$\mathsf{WEBITaint}$ theorems and proofs

April 13, 2023

1 $\mathsf{WEBITaint}$ rules

Service Init. The case where the booting client is untainted is very similar to the original case.

ServiceInit-Safe
$$\frac{\begin{array}{c}\circ \in \{\heartsuit; \diamondsuit\} \qquad \mathcal{I} = \{(u, v_a)_\circ\} \uplus \mathcal{I}' \qquad \mathtt{WebServices}(u) = (\mathsf{serviceinit}, h) \\ sc = \mathsf{serviceinit}(v_a) \qquad j \ \mathsf{fresh} \qquad CS' = CS \cup \{\langle \mathsf{boot} \rangle_\circ^{j,u}\} \qquad SS' = SS \cup \{sc_\circ^{(h,j)}\}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle \mathcal{I}, CS, SS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle \mathcal{I}', CS', SS' \rangle)}$$

Service Step.

ServiceStep
$$\frac{\begin{array}{c}\circ \in \{\heartsuit; \diamondsuit\} \\ SS = \{\langle sc \rangle_\circ^{(h,j)}\} \uplus SS' \qquad \langle sc, HM(h) \rangle \leadsto_S \langle sc', \mu'_h \rangle \qquad HM' = HM[h \mapsto \mu'_h] \qquad SS'' = SS' \cup \{\langle sc' \rangle_\circ^{(h,j)}\}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle SS, HM \rangle \leadsto_\Phi \mathsf{conf}\langle SS'', HM' \rangle}$$

Service Return.

ServiceRet
$$\frac{\begin{array}{c}\circ, \circ' \in \{\heartsuit; \diamondsuit\} \qquad SS = SS' \uplus \{\langle v_s \rangle_\circ^{(h,j)}\} \\ CS = CS' \uplus \{\langle cc, B \cup \{b^j\}, T \rangle_{\circ'}^u\} \qquad \mathsf{gencc}(v_s) = v_c \qquad CS'' = CS' \cup \{\langle cc, B, T \cup \{(b, v_c)\} \rangle_{\mathbf{LUB}(\circ,\circ')}^u\}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle SS, CS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle SS', CS'' \rangle)}$$

ServiceRetBoot
$$\frac{\circ \in \{\heartsuit; \diamondsuit\} \qquad SS = SS' \uplus \{\langle v_s \rangle_\circ^{(h,j)}\} \qquad CS = CS' \uplus \{\langle \mathsf{boot} \rangle_\circ^{j,u}\} \qquad \mathsf{gencc}(v_s) = cc \qquad CS'' = CS' \cup \{\langle cc, \emptyset, \emptyset \rangle_\circ^u\}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle SS, CS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle SS', CS'' \rangle)}$$

Client Step. Any of the clients makes one step (code execution).

ClientStep
$$\frac{\circ \in \{\heartsuit; \diamondsuit\} \qquad CS = \{\langle cc, B, T \rangle_\circ^u\} \uplus CS' \qquad cc \leadsto_C cc' \qquad CS'' = CS' \cup \{\langle cc', B, T \rangle_\circ^u\}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle CS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle CS'' \rangle)}$$

Client Call. A client calls a service. The call occurs on a client step that sends a value v_a on a Url u while it prepares a handler b. In the server side, a service is ready to respond on this Url.

ClientCall
$$\frac{\begin{array}{c}\circ \in \{\heartsuit; \diamondsuit\} \qquad CS = \{\langle cc, B, T \rangle_\circ^u\} \uplus CS' \qquad cc \leadsto_C^{u'?v_a,b} cc' \qquad \mathtt{WebServices}(u') = (\mathsf{serviceinit}, h) \\ sc = \mathsf{serviceinit}(v_a) \qquad j \ \mathsf{fresh} \qquad B' = B \cup \{b^j\} \qquad CS'' = CS' \cup \{\langle cc', B', T \rangle_\circ^u\} \qquad SS'' = SS \cup \{\langle sc \rangle_\circ^{(h,j)}\}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle CS, SS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle CS'', SS'' \rangle)}$$

Run. The currently running code of a client reaches a value (it has finished computed) and there exists at least one thunk that is ready to run; this thunk moves to running, with the current client memory unchanged.

