IBOS: A correct-by-construction modular browser

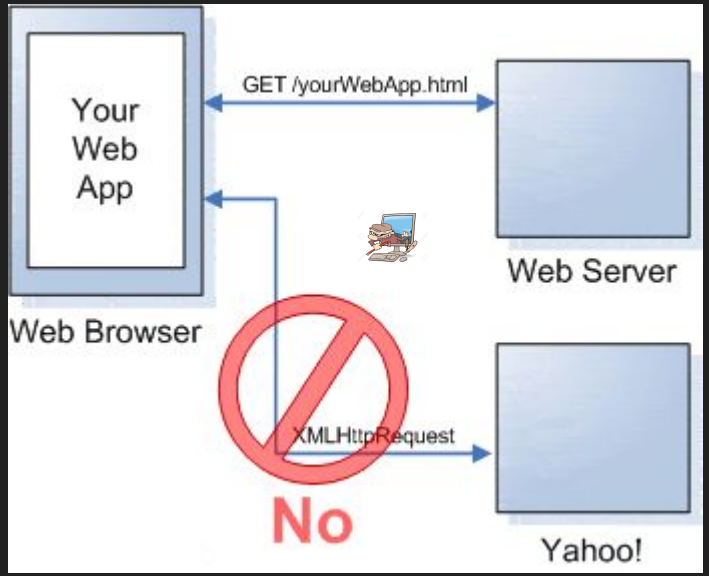
Ralf Sasse, Samuel T. King, José Meseguer, Shuo Tang

Motivation

- Browsers are de-facto OS for webapps
- Increasing functionality leads to technical debt - problem with sensitive data
- Ideally functionality and security are separated into the browser and a browser kernel
- Even still, browser must trust underlying OS
- Solution is to include basic OS code in Trusted Computing Base (TCB)
    - Small microkernel which is verified + small browser kernel which is verified
    - L4Ka microkernel not formally verified, but a replaceable part

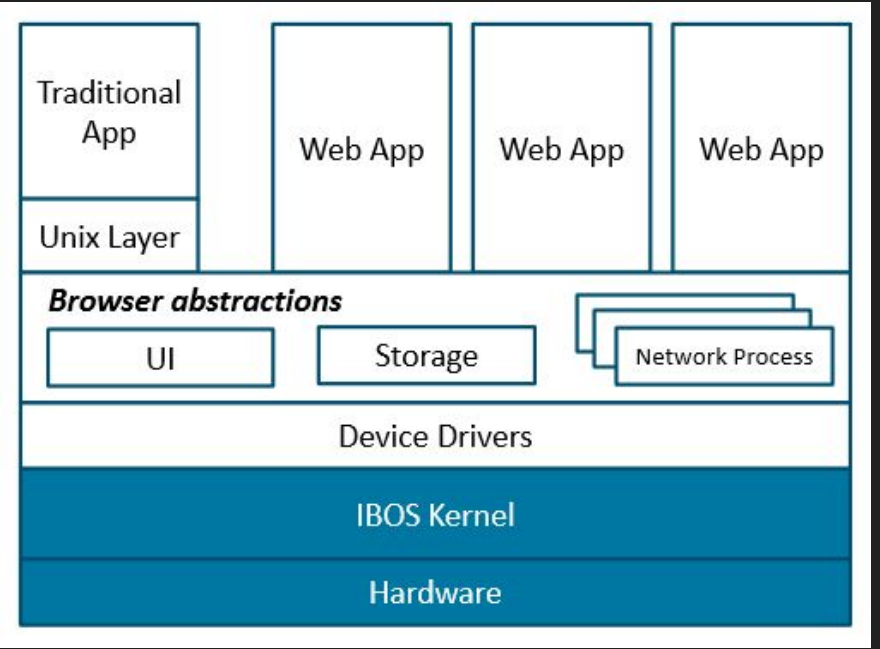
Background

- Same Origin Policy
  - 7 rules about data transfer that guarantee isolation of browser tabs across domains
  - Web apps from different origins are isolated from each other
- Address Bar Correctness
  - Page content displayed is the same as the address in the address bar



Architecture

- IBOS kernel
  - Process creation, memory management, message passing
- Network process
  - HTTP requests, passes data to and from Network Interface Card
  - One network process per unique site
- Web app
  - One app per page, represents the rendered tab
  - Multiple web apps can talk to one network process, only if same site



Security Goals

1. The kernel must route network requests from web page instances to the proper network process.

2. The kernel must route Ethernet frames from the network interface card (NIC) to the proper network process.

3. Ethernet frames from network processes to the NIC must have an IP address and TCP port that matches the origin of the network process.

