Dynamic analysis of dynamic languages

Software Security

Ana Matos   Pedro Adão
Miguel Correia

☞ "Tight Enforcement of Information-Release Policies for Dynamic Languages", Aslan Askarov and Andrei Sabelfeld, 2009.

☞ "On-the-fly Inlining of Dynamic Security Monitors", Jonas Magazinius et. al, 2012.

Client-side Web applications

- Most consist of (JavaScript) mashups - a combination of scripts, possibly from external untrusted origins (gadgets), that are assembled by the integrator script.
- Gadgets are highly dynamic - they can be loaded at runtime and can depend on the input given by the user.

Browser security

- Same Origin Policy (SOP) - a script loaded from one origin is not allowed to access or modify resources obtained from another origin. Implemented in current browsers.
- Access is granted based on the origin, and not on what will be done with that access.
- Information flow violations in the Browser: Cookie Stealing, Location Hijacking, History Sniffing, Behavior Tracking…

Information flow control in the browser

"Information Flow Control is a good fit for whole-browser security." [Yang 2013]

- It can perfectly capture the SOP
- It can also express more fine-grained security policies for eliminating security vulnerabilities in Web applications, while allowing for cross-origin communication.

Dynamic code evaluation

- In the context of web applications, dynamic code evaluation is a popular feature.
  - Nearly a quarter of the pages with embedded JavaScript (indexed by Google code search) uses the eval primitive.
- Example: Scripts for embedding maps in trusted web pages (eg. Google Maps) imply that new code for map rendering is requested and run in response to user events such as moving the map.

Client-side Web security

- As JavaScript is a highly dynamic language -- , mainly dynamic approaches to control information flow. Two main approaches:
  - Modify a JavaScript engine so that it additionally implements the security monitor.
    Example: Lock-step monitor
  - Inline the monitor into the original program, which has the advantage of being browser-independent.
    Example: Inlining compiler

Inlining monitors

- Monitored programs are in the same language as original programs. Therefore, no changes are necessary to the execution environments of the programs.
- Inlined reference monitoring is a mainstream technique for enforcing safety properties.
  - Example for the Web: BrowserShield instruments scripts with checks for known vulnerabilities.

Topics

- Web security and Dynamic Language
- Monitoring Information flow in Dynamic WHILE
  - Lock-step monitor for Dynamic WHILE
  - Inlining compiler of a monitor for Dynamic WHILE

Topics

- Web security and Dynamic Language
- Monitoring Information flow in Dynamic WHILE
    - Lock-step monitor for Dynamic WHILE
    - Inlining compiler of a monitor for Dynamic WHILE

Lock-step monitor: Idea

- To define a monitored semantics for the composition of the program and monitor.
- The monitor can only perform safe executions.
- Synchronize each step of the program and of the monitor.
- Steps that don’t synchronise get blocked (don’t terminate).

Lock-step monitored semantics

- Idea: To compose ($|_m$) the execution of the program and of the monitor

Labelled transition between program configurations

Labelled transition between monitor (m) configurations

$$\frac{cfg \Rightarrow^{\alpha} cfg' \qquad cfgm \Rightarrow_m^{\alpha} cfgm'}{< cfg \;|_m\; cfgm > \Rightarrow <cfg' \;|_m\; cfgm' >}$$

Transition between monitored program configurations

Syntax of Dynamic WHILE

- Syntactic categories:

s - strings
c ∈ *Z* - constants (integers)   } values
x ∈ *Var* - variables
a ∈ *Aexp* - arithmetic expressions
t ∈ *Bexp* - tests
S ∈ *Stm* - statements

- Grammar (BNF notation):

op ::= + | - | * | / | ++   cmp ::= < | ≤ | = | ≠ | ≥ | >

a ::= s | c | x | a_1 op a_2 t ::= a_1 cmp a_2

S ::= x:=a |skip | S_1 ; S_2 | if t then S else S | while t do S |
| eval a

Rules for WHILE

- Skip: $< \text{skip}, \rho > \Rightarrow^{nop} \rho$
- Assignment: $< x{:=}a, \rho > \Rightarrow^{(x,a)} \rho[x \mapsto \mathcal{A}[\![a]\!]_\rho]$
- End: $< \text{end}, \rho > \Rightarrow^{f} \rho$
- Sequential composition:

$$\frac{<S_1, \rho> \Rightarrow^{\alpha} <S_1', \rho'>}{< S_1;S_2, \rho> \Rightarrow^{\alpha} <S_1';S_2, \rho'>} \qquad \frac{<S_1, \rho> \Rightarrow^{\alpha} \rho'}{< S_1;S_2, \rho> \Rightarrow^{\alpha} <S_2, \rho'>}$$