Run
$$\frac{\circ \in \{\heartsuit; \diamondsuit\} \qquad CS = \{\langle \langle v'_c, \mu \rangle, B, T \uplus \{(\lambda x.P), v_c\} \rangle_\circ^u\} \cup CS' \qquad CS'' = \{\langle \langle P\{x \mapsto v_c\}, \mu \rangle, B, T \rangle_\circ^u\} \cup CS'}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle CS \rangle \leadsto_\Phi \mathsf{Ok}(\mathsf{conf}\langle CS'' \rangle)}$$

Device Reading. If the service is tainted it remains tainted, but untainted services must be tainted if the device is tainted.

DeviceReading

$$\frac{\begin{array}{c}\circ,\circ' \in \{\heartsuit;\diamondsuit\} \qquad SS = \{\langle sc\rangle_{\circ}^{(h,j)}\} \uplus SS' \qquad \langle sc, HM(h)\rangle \rightsquigarrow_{S}^{\mathsf{get}(d,p,v_a)} \langle sc', \mu'_h\rangle \\ DS = \{\langle\varphi\rangle_{\circ'}^{d}\} \uplus DS' \qquad DS \rightsquigarrow_{D}^{\mathsf{get}(d,p,v_a)} DS'' \qquad HM' = HM[h \mapsto \mu'_h] \qquad SS'' = \{\langle sc'\rangle_{\mathbf{LUB}(\circ,\circ')}^{(h,j)}\} \cup SS'\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle SS, HM, DS\rangle \rightsquigarrow_{\Phi} \mathsf{Ok}(\mathsf{conf}\langle SS'', HM', DS'\rangle)}$$

Device Actuator. When the service is untainted, the semantic just performs the actuation ignoring the status of the device.

DeviceActuator1

$$\frac{\begin{array}{c}SS = \{\langle sc\rangle_{\diamondsuit}^{(h,j)}\} \uplus SS' \qquad \langle sc, HM(h)\rangle \rightsquigarrow_{S}^{\mathsf{act}(a,d,p)} \langle sc', \mu'_h\rangle \\ DS \rightsquigarrow_{D}^{\mathsf{act}(a,d,p)} DS' \qquad SS'' = \{\langle sc'\rangle_{\diamondsuit}^{(h,j)}\} \cup SS' \qquad \texttt{last-time}(\mathcal{L}) \leq t' \qquad \mathcal{L}' = (a,d,t') :: \mathcal{L}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle \mathcal{L}, SS, HM, DS\rangle \rightsquigarrow_{\Phi} \mathsf{Ok}(\mathsf{conf}\langle \mathcal{L}', SS'', HM', DS'\rangle)}$$

If the device is not sensitive, the device gets tainted. Plus, $\mathcal{D}_{\mathsf{wo}}$is used to taint all devices that have a dependency with the actuating device. To model this we use a filter function to obtain all the dependent devices: $\mathcal{F}^{\in}_{\mathcal{D}_{\mathsf{wo}}}(DS, d) = \{\langle\varphi\rangle^{d'} \in DS \mid (d,d') \in \mathcal{D}_{\mathsf{wo}}\}$ and $\mathcal{F}^{\notin}_{\mathcal{D}_{\mathsf{wo}}}(DS, d) = \{\langle\varphi\rangle^{d'} \in DS \mid (d,d') \notin \mathcal{D}_{\mathsf{wo}}\}$.