4. HTTP data from network processes to web page instances must be from acceptable origins.

5. Network processes for different web page instances must remain isolated.

6. Isolation of the browser chrome (UI elements) and web page content displays.

7. Only the current tab can access the screen, mouse, and keyboard.

8. All components can only perform their designated functions.

9. The URL of the current tab is displayed to the user.

IBOS Specification - Sample Initial State

```
< ui | toKernel(msg(ui,webapp, MSG-NEW-URL, url("0")) @ mt) >
< display | activeWebapp(none), displayedContent(about-blank) >
< nic | in(mtLL), nic-out(mtLS) >
< webappmgr | nextWPN(0) >
< kernel |
    weblabels(mtWPIS), netlabels(mtNPIS),
    displayedTopBar(url("0")),
    handledCurrently(none),
    nextNPN(0),
    msgPolicy(policy(webapp, network,MSG-FETCH-URL ),ps
              policy(ui,     webapp, MSG-NEW-URL   ),ps
              policy(ui,     webapp, MSG-SWITCH-TAB),ps
              policy(network,webapp, MSG-RETURN-URL)) >
```

Example: New URL Rule

```
rl [new-url] :
   { < kernel    |
        handledCurrently(msg(ui, webapp, MSG-NEW-URL, URL)),
        displayedTopBar(L),
        weblabels(WPIS),
        Att >
     < display   |
        activeWebapp(MWI),
        displayedContent(L'),
        Att2 >
     < webappmgr | nextWPN(N), Att3 > Cnf }
```

```
{ < kernel    |
     handledCurrently(none),
     displayedTopBar(URL),
     weblabels(pi(webapp(N), URL),wp WPIS),
     Att >
  < display   |
     activeWebapp(webapp(N)),
     displayedContent(about-blank),
     Att2 >
  < webappmgr | nextWPN(N + 1), Att3 >
  < webapp(N) |
     rendered(about-blank),
     URL(URL),
     loading(false),
     fromKernel(mt),
     toKernel(mt) > Cnf } .
```

Example: Kernel Messages

```
rl [kernelReceivesOPMessage] :
   < kernel-id : kernel |
       handledCurrently(mt) , msgPolicy(MP), Att >
   < ID : pipe | toKernel(msg(ST:SyscallType,
       payload(N, N', M:MsgType, S:String)), ML) , Att2 >
=> < kernel-id : kernel |
      handledCurrently(policyAllows(msg(ST:SyscallType,
        payload(ID, N', M:MsgType, S:String)), MP)) ,
      msgPolicy(MP), Att >
  < ID : pipe | toKernel(ML) , Att2 > .
```

Formal Verification - Trigger / Internal Rules

Lemma 1. *Given terms s_1 and s_2, for any chain of rewrites of the form $s_1 \rightarrow^*_{(T \cup I)/E} s_2$, with n uses of trigger rule, we can rearrange that sequence, using the same rewrites, to $s_1 \rightarrow^1_{T/E} \rightarrow^!_{I/E} \cdots \rightarrow^i_{T/E} \rightarrow^!_{I/E} \cdots \rightarrow^n_{T/E} \rightarrow^*_{I/E} s_2$.*

- Split rules into two types, trigger and internal
- A trigger rule is any rule that takes an action (new url or switch tab)
- Internal rules are the rest of the rules
- We can normalize the state by executing internal rules until there are no more rewrites possible
- Run trigger rule followed by normalization repeatedly, yields same result as any order of trigger and internal rules (because they are independent)