- Conditional test:

$<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \Rightarrow^{b(t)} <S_1;\text{end}, \rho>$ if $\mathcal{B}[\![t]\!]_\rho = \text{true}$

$<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \Rightarrow^{b(t)} <S_2;\text{end}, \rho>$ if $\mathcal{B}[\![t]\!]_\rho = \text{false}$

- While loop:

$<\text{while } t \text{ do } S, \rho> \Rightarrow^{b(t)} <S;\text{end};\text{while } t \text{ do } S, \rho>$ if $\mathcal{B}[\![t]\!]_\rho = \text{true}$

$<\text{while } t \text{ do } S, \rho> \Rightarrow^{b(t)} <\text{end}, \rho>$ if $\mathcal{B}[\![t]\!]_\rho = \text{false}$

… plus for the dynamic code evaluation

- Eval:

Evaluates the expression into a string…

… parses the string into a command…

$$\frac{A[\![a]\!]_\rho = s \qquad \mathrm{parse}(s) = S}{\langle\, \mathrm{eval}(a), \rho \,\rangle \Rightarrow^{b(a)} \langle\, S\,;\mathrm{end}, \rho \,\rangle}$$

… and prepares to evaluate the command.

(The label and end is similar to that of a branch.)

Monitor semantics (same as for WHILE)

- nop: $st \Rightarrow_m^{nop} st$
- branching: $st \Rightarrow_m^{b(a)} lev(a) :: st$
- end: $l : st \Rightarrow_m^{f} st$
- assignment: $$\frac{lev(a) \leq \Gamma(x) \quad lev(st) \leq \Gamma(x)}{st \Rightarrow_m^{(x,a)} st}$$

$lev(a) = \bigvee$ levels of the variables in a

$lev(st) = \bigvee$ levels in st

Example 1

$\rho(x_H)=\text{true}$
$\rho(y_L)=\text{“}y_L := 1\text{”}$

Program execution:

$<(\text{if } x_H \text{ then eval}(y_L) \text{ else } x_H := 0) \,, \rho >$

$\Rightarrow^{b(xH)} < \text{eval}(y_L); \text{end} \,, \rho >$

$\Rightarrow^{b(yL)} < y_L := 1; \text{end}; \text{end} \,, \rho >$

$\Rightarrow^{(yL,1)} < \text{end}; \text{end} \,, \rho[y_L \mapsto 1] >$

...

Monitor execution:

ε

$\Rightarrow^{b(xH)} H::\varepsilon$

$\Rightarrow^{b(yL)} L::H::\varepsilon$

$\Rightarrow^{(yL,1)}$

Monitored execution:

$< <(\text{if } x_H \text{ then eval}(y_L) \text{ else } x_H := 0) \,, \rho > |_m \,\varepsilon >$

$\Rightarrow^{b(xH)} < < \text{eval}(y_L); \text{end} \,, \rho > |_m \, H::\varepsilon >$

$\Rightarrow^{b(yL)} < < yL := 1; \text{end}; \text{end}, \rho > |_m \, L::H::\varepsilon > \Rightarrow$

Example 2

$\rho(x_H) = \text{true}$
$\rho(y_L) = \text{“}y_L := 1\text{”}$

Program execution:

$<(\text{if } x_H \text{ then } z_H := \text{“yL := 1”} \text{ else } z_H := \text{“yL := 0”} ; \text{eval}(z_H)) , \rho >$

$\Rightarrow^{b(xH)} < z_H := \text{“yL := 1”}; \text{end}; \text{eval}(z_H) , \rho >$

$\Rightarrow^{(zH, \text{“yL := 1”})} < \text{end}; \text{eval}(z_H) , \rho[z_H \mapsto \text{“yL := 1”}] >$

$\Rightarrow^{f} < \text{eval}(z_H) , \rho[z_H \mapsto \text{“yL := 1”}] >$

$\Rightarrow^{b(zH)} < \text{yL := 1;end} , \rho[z_H \mapsto \text{“yL := 1”}] >$

$\Rightarrow^{(yL, 1)} < \text{end} , \rho[y_L \mapsto 1, z_H \mapsto \text{“yL := 1”}] >$

