ADsafety: Type-Based Verification of JavaScript Sandboxing, auxiliary proofs.

1 Language, Reduction Relation, and Types

$$
\begin{array}{lcl}
s & := & \text{constant strings} \\
n & := & \text{numbers} \\
a & := & v \mid \textsf{err} \\
v & := & s \mid n \mid \textsf{null} \mid \textsf{undefined} \mid \textsf{func}\ (x{:}T)\ \{e\} \mid \{s{:}v, \cdots\} \\
e & := & \textsf{let}\ x\ \texttt{=}\ e\ \textsf{in}\ e \mid e(e) \mid e[e] \mid e[e\ \texttt{:=}\ e] \mid \{str{:}e, \cdots\} \\
E & := & \textsf{let}\ x\ \texttt{=}\ E\ \textsf{in}\ e \mid E(e) \mid v(E) \mid E[e] \mid v[E] \\
 & \mid & E[e\ \texttt{:=}\ e] \mid v[E\ \texttt{:=}\ e] \mid \{s{:}v, \cdots, s{:}E, \cdots, s{:}e\}
\end{array}
$$

$$(\textsf{func}(x{:}T)\ \{e\})(v) \hookrightarrow e[x/v] \qquad \text{(E-}\beta\text{)}$$

$$\textsf{let}\ x\ \texttt{=}\ v\ \textsf{in}\ e \hookrightarrow e[x/v] \qquad \text{(E-Let)}$$

$$\frac{s_i = s_f}{\{s_1{:}v_1, \cdots, s_i{:}v_i, \cdots, s_{i+1}{:}v_{i+1}, \cdots\}[s_f] \hookrightarrow v_i} \qquad \text{(E-Found)}$$

$$\frac{s_f \notin (s_1, \cdots, s_i, \cdots) \qquad v_p \neq \textsf{null}}{\{s_1{:}v_1, \cdots, \texttt{"__proto__"}{:}v_p, s_i{:}v_i, \cdots\}[s_f] \hookrightarrow v_p[s_f]} \qquad \text{(E-Proto)}$$

$$\frac{s_f \notin (s_1, \cdots, s_i, \cdots)}{\{s_1{:}v_1, \cdots, \texttt{"__proto__"}{:}\textsf{null}, s_i{:}v_i, \cdots\}[s_f] \hookrightarrow \textsf{undefined}} \qquad \text{(E-ProtoNull)}$$

$$\frac{s_i = s_f}{\{s_1{:}v_1, \cdots, s_i{:}v_i, \cdots, s_{i+1}{:}v_{i+1}, \cdots\}[s_f\ \texttt{:=}\ v_f] \hookrightarrow \{s_1{:}v_1, \cdots, s_i{:}v_f, \cdots, s_{i+1}{:}v_{i+1}, \cdots\}[s_f\ \texttt{:=}\ v_f]} \qquad \text{(E-Update)}$$

$$\frac{s_f \notin (s_1, \cdots)}{\{s_1{:}v_1, \cdots\}[s_f\ \texttt{:=}\ v_f] \hookrightarrow \{s_f{:}v_f, s_1{:}v_1, \cdots\}[s_f\ \texttt{:=}\ v_f]} \qquad \text{(E-Add)}$$

$$\frac{v_f \text{ is not a string or } v_o \text{ is not an object}}{v_o[v_f\ \texttt{:=}\ v] \hookrightarrow \textsf{err}} \qquad \text{(E-UpdateErr)}$$

$$\frac{v_f \text{ is not a string or } v_o \text{ is not an object}}{v_o[v_f] \hookrightarrow \textsf{err}} \qquad \text{(E-LookupErr)}$$

$$\frac{v_f \text{ is not a function}}{(v_f)(v) \hookrightarrow \textsf{err}} \qquad \text{(E-}\beta\text{Err)}$$

$$\frac{e \hookrightarrow e'}{E\langle e\rangle \to E\langle e'\rangle} \qquad \text{(E-Compat)}$$

$$E\langle \textsf{err}\rangle \to \textsf{err} \qquad \text{(E-Err)}$$

$$
\begin{array}{rcl}
T & := & \mathsf{Num} \mid \mathsf{Str} \mid \mathsf{Undef} \mid \mathsf{Null} \mid T \to T \mid T \cup T \\
 & \mid & \{\star : F, \mathit{proto} : T, s : F, \ldots\} \\
 & \mid & (f, \cdots)^+ \mid (f, \cdots)^- \\
F & := & T \mid \text{☠} \mid \mathsf{Absent} \\
\Gamma & := & \cdot \mid \Gamma, x : T
\end{array}
$$

