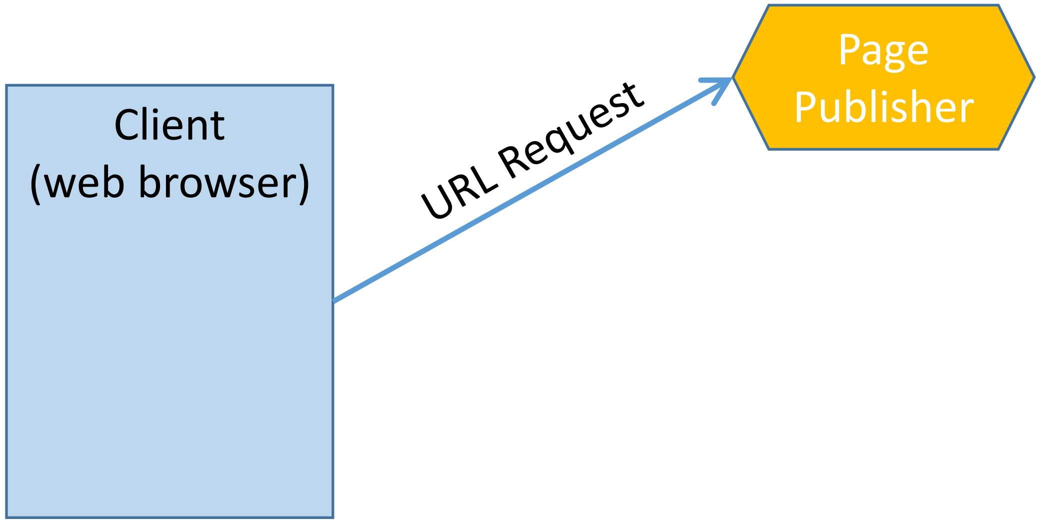
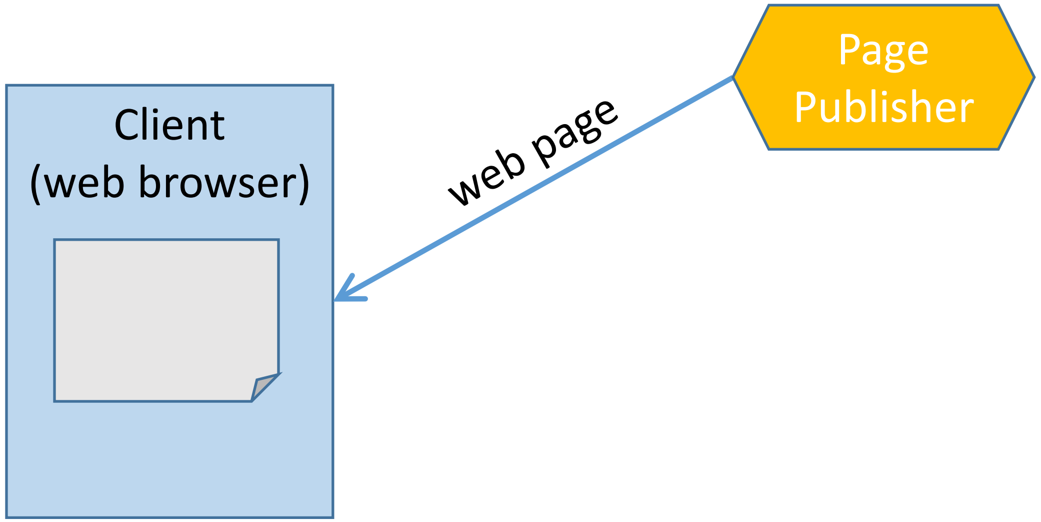
Between Worlds: Securing Mixed JS/AS Multi-Party Web Content

Phu H. Phung, Maliheh Monshizadeh, Meera Sridhar, Kevin W. Hamlen, and V.N. Venkatakrishnan