…

Monitor execution:

ε

$\Rightarrow^{b(xH)} H::\varepsilon$

$\Rightarrow^{(zH, \text{“yL := 1”})} H::\varepsilon$

$\Rightarrow^{f} \varepsilon$

$\Rightarrow^{b(zH)} H::\varepsilon$

$\Rightarrow^{(yL, 1)}$ ✗

Topics

- Web security and Dynamic Language
- Monitoring Information flow in Dynamic WHILE
  - Lock-step monitor for Dynamic WHILE
  - Inlining compiler of a monitor for Dynamic WHILE

Inlined monitor: Idea

- To define a source-to-source compiler that transforms programs into "programs that monitor themselves".
- The monitor will be inlined into the program.
- Transformed programs should be safe to execute.

Syntax of Dynamic WHILE

- Syntactic categories:

  l - security levels
  s - strings
  $c \in Z$ - constants (integers)
  $x \in Var$ - variables
  $a \in Aexp$ - arithmetic expressions
  $t \in Bexp$ - tests
  $S \in Stm$ - statements

  values

We also assume informally that the language can express function definitions and calls.

- Grammar (BNF notation):

  $op ::= + \mid - \mid * \mid / \mid ++ \mid \vee$ $\quad cmp ::= < \mid \leq \mid = \mid \neq \mid \geq \mid >$

  $a ::= l \mid s \mid c \mid x \mid a_1\ op\ a_2$ $\quad t ::= a_1\ cmp\ a_2$

  $S ::= x{:=}a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } t \text{ then } S \text{ else } S \mid \text{while } t \text{ do } S \mid$
  $\mid \text{eval } a$

... plus for the dynamic code evaluation

Evaluates the expression into a string...

- Eval:

$$\frac{\mathcal{A}[\![a]\!]_\rho = s \quad \mathrm{parse}(s) = S}{<\mathrm{eval}(a), \rho> \Rightarrow\ < S, \rho>}$$

... parses the string into a command...

... and prepares to evaluate the command.

Labels and ‘end’ are not needed.

Inlining compiler

Original program in the Dynamic WHILE language

Transformed program, also in the Dynamic WHILE language

$$[\![S]\!]_{\Gamma}^{pc} = S'$$

Mapping from variables to security levels

Security level of the “program counter”

Compilation of statements

$$[\![\mathrm{skip}]\!]_{\Gamma}^{pc} = \mathrm{skip}$$

This instruction does not have any impact in our security setting

Compilation of statements

$[\![x := a]\!]_{\Gamma}^{pc}$

$=$

if $(pc \vee lev(a) \leq \Gamma(x))$ then $x := a$ else loop

Only performs the assignment if flow is safe.

while true do skip

Compilation of statements

$$[\![S_1 \; ; S_2]\!]_{\Gamma^{pc}} = \quad [\![S_1]\!]_{\Gamma^{pc}} \; ; [\![S_2]\!]_{\Gamma^{pc}}$$

Compilation of statements

$$[\![\text{if } t \text{ then } S_1 \text{ else } S_2]\!]_{\Gamma}^{pc}$$

$$=$$

$$\text{if } t \text{ then } [\![S_1]\!]_{\Gamma}^{pc \vee lev(t)} \text{ else } [\![S_2]\!]_{\Gamma}^{pc \vee lev(t)}$$

Transform the branches using a PC level that is updated with that of the test.

Compilation of statements

$[\![\text{while t then S}]\!]_{\Gamma^{pc}}$

$=$

$\text{while t then } [\![S]\!]_{\Gamma^{pc \vee lev(t)}}$

Transform the body using a PC level that is updated with that of the test.

Compilation of statements

Make a string encoding what we want to evaluate at runtime.

$[\![\text{eval a}]\!]_\Gamma{}^{pc}$

=

eval "$[\![\text{parse(a)}]\!]$"++string(Γ,pc$\vee$lev(a))

Evaluate the expression (into a string), parse it, compile the resulting statement, and evaluate the result.