DeviceActuator2

$$\frac{\begin{array}{c}SS = \{\langle sc\rangle_{\heartsuit}^{(h,j)}\} \uplus SS' \\ \langle sc, HM(h)\rangle \rightsquigarrow_{S}^{\mathsf{act}(a,d,p)} \langle sc', \mu'_h\rangle \qquad HM' = HM[h \mapsto \mu'_h] \qquad \Phi(d) = \mathrm{ff} \qquad DS = \{\langle\varphi\rangle_{\circ}^{d}\} \uplus DS' \\ \{\langle\varphi\rangle_{\circ}^{d}\} \rightsquigarrow_{D}^{\mathsf{act}(a,d,p)} \{\langle\varphi'\rangle_{\heartsuit}^{d}\} \qquad DS'' = \{\langle\varphi'\rangle_{\heartsuit}^{d}\} \cup \mathcal{F}^{\notin}_{\mathcal{D}_{\mathsf{wo}}}(DS', d) \cup \{\langle\varphi\rangle_{\heartsuit}^{d'} \mid \langle\varphi\rangle^{d'} \in \mathcal{F}^{\in}_{\mathcal{D}_{\mathsf{wo}}}(DS', d)\} \\ SS'' = \{\langle sc'\rangle_{\heartsuit}^{(h,j)}\} \cup SS' \qquad \texttt{last-time}(\mathcal{L}) \leq t' \qquad \mathcal{L}' = (a,d,t') :: \mathcal{L}\end{array}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle\{\rangle\mathcal{L}, SS, HM, DS\} \rightsquigarrow_{\Phi} \mathsf{Ok}(\mathsf{conf}\langle\{\rangle\mathcal{L}', SS'', HM', DS''\})}$$

If the service is tainted and trying to actuate a "sensitive" device the semantics go to the error status, implying an illegal flow. The rule shows that the analysis is not observing the underlying behavior of the system, but just noticing the flow of the tainted call.

DeviceActuator3

$$\frac{SS = \{\langle sc\rangle_{\heartsuit}^{(h,j)}\} \uplus SS' \qquad \langle sc, HM(h)\rangle \rightsquigarrow_{S}^{\mathsf{act}(a,d,p)} \langle sc', \mu'_h\rangle \qquad DS \rightsquigarrow_{D}^{\mathsf{act}(a,d,p)} DS' \qquad \Phi(d) = \mathrm{tt}}{\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}\langle SS, HM\rangle \rightsquigarrow_{\Phi} \mathsf{Error}(d)}$$

2 Theorems and proofs of WEBITaint

Lemma 1 (One execution step)**.** *For all c_0, c_1 WEBIot configurations, c_2, c_3 WEBITaint configurations, such that $\mathsf{c}_0 \equiv \mathsf{c}_2$ and $\mathsf{c}_1 \equiv \mathsf{c}_3$. Then,*

$$\mathit{wo}, \mathcal{E} \vdash \mathsf{c}_0 \rightsquigarrow \mathsf{c}_1 \;\Leftrightarrow\; \mathcal{D}_{\mathit{wo}}, \mathit{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathit{Ok}(\mathsf{c}_3) \vee \exists d \in \mathbb{D} : \mathcal{D}_{\mathit{wo}}, \mathit{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathit{Error}(d)$$

Proof. ($\Rightarrow$)

$$\mathsf{c}_0 \equiv \mathsf{c}_2 \;\wedge\; \mathsf{c}_1 \equiv \mathsf{c}_3 \qquad \text{(H0)}$$
$$\mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_0 \rightsquigarrow \mathsf{c}_1 \qquad \text{(H1)}$$

Rules of WEBITaint are annotated with $\mathbb{t}$ to make a clear difference. Induction over semantics ($\rightsquigarrow$):

- Rules ServiceInit, ServerStep, RetServiceBoot, RetService, ClientStep, ClientCall, Run, DeviceSensor, DeviceReading are 1-to-1.
- DeviceActuator gets split into three rules.
  - Mapping to ${}_{\mathbb{t}}$DeviceActuator1 and ${}_{\mathbb{t}}$DeviceActuator2 goes to Ok. As seen for ServiceInit, these rules are similar except for the tainting.