Formal Verification - Address Bar Correctness

```
op inspect : -> Cmd .
op inspect : Nat -> Cmd .
rl inspect => inspect(3) .
rl inspect(0) => mtCmdList .
rl inspect(s(N:Nat)) => new-url , inspect(N:Nat) .
rl inspect(s(N:Nat)) => switch-tab , inspect(N:Nat) .
```

- A trigger rule corresponds to some action taken (new url or switch tab).
- Inspect generates all possible sequences of trigger rules
- Use model checking to verify all possible sequences of 3 trigger rules have address bar correctness
- Argue if there is a violation in 4 or more trigger rules, we can reduce it to a violation using only 3 trigger rules, which we know can’t exist

Search for Mismatched Network Proc and Web App

```
search init-simp-kernel inspect-space =>*
X:Configuration < N:Nat : pipe |  toKernel(ML:MessageList) ,
   fromKernel(msg(OPOS-SYSCALL-FD-SEND-MESSAGE,
         payload(Num:Nat, N:Nat, MSG-FETCH-URL, L1:Label)),
         ML':MessageList) , Att:AttributeSet >
< kernel-id : kernel | Att2:AttributeSet ,
     weblabels(pi(Num:Nat,L1':Label), WAPIS:WebappProcInfoSet) ,
     networklabels(pi(N:Nat, L2':Label, L2:Label),
                   NPIS:NetworkProcInfoSet)  ,
     displayedTopBar(URL:Label) >
such that  L1:Label =/= L2:Label or L1':Label =/= L2':Label .
```

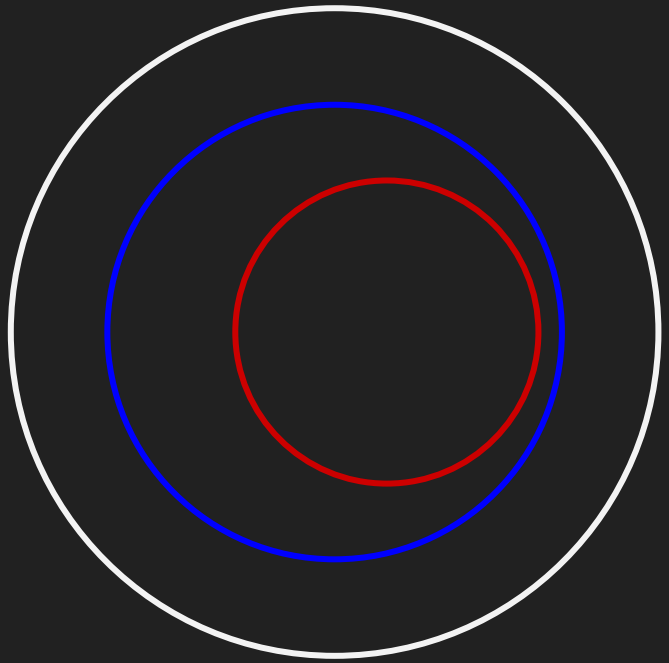
Verification of the IBOS Browser Security Properties in Reachability Logic

Stephen Skeirik, José Meseguer, Camilo Rocha

Motivation

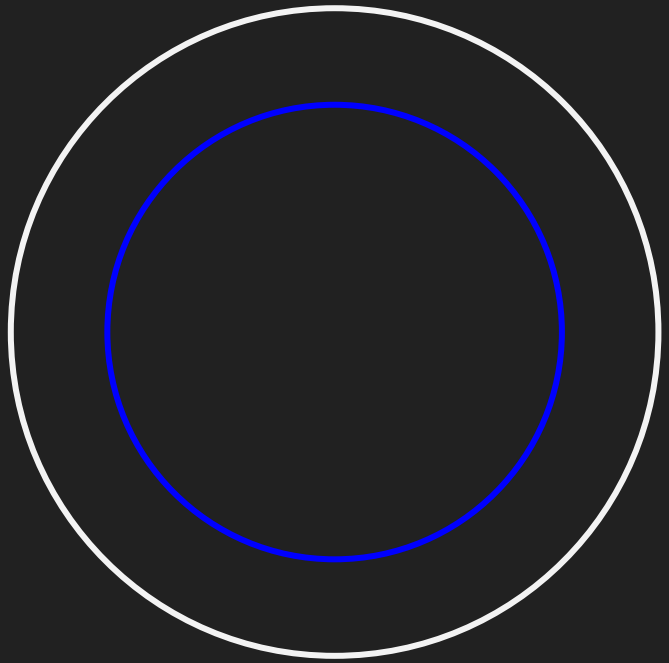
- Sasse’s paper uses a hand-made proof to complete the ABC and SOP invariants
- Instead, create an inductive invariant strong enough to be true for all rewrite rules
- Once we have an inductive invariant, can use reachability logic prover