All monitor inlining compilation rules

$[\![\text{skip}]\!]_\Gamma^{pc} =$ skip

$[\![x := a]\!]_\Gamma^{pc} =$ if $(pc \vee lev(a) \leq \Gamma(x))$ then $x := a$ else loop

$[\![S_1 \,;\, S_2]\!]_\Gamma^{pc} = [\![S_1]\!]_\Gamma^{pc} \,;\, [\![S_2]\!]_\Gamma^{pc}$

$[\![\text{if } t \text{ then } S_1 \text{ else } S_2]\!]_\Gamma^{pc} =$

if t then $[\![S_1]\!]_\Gamma^{pc \vee lev(t)}$ else $[\![S_2]\!]_\Gamma^{pc \vee lev(t)}$

$[\![\text{while } t \text{ then } S]\!]_\Gamma^{pc} =$ while t then $[\![S]\!]_\Gamma^{pc \vee lev(t)}$

$[\![\text{eval } a]\!]_\Gamma^{pc} =$ eval "$[\![\text{parse}(a)]\!]$"++string$(\Gamma, pc \vee lev(a))$

Example

Program S to execute: x_M:="y_L:=z_H" ; eval(x)

$[\![S]\!]_\Gamma^L$

= $[\![x_M$:="y_L:=z_H" ; eval(x)$]\!]_\Gamma^L$

Compile it into a program with an inlined monitor.

= $[\![x_M$:="y_L:=z_H"$]\!]_\Gamma^L$; $[\![$eval(x_M)$]\!]_\Gamma^L$

= if $L \vee L \leq L$ then x_M := "y_L:=z_H" else loop ;

eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$)

Example

Program S to execute: x_M:="y_L:=z_H" ; eval(x)

$[\![S]\!]_\Gamma^L$

$= [\![x_M := "y_L := z_H" \; ; \; \mathrm{eval}(x)]\!]_\Gamma^L$

$= [\![x_M := "y_L := z_H"]\!]_\Gamma^L \; ; \; [\![\mathrm{eval}(x_M)]\!]_\Gamma^L$

$= \text{if } L \vee L \leq M \text{ then } x_M := "y_L := z_H" \text{ else loop ;}$

eval "$[\![\mathrm{parse}(x_M)]\!]$"++string(Γ

Sequential composition of the transformed subprograms.

Example

Program S to execute: x_M:="y_L:=z_H

Will later (at runtime) retrieve the string in x_M, parse it, and compile it using Γ and an updated PC level, and execute it.

$[\![S]\!]_\Gamma^L$

$= [\![x_M := \text{"}y_L := z_H\text{"} ; \text{eval}(x)]\!]_\Gamma^L$

$= [\![x_M := \text{"}y_L := z_H\text{"}]\!]_\Gamma^L ; [\![\text{eval}(x_M)]\!]_\Gamma^L$

$=$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;

eval "$[\![\text{parse}(x_M)]\!]$"++string(Γ,L∨M)

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;

eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), ρ>

$\Rightarrow$ < x_M:="y_L:=z_H" ; eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), ρ>

$\Rightarrow$ < eval "$[\![$parse(xM)$]\!]$"++string

$\Rightarrow$ < $[\![$parse(x_M)$]\!]_{\Gamma}^{M}$, $\rho[x_M \mapsto$"y_L:=z_H"$]$>

$\Rightarrow$ < >

$\Rightarrow$ <

$\Rightarrow$ < if ($M \vee H \leq L$) then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]$>

$\Rightarrow$ < loop , $\rho[x_M \mapsto$"y_L:=z_H"$]$> $\Rightarrow$...

Execute the program with the inlined monitor.

Implicit: Function definition of $[\![\ ]\!]$ and parse()

Example (execution)

$< [\![S]\!]_\Gamma^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L{:=}z_H$" else loop ;
eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho>$

$\Rightarrow < x_M{:=}$"$y_L{:=}z_H$" ; eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho>$

$\Rightarrow <$ eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![$parse$(x_M)]\!]_\Gamma^{M} , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![$parse("$y_L{:=}z_H$")$]\!]_\Gamma^{M} , \rho[x_M \mapsto$"$yL{:=}z_H$"$]>$

$\Rightarrow < [\![y_L{:=}z_H]\!]_\Gamma^{M} , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L{:=}z_H$ else loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]> \Rightarrow \ldots$

Example (execution)

Next step is a conditional.