- Instead, $_{\mathfrak{t}}$DeviceActuator3 goes to $\mathsf{Error}(d)$ for some device d. Since these three rules cover all combinations of tainting and sinks, these means that there is indeed an error

($\Leftarrow$)

$$\mathsf{c}_0 \equiv \mathsf{c}_2 \ \wedge \ \mathsf{c}_1 \equiv \mathsf{c}_3 \qquad \text{(H0)}$$
$$\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathsf{Ok}(\mathsf{c}_3) \vee \exists d \in \mathbb{D} : \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathsf{Error}(d) \qquad \text{(H1)}$$

Destruct (H1).

- Left side: $\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathsf{Ok}(\mathsf{c}_3)$. By just removing the tainting information, this holds.
- Right side: $\exists d \in \mathbb{D} : \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi} \mathsf{Error}(d)$

  The only rule that goes to Error is $_{\mathfrak{t}}$DeviceActuator3. Since $_{\mathfrak{t}}$DeviceActuator3 shares the same hypothesis as its counterpart DeviceActuator it is clear that the latter can be applied, producing an equivalent configuration.

Qed. □

Lemma 2 (Several execution steps)**.** *For all $n \in \mathbb{N}$, c_0, c_1 WEBIot configurations, it exists c_2, c_3 WEBITaint configurations, such that $\mathsf{c}_0 \equiv \mathsf{c}_2$ and $\mathsf{c}_1 \equiv \mathsf{c}_3$. Then,*

$$\mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_0 \rightsquigarrow^n \mathsf{c}_1 \ \Leftrightarrow \ \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^n \mathsf{Ok}(\mathsf{c}_3) \vee$$
$$[\exists d \in \mathbb{D}, m \in \mathbb{N} : m \leq n \wedge \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^m \mathsf{Error}(d)]$$

Proof. ($\Rightarrow$) Induction on n.

- Base case: $n = 1$. By Lemma 1
- Inductive step: holds for n, must hold for $n + 1$.

$$\mathsf{c}_0 \equiv \mathsf{c}_2 \ \wedge \ \mathsf{c}_1 \equiv \mathsf{c}_3 \qquad \text{(H0)}$$
$$\mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_0 \rightsquigarrow^n \mathsf{c}_1 \ \Rightarrow \ \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^n \mathsf{Ok}(\mathsf{c}_3) \vee [\exists d \in \mathbb{D}, m \in \mathbb{N} : m \leq n \wedge \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^m \mathsf{Error}(d)] \qquad \text{(H1)}$$

If we have Ok, then $\mathsf{c}_1 \equiv \mathsf{c}_3$, then we can apply Lemma 1. In case of Error, we also apply the same Lemma.

($\Leftarrow$) Induction on n.

- Base case: $n = 1$. By Lemma 1.
- Inductive step: holds for n, must hold for $n + 1$.

$$\mathsf{c}_0 \equiv \mathsf{c}_2 \ \wedge \ \mathsf{c}_1 \equiv \mathsf{c}_3 \qquad \text{(H0)}$$
$$\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^n \mathsf{Ok}(\mathsf{c}_3) \vee [\exists d \in \mathbb{D}, m \in \mathbb{N} : m \leq n \wedge \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^m \mathsf{Error}(d)] \Rightarrow \ \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_0 \rightsquigarrow^n \mathsf{c}_1 \qquad \text{(H1)}$$

  - Case $\mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^n \mathsf{Ok}(\mathsf{c}_3)$: by Lemma 1.
  - Case $[\exists d \in \mathbb{D}, m \in \mathbb{N} : m \leq n \wedge \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_2 \rightsquigarrow_{\Phi}^m \mathsf{Error}(d)]$: by Lemma 1.