Invariants



- Red = Reachable from initial states
- Blue = Invariant
- White = All states
- All reachable states are inside invariant

Invariants



- Blue = Inductive Invariant
- White = All states
- If a state is inside the invariant, then any following state will also be inside the invariant
- Given I is true, after any rewrite, I is still true
- Assuming our program can get to a state where I is true, I will stay true

Creating Inductive Invariant

$$\{\langle kernel \mid addrBar(U), Atts\rangle \quad \langle display \mid displayContent(U'), Atts'\rangle \; Conf\} \mid U' \trianglelefteq U \wedge WF(\ldots)$$

- Initial requirements:
- Address bar contains U
- Display contains U’
- U’ is either blank or equal to U
- The state is well formed
- Proof fails (invariant is not strong enough to prove itself under every rewrite rule)

Creating Inductive Invariant

$$\{\langle kernel \mid addrBar(U), Atts\rangle \langle display \mid displayContent(U'), activeTab(WA), Atts'\rangle \langle WA \mid URL(U''), Atts''\rangle\ Conf\} \mid U' \trianglelefteq U \wedge U \trianglelefteq U'' \wedge WF(\ldots)$$

- Web app exists (WA)
- WA is active
- Displayed content related to URL
- URL related to web app URL
- Not strong enough still

Why It’s Failing

```
crl [change-display] :
    { < display | activeWebapp(WI), displayedContent(L),  Att2 >
      < WI      | rendered(L'),                           Att3 > Cnf }
 => { < display | activeWebapp(WI), displayedContent(L'), Att2 >
      < WI      | rendered(L'),                           Att3 > Cnf }
  if L ~l L' = false .
```

- In change-display, we have a webapp with rendered content
- Our invariant does not state anything about rendered content in a webapp
- This means we are replacing displayedContent with an unknown value so we cannot say the invariant still holds

Creating Inductive Invariant

$$R(\langle WA \mid rendered(U), URL(U'), Atts\rangle\ Conf) = U \trianglelefteq U' \wedge R(Conf) \qquad (R_1)$$

$$R(\langle P \mid Atts\rangle\ Conf) = R(Conf)\ if\ \neg WA(P) \qquad (R_2)$$

$$R(none) = \top \qquad (R_3)$$

- All web apps have rendered content equal to URL
- Important if a rewrite rule uses the rendered content, otherwise we have no assertions about what the rendered content could be