Figure 1: Types

$$
\frac{}{\Gamma \vdash \mathit{str} : (\mathit{str})^+}\ \text{T-StringSet}
$$

$$
\frac{\Gamma \vdash v_p : T_p \qquad \forall i : s_i \in (\mathit{fld}_1, \cdots).\Gamma \vdash v_i : T_i \qquad \forall i : s_i \notin (\mathit{fld}_1, \cdots).\Gamma \vdash v_i : T_\star}{\Gamma \vdash \texttt{\{}s_1\texttt{:}v_1\texttt{,}\cdots\texttt{, "__proto__":}v_p\texttt{\}} : \{\star : T_\star, \mathit{proto} : T_p, \mathit{fld}_1 : T_1, \cdots\}}\ \text{(T-Object)}
$$

Figure 2: Typing objects and strings

$\{\star\}$ is shorthand for $\{\star : F_\star, \mathit{proto} : T_p, f_1 : F_1, \ldots\}$

$$
\mathit{fields}_\star(\{\star\}, S) = \begin{cases} F_\star & : S_\star \neq \emptyset \text{ and } F_\star \neq \mathsf{Absent} \\ \bot & : \text{otherwise} \end{cases} \quad \text{where } S_\star = S - (f_1, \cdots)^+
$$

$$
\mathit{fields}_p(\{\star\}, S) = \begin{cases} \mathsf{Undef} & : T_p = \mathsf{Null} \\ \mathit{fields}(T_p, S_p) & : S_p \neq \emptyset \\ \bot & : \text{otherwise} \end{cases} \quad \text{where } S_p = S - (f_i | F_i \neq \mathsf{Absent})^+
$$

$$
\begin{array}{lcl}
\mathit{fields}(\{\star\}, S) & = & \{T_i \mid f_i \in S \text{ and } F_i = T_i\} \cup \mathit{fields}_\star(\{\star\}, S) \cup \mathit{fields}_p(\{\star\}, S) \\
\mathit{fields}(T_1 \cup T_2, S) & = & \mathit{fields}(T_1, S) \cup \mathit{fields}(T_2, S) \\
\mathit{fields}(T, S) & = & \bot \text{ when } T \in (\mathsf{Str}, \mathsf{Num}, \mathsf{Null}, \mathsf{Undef}, \mathsf{Bool}) \\
\mathit{fields}(T, \emptyset) & = & \bot
\end{array}
$$

$$
\frac{\Gamma \vdash e_o : T_o \qquad \Gamma \vdash e_f : S \qquad S <: \mathsf{Str} \qquad T_{res} = \mathit{fields}(T_o, S)}{\Gamma \vdash e_o\texttt{[}e_f\texttt{]} : T_{res}}\ \text{(T-Lookup)}
$$

Figure 3: Object Lookup

$$
\begin{array}{lcl}
\mathit{objects}(\{\star\}) & = & \{\star\} \\
\mathit{objects}(T_1 \cup T_2) & = & \mathit{objects}(T_1) \cup \mathit{objects}(T_2) \\
\mathit{objects}(T) & = & \bot \text{ otherwise}
\end{array}
$$

$$
\frac{\forall \{F_i | f_i \in S \wedge F_i \neq \mathsf{Absent}\}, T_v <: F_i \qquad S \cap \{f_i | F_i = \mathsf{Absent}\} = \emptyset \qquad S - \{f_i | F_i \neq \mathsf{Absent}\} = \emptyset \vee T_v <: T_\star}{\mathit{okupdate}(\{\star : T_\star, \mathit{proto} : T_p, \mathit{code} : T_c, f_1 : F_1, \cdots\}, S, T_v)} (\textsc{okupdate1})
$$

$$
\frac{\mathit{okupdate}(T_1, S, T_v) \qquad \mathit{okupdate}(T_2, S, T_v)}{\mathit{okupdate}(T_1 \cup T_2, S, T_v)} (\textsc{okupdate2})
$$

$$
\frac{T \in (\mathsf{Undef}, \mathsf{Str}, \mathsf{Bool}, \mathsf{Num}, \mathsf{Null})}{\mathit{okupdate}(T, S, T_v)} (\textsc{okupdate3})
$$

$$
\frac{\Gamma \vdash e_o : T_o \qquad \Gamma \vdash e_f : S \qquad S <: \mathsf{Str} \qquad \Gamma \vdash e_v : T_v \qquad \mathit{okupdate}(T_o, S, T_v)}{\Gamma \vdash e_o \texttt{[} e_f \texttt{ = } e_v \texttt{]} : \mathit{objects}(T_o)} (\textsc{T-Update})
$$