$< [\![S]\!]_\Gamma{}^L , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L{:=}z_H$" else loop ;

eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho>$

$\Rightarrow < x_M{:=}$"$y_L{:=}z_H$" ; eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho>$

$\Rightarrow <$ eval "$[\![$parse$(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![$parse$(x_M)]\!]_\Gamma{}^M , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![$parse("$y_L{:=}z_H$")$]\!]_\Gamma{}^M , \rho[x_M \mapsto$"$yL{:=}z_H$"$]>$

$\Rightarrow < [\![y_L{:=}z_H]\!]_\Gamma{}^M , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L{:=}z_H$ else loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L:=z_H$" else loop ;
eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho>$

$\Rightarrow < x_M:=$"$y_L:=z_H$" ; eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow < [\![\text{parse}(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow < [\![\text{parse}($"$y_L:=z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$yL:=z_H$"$]>$

$\Rightarrow < [\![y_L:=z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L:=z_H$ else loop , $\rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L:=z_H$"$]> \Rightarrow \ldots$

Example (execution)

Next step is an assignment.

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L{:=}z$

eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma$,$L \vee M$), $\rho>$

$\Rightarrow <$ $x_M{:=}$"$y_L{:=}z_H$" ; eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma$,$L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma$,$L \vee M$), $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![\text{parse}(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![\text{parse}($"$y_L{:=}z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$yL{:=}z_H$"$]>$

$\Rightarrow < [\![y_L{:=}z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L{:=}z_H$ else loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L{:=}z_H$"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;

eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho>$

$\Rightarrow <$ x_M:="y_L:=z_H" ; eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![\text{parse}(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow < [\![\text{parse}(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow < [\![\text{parse}($"$y_L$:=$z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"yL:=$z_H$"$]>$

$\Rightarrow < [\![y_L := z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L{:=}z_H$"

eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L\vee$

$\Rightarrow <$ x_M:="$y_L{:=}z_H$" ; eval "$[\![parse($ ... $)]\!]$"++string(Γ,L∨M), ρ>

Evaluate and parse the string.

$\Rightarrow <$ eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L\vee M$), $\rho[x_M \mapsto$ "$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![parse(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "$y_L{:=}z_H$"$]>$

$\Rightarrow < [\![parse($"$y_L{:=}z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "$yL{:=}z_H$"$]>$

$\Rightarrow < [\![y_L{:=}z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "$y_L{:=}z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L{:=}z_H$ else loop , $\rho[x_M \mapsto$ "$y_L{:=}z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$ "$y_L{:=}z_H$"$]> \Rightarrow \ldots$

Example (execution)

$< \llbracket S \rrbracket_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L := z_H$" else loop ;
eval "$\llbracket parse(x_M) \rrbracket$"++string($\Gamma, L \vee M$), $\rho >$

$\Rightarrow < x_M :=$ "$y_L := z_H$" ; eval "$\llbracket parse(x_M) \rrbracket$"++string($\Gamma, L \vee M$), $\rho >$

$\Rightarrow <$ eval "$\llbracket parse(x_M) \rrbracket$"++string($\Gamma, L \vee M$), $\rho[x_M \mapsto$ "$y_L := z_H$"$] >$

$\Rightarrow < \llbracket parse(x_M) \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$ "$y_L := z_H$"$] >$

$\Rightarrow < \llbracket parse($"$y_L := z_H$"$) \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$ "$yL := z_H$"$] >$

$\Rightarrow < \llbracket y_L := z_H \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$ "$y_L := z_H$"$] >$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L := z_H$ else loop , $\rho[x_M \mapsto$ "$y_L := z_H$"$] >$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$ "$y_L := z_H$"$] > \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L:=z$

eval "$[\![parse(x_M)]\!]$"++string(Γ

Evaluate the argument

$\Rightarrow < x_M :=$"$y_L:=z_H$" ; eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho >$

$\Rightarrow <$ eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow < [\![parse(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow < [\![parse($"$y_L:=z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$yL:=z_H$"$]>$

$\Rightarrow < [\![y_L:=z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L:=z_H$ else loop , $\rho[x_M \mapsto$"$y_L:=z_H$"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L:=z_H$"$]> \Rightarrow \ldots$

Example (execution)

$< \llbracket S \rrbracket_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "$y_L := z_H$" else loop ;

eval “$\llbracket parse(x_M) \rrbracket$"++string$(\Gamma, L \vee M)$, $\rho >$

$\Rightarrow < x_M :=$"$y_L := z_H$" ; eval “$\llbracket parse(x_M) \rrbracket$"++string$(\Gamma, L \vee M)$, $\rho >$

$\Rightarrow <$ eval “$\llbracket parse(x_M) \rrbracket$"++string$(\Gamma, L \vee M)$, $\rho[x_M \mapsto$"$y_L := z_H$"$] >$

$\Rightarrow < \llbracket parse(x_M) \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L := z_H$"$] >$

$\Rightarrow < \llbracket parse($"$y_L := z_H$"$) \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$"$yL := z_H$"$] >$

$\Rightarrow < \llbracket y_L := z_H \rrbracket_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L := z_H$"$] >$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L := z_H$ else loop , $\rho[x_M \mapsto$"$y_L := z_H$"$] >$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"$y_L := z_H$"$] > \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "y_L:=

eval "$[\![parse(x_M)]\!]$"++string(

Parse the retrieved string.