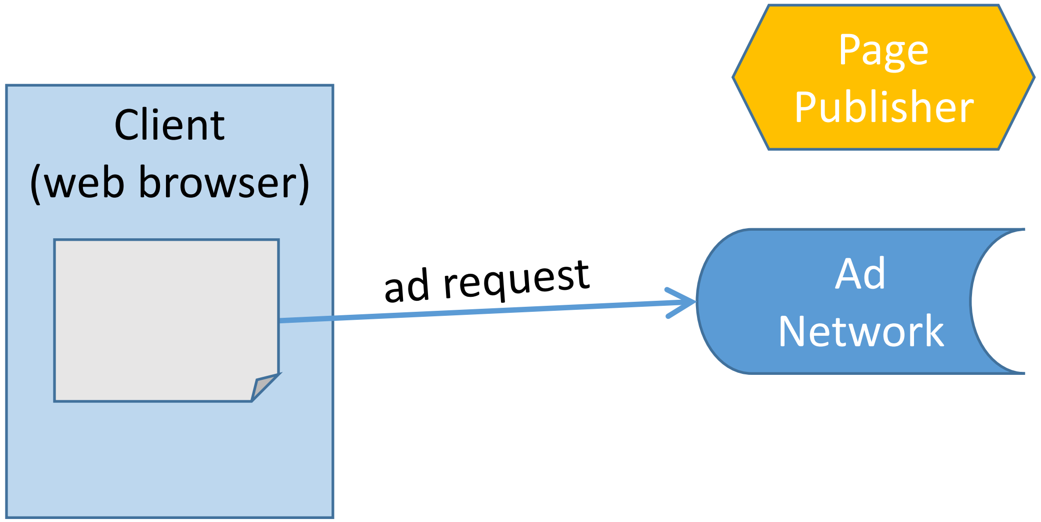
Loading a Web Advertisement



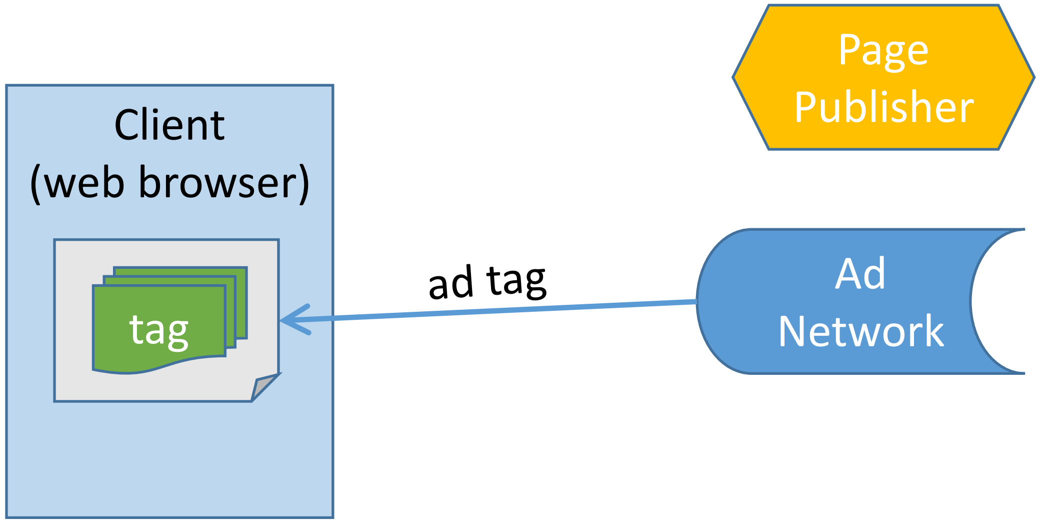
Loading a Web Advertisement



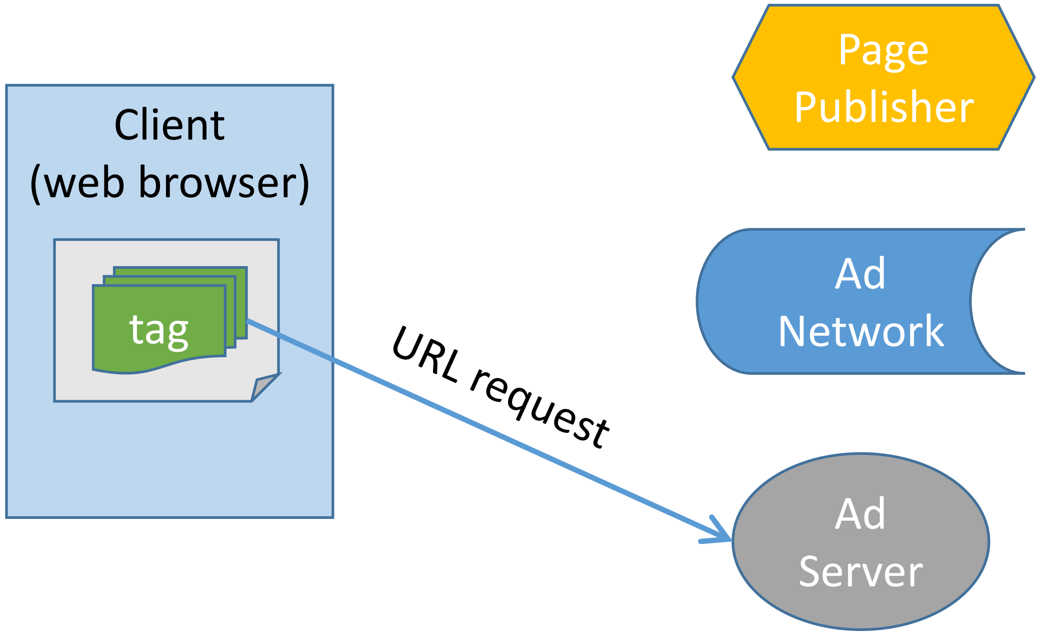
Loading a Web Advertisement



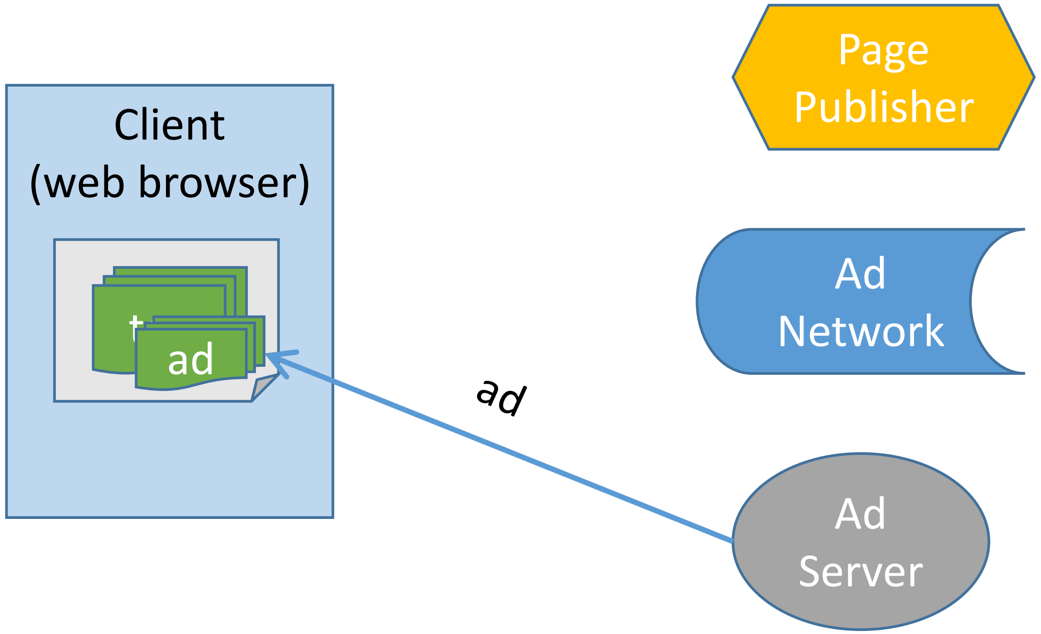
Loading a Web Advertisement



Loading a Web Advertisement



Loading a Web Advertisement



Ads Behaving Badly

- Many well-known language-specific (e.g., JS/Flash) attacks
  - invisible iframe expansion (JS)
  - DOM API hijacking (JS)
  - malformed binary that exploits VM parser error (Flash)
- A newly emerging class of attacks: cross-domain attacks
  - Many ads are part JS and part Flash – opens new attack vectors
  - SOP Circumvention: JS and Flash have *different* Same-Origin Policies!
    - not easily reconcilable, since computation models differ between languages
  - Cross-domain heap-spraying attacks
    - separate payload injector from payload execution across different languages
  - Cross-principal resource abuse
    - Flash ads use allowDomain(“*”) (!!!)