□

Theorem 1 (TIONI$_{\Phi}$)**.** *For all $\mathcal{I} = \mathcal{I}_{\diamondsuit} \cup \mathcal{I}_{\heartsuit}$, IC,*

$$\nexists \ d \in \mathbb{D} : \mathcal{D}_{\mathsf{wo}}, \mathsf{wo}, \mathcal{E} \vdash \mathsf{conf}_{\mathsf{i}}\langle \mathcal{I}_{\diamondsuit} \uplus \mathcal{I}_{\heartsuit}, DS \rangle \rightsquigarrow_{\Phi}^{*} \mathsf{Error}(d)$$
$$\Rightarrow \ TIONI_{\Phi}(\mathcal{I}_{\diamondsuit}, \mathcal{I}_{\heartsuit})$$

Proof. Instead of looking at the set of observations, we focus on the collection of traces that make up each set. Let $\mathbb{S}_{\diamondsuit}$ be the set of all WEBIot configurations resulting from $\mathsf{conf}_{\mathsf{i}}\langle \mathcal{I}_{\diamondsuit}, DS \rangle$. Alternatively, let $\mathbb{S}_{\heartsuit}$ be the set of all WEBIot configurations resulting from $\mathsf{conf}_{\mathsf{i}}\langle \mathcal{I}_{\diamondsuit} \uplus \mathcal{I}_{\heartsuit}, DS \rangle$. We can also collect the set of executions for WEBITaint: set $\mathbb{S}^{\mathfrak{t}}$ is the set of all configurations resulting from $\mathsf{conf}_{\mathsf{i}}\langle \mathcal{I}_{\diamondsuit} \uplus \mathcal{I}_{\heartsuit}, DS \rangle$ by applying ($\rightsquigarrow_{\Phi}$). Then we can split $\mathbb{S}^{\mathfrak{t}}$ for traces that went to Ok and Error: $\mathbb{S}^{\mathbb{t}}_{\mathsf{O}}$ and $\mathbb{S}^{\mathbb{t}}_{\mathsf{E}}$ respectively. Notice these sets might be of infinite cardinality, since we cannot assume WEBIot systems ever finish.

By hypothesis, we know that $\mathbb{S}^{\mathbb{t}}_{\mathsf{E}}$ is empty. We would like to see that $\mathbb{S}_{\diamondsuit}$ and $\mathbb{S}_{\heartsuit}$ generate the same observations.

Let us assume there is a trace in $(\mathsf{c}_i, \mathsf{c}_f) \in \mathbb{S}_{\heartsuit}$ that generates an observation that cannot be seen by any trace of $\mathbb{S}_{\diamondsuit}$, and that this observation is sensitive, i.e. $\phi(\psi) = \mathit{true}$. Let us take any $\mathsf{WEBITaint}$ configuration $\mathsf{c}^{\mathbb{t}}_i$ such that $\mathsf{c}^{\mathbb{t}}_i \equiv \mathsf{c}_i$. Assuming n such that $\mathsf{wo}, \mathcal{E} \vdash \mathsf{c}_i \rightsquigarrow^n \mathsf{c}_f$, by Lemma 2, there is either

1. n amount of execution steps that lead from $\mathsf{c}^{\mathbb{t}}_i$ to some $\mathsf{c}^{\mathbb{t}}_f$, and $\mathsf{c}^{\mathbb{t}}_f \equiv \mathsf{c}_f$.
2. or $m \leq n$ execution steps that lead to $\mathsf{Error}(d)$ for some device d.

By hypothesis, option 2 is not feasible. Then it has to be that after executing n steps, there is an equivalent $\mathsf{WEBITaint}$ configuration as in option 1. By Lemma 2 the traces $(\mathsf{c}_i, \mathsf{c}_f)$ and $(\mathsf{c}^{\mathbb{t}}_i, \mathsf{c}^{\mathbb{t}}_f)$ behave the same way. Therefore, a new the observation cannot happen in $(\mathsf{c}_i, \mathsf{c}_f)$.

Forcibly, $\mathbb{S}_{\diamondsuit}$ and $\mathbb{S}_{\heartsuit}$ generate the same sensitive observations, and we have $\mathrm{TIONI}(\mathcal{I}_{\diamondsuit}, \mathcal{I}_{\heartsuit})$. $\square$