$\Rightarrow < x_M$:="y_L:=z_H" ; eval "$[\![parse(x_M)]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![parse(x_M)]\!]$"++string(Γ,$L \vee M$), $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow < [\![parse(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow < [\![parse($"y_L:=z_H"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"yL:=$z_H$"$]>$

$\Rightarrow < [\![y_L := z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho>$

$= <$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;
eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ x_M:="y_L:=z_H" ; eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ $[\![$parse(x_M)$]\!]_{\Gamma}^{M}$, $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ $[\![$parse("y_L:=z_H")$]\!]_{\Gamma}^{M}$, $\rho[x_M \mapsto$"yL:=z_H"$]>$

$\Rightarrow <$ $[\![y_L$:=$z_H]\!]_{\Gamma}^{M}$, $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ if ($M \vee H \leq L$) then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]> \Rightarrow$...

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "y_L

eval "$[\![parse(x_M)]\!]$"++strin

Compile the assignment.

$\Rightarrow < x_M$:="y_L:=z_H" ; eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho >$

$\Rightarrow <$ eval "$[\![parse(x_M)]\!]$"++string($\Gamma, L \vee M$), $\rho[x_M \mapsto$ "y_L:=z_H"$] >$

$\Rightarrow < [\![parse(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "y_L:=z_H"$] >$

$\Rightarrow < [\![parse($"y_L:=z_H"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "yL:=z_H"$] >$

$\Rightarrow < [\![y_L := z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$ "y_L:=z_H"$] >$

$\Rightarrow <$ if $(M \vee H \leq L)$ then y_L:=z_H else loop , $\rho[x_M \mapsto$ "y_L:=z_H"$] >$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$ "y_L:=z_H"$] > \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_\Gamma^L , \rho>$

$= <$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;
eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow < x_M$:="y_L:=z_H" ; eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow < [\![$parse(x_M)$]\!]_\Gamma^M , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow < [\![$parse("y_L:=z_H")$]\!]_\Gamma^M , \rho[x_M \mapsto$"yL:=$z_H$"$]>$

$\Rightarrow < [\![y_L := z_H]\!]_\Gamma^M , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow <$ if ($M \vee H \leq L$) then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_{\Gamma}^{L} , \rho >$

$= <$ if $L \vee L \leq M$ then $x_M :=$ "y_L:

eval "$[\![\text{parse}(x_M)]\!]$"++string

Execute the if.

$\Rightarrow < x_M$:="y_L:=z_H" ; eval "$[\![\text{parse}(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho>$

$\Rightarrow <$ eval "$[\![\text{parse}(x_M)]\!]$"++string$(\Gamma, L \vee M), \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow < [\![\text{parse}(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow < [\![\text{parse}($"$y_L$:=$z_H$"$)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"yL:=$z_H$"$]>$

$\Rightarrow < [\![y_L := z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto$"$y_L$:=$z_H$"$]>$

$\Rightarrow <$ if $(M \vee H \leq L)$ then $y_L := z_H$ else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]> \Rightarrow \ldots$

Example (execution)

$< [\![S]\!]_\Gamma^L , \rho>$

$= <$ if $L \vee L \leq M$ then x_M := "y_L:=z_H" else loop ;
eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ x_M:="y_L:=z_H" ; eval "$[\![$parse(x_M)$]\!]$"++string(Γ,$L \vee M$), $\rho>$

$\Rightarrow <$ eval "$[\![$parse(x_M)v"++string(Γ,$L \vee M$), $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ $[\![$parse(x_M)$]\!]_\Gamma^M$, $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ $[\![$parse("y_L:=z_H")$]\!]_\Gamma^M$, $\rho[x_M \mapsto$"yL:=z_H"$]>$

$\Rightarrow <$ $[\![y_L$:=$z_H]\!]_\Gamma^M$, $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ if ($M \vee H \leq L$) then y_L:=z_H else loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$

$\Rightarrow <$ loop , $\rho[x_M \mapsto$"y_L:=z_H"$]>$ $\Rightarrow$...