Figure 4: Typing update

$$
\textsc{ST-Union}\ \frac{S <: T_1 \text{ or } S <: T_2}{S <: T_1 \cup T_2} \qquad \textsc{ST-Elim}\ \frac{S_1 <: T \qquad S_2 <: T}{S_1 \cup S_2 <: T} \qquad \textsc{ST-Trans}\ \frac{S <: T \qquad T <: U}{S <: U}
$$

$$
\textsc{ST-Refl}\ S <: S \qquad\qquad \textsc{T-Sub}\ \frac{\Gamma \vdash e : S \qquad S <: T}{\Gamma \vdash e : T}
$$

$$
\frac{F_1 <: F_\star \qquad F_1 \neq \mathsf{Absent} \qquad F_1 \neq \text{☠}}{\{\star : F_\star, \mathit{proto} : T_p, f_1 : F_1, f_2 : F_2, \ldots\} <: \{\star : F_\star, \mathit{proto} : T_p, f_2 : F_2, \ldots\}} (\textsc{ST-Width})
$$

Figure 5: Subtyping

$$\frac{\forall f \in (f_1, \ldots), f \in (s_1, \ldots)}{(f_1, \ldots)^+ <: (s_1, \ldots)^+}\ \text{ST-StringSet}^+ \qquad \frac{\forall f \in (f_1, \ldots), f \notin (s_1, \ldots)}{(f_1, \ldots)^+ <: (s_1, \ldots)^-}\ \text{ST-StringSet}^-$$

$$(f_1, \ldots)^+ <: \mathsf{Str}\ \text{ST-String}^+ \qquad (f_1, \ldots)^- <: \mathsf{Str}\ \text{ST-String}^- \qquad \mathsf{Str} \equiv ()^-\ \text{Equiv-Str}$$

$$\begin{array}{lcl} (f_1, \ldots)^+ - (s_1, \ldots)^+ & = & \forall f_i \notin (s_1, \ldots), (f_i, \ldots)^+ \\ (f_1, \ldots)^- - (s_1, \ldots)^+ & = & (f_1, \ldots, s_1, \ldots)^- \end{array}$$

$$\begin{array}{ll} f \in (f_1, \ldots)^+ & : \exists f_1 . f = f_1 \\ f \in (f_1, \ldots)^- & : \forall f_1 . f \neq f_1 \end{array}$$

Figure 6: String sets and subtyping

2 Preservation

Theorem 1 (Preservation) *If $\Gamma \vdash e : T$, then either $e \to e'$ such that $\Gamma \vdash e' : T$, or $e \to \mathsf{err}$.*

If $e \to e'$, then by case analysis of E, e, ae, and the reduction relation, there exists an E, ae, and e'' such that $\sigma E\langle ae \rangle \to \sigma E\langle e'' \rangle$, $ae \hookrightarrow e''$, and $E\langle e'' \rangle = e'$. There is some subdeduction $\Gamma \vdash ae : U$ of the original proof. We proceed by case analysis on ae to show that the $ae \hookrightarrow e''$ step preserves the type U:

- **let** x `=` v **in** e and v_f`(`v_a`)`—These follow by straightforward use of substitution.
- v_o`[`v_f`]`—By inversion, $\Gamma \vdash v_o : T_o$, $\Gamma \vdash v_f : S$, $S <: \mathsf{Str}$, $\mathit{fields}(T_o, S) <: U$, and there is some object type $T_o' <: T_o$ and $T_o = \{\star : F_\star, \mathit{proto} : T_p, f_1 : F_1, \ldots\}$. By Canonical Forms, v_o is an object `{`s_i`:`v_i`,`$\cdots$`}` and v_f is a string. One of three rules applies:
  - E-Lookup: Further By E-Lookup and Canonical Forms, $v_f = s_i$ for some s_i, and one of the following is true:
    - $F_\star$ is Absent, and there exists $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : T_i$
    - $F_\star$ is not Absent, there exists $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : T_i \cup F_\star$
    - $F_\star$ is not Absent, there does not exist $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : F_\star$.

    In the first case, the result of $\mathit{fields}(T_o', (s_i)^+)$ is just T_i, $\cdot \vdash v_i : T_i$, and $T_i <: U$. In the second case, the *fields* has the same result. In the third, f_i is Absent in T_o', so $\cdot \vdash v_i : F_\star$, which is a subtype of U by lemmas 4 and 5. Since $e'' = v_i$ by E-Lookup, this completes the proof.