Creating Inductive Invariant

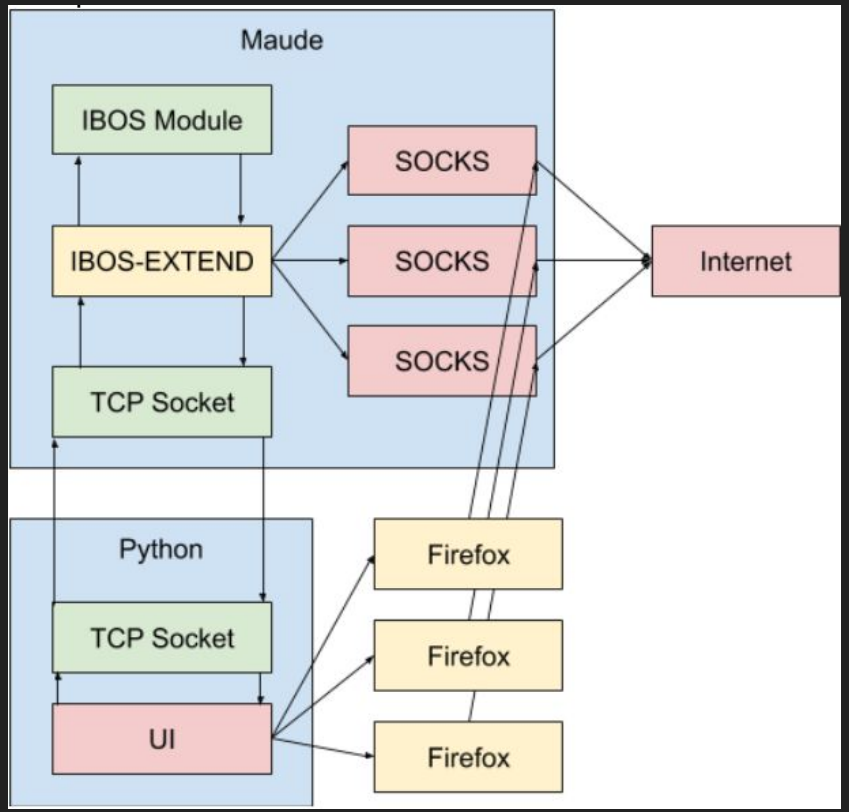
$$\{\langle kernel \mid addrBar(U), Atts\rangle \langle display \mid displayContent(U'), activeTab(WA), Atts'\rangle \langle WA \mid URL(U''), Atts''\rangle \; Conf\} \mid U' \trianglelefteq U \wedge U \trianglelefteq U'' \wedge WF(\ldots) \wedge R(\ldots)$$

- Add in R(...), ensuring rendered content relates to URL
- Proof succeeds
- Adding these statements allows us to assert the invariant is true for all rewrite rules
  - Ex: Without rendered content matching URL, we can’t verify a rule transferring the content from web app to display

Our Project

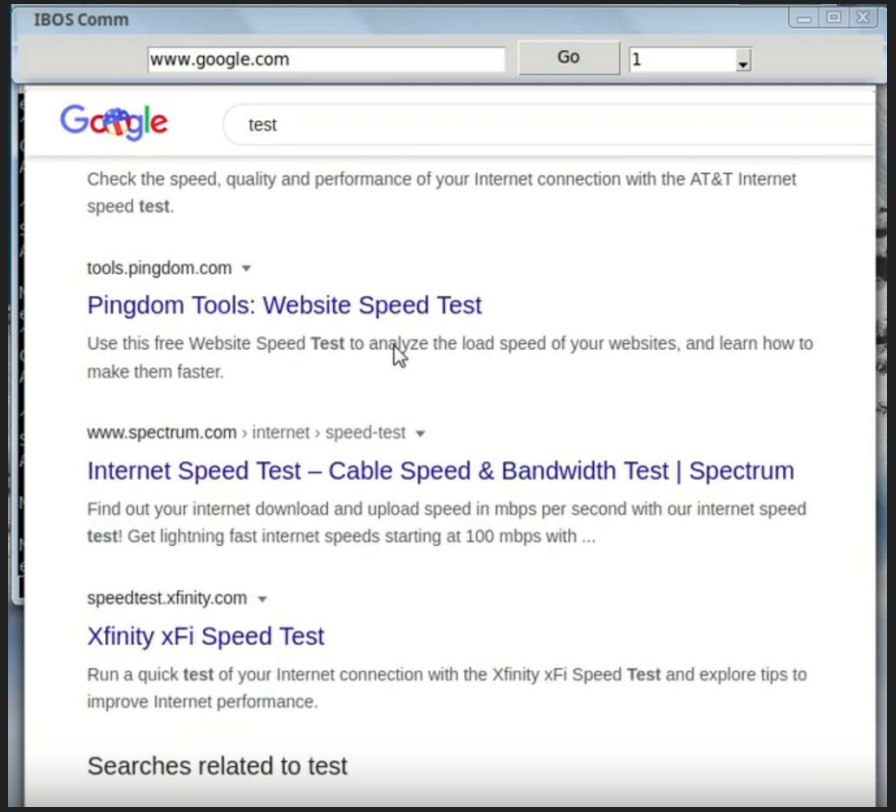
- Add a new module IBOS-EXTEND
  - Opens a TCP socket from maude to python
  - Use erewrite instead of rewrite (external objects)
- Create SOCKS module, which is a proxy server with a whitelist
- Create firefox instance piped to a SOCKS proxy, representing web apps and network procs from IBOS
- Python code displays UI by asking maude