Example (execution)

$$< [\![S]\!]_{\Gamma}^{L} , \rho>$$

$$= < \text{if } L \vee L \leq M \text{ then } x_M := \text{"}y_L{:=}z_H\text{"} \text{ else loop ;}$$
$$\quad \text{eval ``}[\![\text{parse}(x_M)]\!]\text{"++string}(\Gamma, L \vee M), \rho>$$

$$\Rightarrow < x_M{:=}\text{"}y_L{:=}z_H\text{" ; eval ``}[\![\text{parse}(x_M)]\!]\text{"++string}(\Gamma, L \vee M), \rho>$$

$$\Rightarrow < \text{eval ``}[\![\text{parse}(x_M)]\!]\text{"++string}(\Gamma, L \vee M), \rho[x_M \mapsto \text{"}y_L{:=}z_H\text{"}]>$$

$$\Rightarrow < [\![\text{parse}(x_M)]\!]_{\Gamma}^{M} , \rho[x_M \mapsto \text{"}y_L{:=}z_H\text{"}]>$$

$$\Rightarrow < [\![\text{parse}(\text{"}y_L{:=}z_H\text{"})]\!]_{\Gamma}^{M} , \rho[x_M \mapsto \text{"}yL{:=}z_H\text{"}]>$$

$$\Rightarrow < [\![y_L{:=}z_H]\!]_{\Gamma}^{M} , \rho[x_M \mapsto \text{"}y_L{:=}z_H\text{"}]>$$

$$\Rightarrow < \text{if } (M \vee H \leq L) \text{ then } y_L{:=}z_H \text{ else loop} , \rho[x_M \mapsto \text{"}y_L{:=}z_H\text{"}]>$$

$$\Rightarrow < \text{loop} , \rho[x_M \mapsto \text{"}y_L{:=}z_H\text{"}]> \Rightarrow \ldots$$

Inlining of programs is sound (w.r.t. Deterministic Input-Output NI)

- Theorem:
  For every program S, security level l and memories ρ_1 and ρ_2 such that $\rho_1 \sim_l \rho_2$, we have that
  $< [\![S]\!]_{\Gamma^L}, \rho_1 > \Rightarrow^* \rho_1'$ and

  $< [\![S]\!]_{\Gamma^L}, \rho_2 > \Rightarrow^* \rho_2'$ implies $\rho_1' \sim_l \rho_2'$.

Conclusion

- Dynamism of web applications puts higher demands on the permissiveness of the security mechanism: hence the importance of dynamic analysis.
- We have formally defined two monitors that enforce Noninterference in the presence of dynamic code evaluation.
- Information flow control is a promising approach for preventing vulnerabilities in Web applications.

An Information Flow Monitor-Inlining Compiler for Securing a Core of JavaScript

by José Fragoso Santos and Tamara Rezk (2014)

Abstract. Web application designers and users alike are interested in isolation properties for trusted JavaScript code in order to prevent confidential resources from being leaked to untrusted parties. Noninterference provides the mathematical foundation for reasoning precisely about the information flows that take place during the execution of a program. Due to the dynamicity of the language, research on mechanisms for enforcing noninterference in JavaScript has mostly focused on dynamic approaches. We present the first <u>information flow monitor inlining compiler for a realistic core of JavaScript</u>. We prove that the proposed compiler enforces termination-insensitive noninterference and we provide an implementation that illustrates its applicability.

BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML

by C. Reis et al. (2006)

Abstract. (...) The dynamic content we target is the dynamic HTML in web pages, which have become a popular vector for attacks. The key challenge in filtering dynamic HTML is that it is undecidable to statically determine whether an embedded script will exploit the browser at run-time. We avoid this undecidability problem by <u>rewriting web pages and any embedded scripts into safe equivalents, inserting checks so that the filtering is done at run-time</u>. The rewritten pages contain logic for recursively applying run-time checks to dynamically generated or modified web content, based on known vulnerabilities. We have built and evaluated BrowserShield, a system that <u>performs this dynamic instrumentation of embedded scripts</u>, and that admits policies for customized run-time actions like vulnerability-driven filtering.