)^\tau \mid \tau = \kappa(\mathit{host}(u), k)\}$, then

      $$\begin{array}{lcl} Q & = & \langle K \xleftarrow{CK}_{\tau_s} u, N, H, \mathsf{wait}, [\,] \rangle \\ Q' & = & \langle K' \xleftarrow{CK}_{\tau_s} u, N', H', \mathsf{wait}, [\,] \rangle \end{array}$$

      and $o = o' = \bullet$. We get $K \xleftarrow{CK}_{\tau_s} u \simeq^I_\ell K' \xleftarrow{CK}_{\tau_s} u$ using (1) and Lemma 16. The claim then follows using $(2), (3)$ and reflexivity.

$*$ If $e = \mathsf{xhr}(u', \lambda x.e')$ then we distinguish two cases:

$\cdot$ If *transfer*$(\mathsf{send}, \tau_s, \mathit{tag}(u')) = (\tau_n, -, \tau_{co}, -)$ for some τ_n, τ_{co} then rule (O-XHR) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ Q' & = & \langle K', N', H' \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}, \lceil \mathsf{unit} \rfloor^{\tau_s}_{@u}, [\,] \rangle \\ o & = & \mathsf{xhr\_req}(u' : !K(u')^{\tau_{co}}) \\ o' & = & \mathsf{xhr\_req}(u' : !K'(u')^{\tau_{co}}) \end{array}$$

We get $H \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\} \simeq^I_\ell H' \uplus \{n^{\tau_n} : (u', \lceil \lambda x.e \rfloor_{@u})\}$ using (3) and the definition of $\simeq^I_\ell$. By Lemma 14 we get $\mathit{ck\text{-}erase}^I_\ell(!K(u)^{\tau_{co}}) = \mathit{ck\text{-}erase}^I_\ell(!K(u)^{\tau_{co}})$ and hence we get $o \sim_\ell o'$ by the definition of $\sim_\ell$. The claim follows using (1), (2) and reflexivity.

$\cdot$ Otherwise rule (O-COMPLETE) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,] \rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,] \rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

$*$ For all other forms of e the rule (O-COMPLETE) is used on P and P' and

$$\begin{array}{lcl} Q & = & \langle K, N, H, \mathsf{wait}, [\,] \rangle \\ Q' & = & \langle K', N', H', \mathsf{wait}, [\,] \rangle \end{array}$$

and $o = o' = \bullet$. The claim follows using $(1), (2), (3)$ and reflexivity.

□