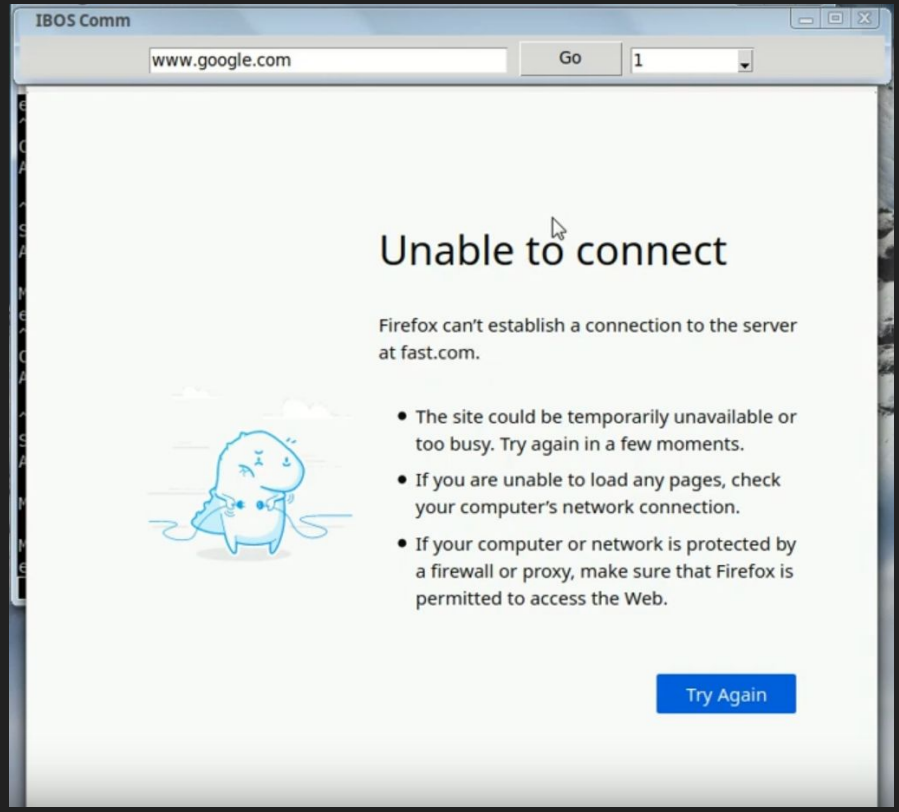
Our Project



- SOCKS servers created inside maude, free and easy parallelism due to using rewrite rules
- Firefox processes have network calls hijacked using socksify CLI
- Now working on model checking the system to verify IBOS goals still hold

Our Project


IBOS Comm
www.google.com
Go
1
Google
test
Check the speed, quality and performance of your Internet connection with the AT&T Internet speed test.
tools.pingdom.com
Pingdom Tools: Website Speed Test
Use this free Website Speed Test to analyze the load speed of your websites, and learn how to make them faster.
www.spectrum.com › internet › speed-test
Internet Speed Test – Cable Speed & Bandwidth Test | Spectrum
Find out your internet download and upload speed in mbps per second with our internet speed test! Get lightning fast internet speeds starting at 100 mbps with ...
speedtest.xfinity.com
Xfinity xFi Speed Test
Run a quick test of your Internet connection with the Xfinity xFi Speed Test and explore tips to improve Internet performance.
Searches related to test


IBOS Comm
www.google.com
Go
1
Unable to connect
Firefox can't establish a connection to the server at fast.com.
The site could be temporarily unavailable or too busy. Try again in a few moments.
If you are unable to load any pages, check your computer's network connection.
If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.
Try Again