  - E-PROTO: From Inversion and Canonical Forms, $\mathit{fields}(T_p, S - (f_i \mid F_i \neq \mathsf{Absent})^+) <: \mathit{fields}(T_o, S)$. From E-PROTO and Canonical Forms, v_o is an object `{`s_1`:`v_1`,`$\cdots$`, "__proto__": v_p}`, v_f is a string, $v_f \notin (s_1, \ldots)^+$, $v_p : T_p$, and $e'' = v_p$`[`v_f`]`. We can use T-LOOKUP to type this new expression, whose result is a subtype of $\mathit{fields}(T_p, S - (f_i \mid F_i \neq \mathsf{Absent})^+)$, which completes the proof.
  - E-PROTONULL This follows from Inversion and Canonical forms, by a similar argument as for E-PROTO.

- v_o`[`v_f `=` v`]`—By inversion, $\Gamma \vdash v_o : T_o$, $\Gamma \vdash v_f : S$, $S <: \mathsf{Str}$, $\Gamma \vdash v : T_v$, there exists an object type $T_o = \{\star : F_\star, \mathit{proto} : T_p, f_1 : F_1, \ldots\}$ with $T'_o <: T_o$, $\mathit{okupdate}(T'_o, S, T_v)$, and $T_o <: U$. By Canonical Forms, v_o is an object `{`s_i`:`v_i`,`$\cdots$`}`, v_f is a string, and $v_f \in S$. One of two rules applies:
  - E-UPDATE: In this case, the new expression e'' is an object $v'_o = \{s_1 : v_1, \cdots, v_f : v, \cdots, s_n, v_n\}$. By *okupdate*, either $\Gamma \vdash v <: F_i$ for some $f_i : F_i$ in T'_o, or $\Gamma \vdash v <: F_\star$. Either case combined with canonical forms, satisfies the precedent of a use of T-OBJECT that can be used to type the new object expression e'' with the same type $T'_o <: U$.
  - E-ADD: This case follows the same argument as for E-UPDATE.

If none of these rules apply, then one of the error relations applies, and by E-COMPAT and E-ERR, the whole computation results in **err**. ■

Definition 1 (Active Expressions) *Active expressions ae are:*

- *v*`(`*v*`)`
- **let** x `=` v **in** e
- *v*`[`*v*`=`*v*`]`
- *v*`[`*v*`]`

Lemma 2 (Canonical Forms) *If $\cdot \vdash v : T$, then if T is:*

- *A string set type S of the form $(s_1, \ldots)^+$ or $(s_1, \ldots)^-$, then v is a string s and $s \in S$.*
- *An object type $T_o = \{\star : F_\star, \mathit{proto} : T_p, f_1 : F_1, \ldots\}$, then*
  1. *v is an object `{`s_1`:`v_1`,`$\cdots$`}`,*
  2. *one of the following is true:*
     - *$F_\star$ is Absent, and for each s_i, v_i there exists $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : T_i$*
     - *$F_\star$ is not Absent, and for each s_i, v_i there exists $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : T_i \cup F_\star$*
     - *$F_\star$ is not Absent, and for each s_i, v_i there does not exist $f_i : T_i$ in T_o with $f_i = s_i$, and $\cdot \vdash v_i : F_\star$.*

Proof By inspection of the typing and subtyping rules for objects and strings.

Lemma 3 (Inversion) *If* $\Gamma \vdash e : T$ *then:*

- *if* $e = e_o[\![e_f]\!]$, *then* $\Gamma \vdash e_f : S$, $S <: \mathsf{Str}$, $\Gamma \vdash e_o : T_o$, *and* $\mathit{fields}(T_o, S) <: T$. *Further, if* $T \neq \bot$, *then there exists a* $T'_o <: T_o$ *with* $T'_o = \{\star : F_\star, \mathit{proto} : T_p, \mathit{code} : T_c, f_1 : F_1, \ldots\}$, $S \neq \emptyset$, *and each of the following is true:*
  - *If* $S - (f_1, \ldots) \neq \emptyset$ *and* $F_\star \neq \mathsf{Absent}$, *then* $F_\star <: T$,
  - *For each* $f_i \in S$, *if* $F_i \neq \mathsf{Absent}$, *then* $F_i <: T$,
  - $\mathit{fields}(T_p, S - (f_i | F_i \neq \mathsf{Absent})^+) <: T$
- *if* $e = e_o[\![e_f = e_v]\!]$, *then* $\Gamma \vdash e_f : S$, $S <: \mathsf{Str}$, $\Gamma \vdash e_o : T_o$, $\Gamma \vdash e_v : T_v$, $T_o = T$, *and there exists a* $T'_o <: T_o$ *such that* T_o *is an object type and* $\mathit{okupdate}(T'_o, S, T_v)$.

Proof

- If $e = e_o[\![e_f]\!]$: The expression was typed either by T-Sub or T-Lookup. If by T-Lookup, the conclusion follows directly. If by T-Sub, then by induction the expression types to some T' using T-Lookup, $T' <: T$, and in that use of T-Lookup, $\mathit{fields}(T_o, S) = T'$. Further, by inspection of *fields*, if $T \neq \bot$, then there exists a T'_o with $T'_o <: T_o$ such that T'_o is an object type, $S \neq \emptyset$, and by inspection of *fields*, all of the above conclusions must hold.
- If $e = e_o[\![e_f = e_v]\!]$: The expression was typed either by T-Sub or T-Update. If by T-Update, then this follows directly. If by T-Sub, then by induction there is some use of T-Update where the expression types to T' and $T' <: T$. T' is either an object type or a union of object types, by inspection *objects*, so there is an object type $T'' <: T'$ such that $\mathit{okupdate}(T', S, T_v)$ holds. ■

Lemma 4 (Fields and Object Types) *If* $\mathit{fields}(T_o, S) = T$, *and* $T_o <: T'_o$, *then* $T <: \mathit{fields}(T'_o, S)$.

Proof T'_o can be a subtype of T_o by either ST-Refl, ST-Union, or ST-Width. The first case is trivial, and the second is obvious from the structure of *fields*. If by ST-Width, the field $f : F$ that is no longer present in the object literal satisfies $F <: F_\star$. The name f will not be included in the first clause of the union in $\mathit{fields}(\{\star\}, S)$, but will be included in $\mathit{fields}_\star(\{\star\}, S)$. By ST-Union, there is a new proof that shows $T <: \mathit{fields}(T'_o, S)$. ■

Lemma 5 *If* $S <: \mathsf{Str}$, $\mathit{fields}(T_o, S) = T$, *and* $S <: S'$, *then* $T <: \mathit{fields}(T_o, S')$, *or* $\mathit{fields}(T_o, S')$ *is undefined.*

Proof If $S = \emptyset$ or there is no object type T'_o such that $T'_o <: T_o$, then $T = \bot$ and the result follows trivially.

By lemma 6, for all $f \in S$, $f \in S'$. Therefore, in the first cast of *fields*, strictly more fields will be included in the union. If any of these fields are ☠, then *fields* is undefined. Otherwise, by ST-UNION, the new type resulting from $\{T_i \mid f_i \in S' \text{ and } F_i = T_i\}$ is a subtype of $\{T_i \mid f_i \in S \text{ and } F_i = T_i\}$.

Let $S'_\star = S' - (f_1, \ldots)^+$ and $S'_p = S' - (f_i \mid F_i \neq \mathsf{Absent})$ for some string type $(f_1, \ldots)$. By lemma 7, $S_\star <: S'_\star$ and $S_p <: S'_p$. By induction, $\mathit{fields}(T_p, S'_p) <: \mathit{fields}(T_p, S_p)$. Thus, $\mathit{fields}_\star(T_o, S) <: \mathit{fields}_\star(T_o, S')$ and $\mathit{fields}_p(T_p, S) <: \mathit{fields}_p(T_p, S')$. By ST-UNION, we complete the proof. ■

Lemma 6 *If $S <: \mathsf{Str}$, $S <: S'$, and $s \in S$, then $s \in S'$.*

Proof S can be a subtype of S' by ST-STRINGSET$^+$, ST-STRINGSET$^-$, ST-STRINGUNION, ST-STRING$^+$, ST-STRING$^-$, or ST-REFL. The result follows by inspection of these rules and the definition of $\in$ for string set types.

Lemma 7 *If $S_1 <: \mathsf{Str}$, $S_2 <: \mathsf{Str}$, $S_1 <: S'_1$, $T = S_1 - S_2$, $T <: S'_1 - S_2$.*

Proof S_1 can be a subtype of S'_1 by ST-STRINGSET$^+$, ST-STRINGSET$^-$, ST-STRINGUNION, ST-STRING$^+$, ST-STRING$^-$, or ST-REFL. The result follows by inspection of these rules and the definitions of $-$ for string set types.