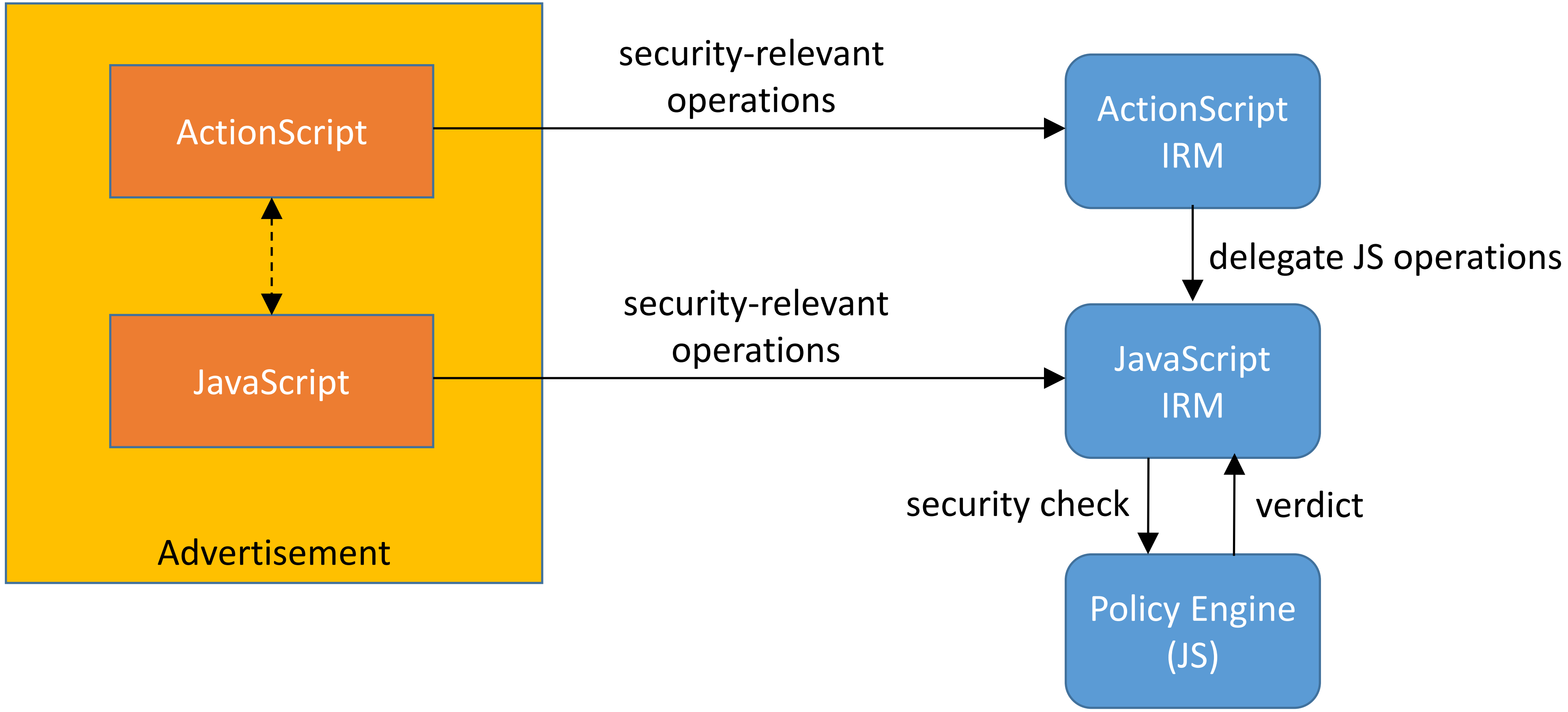
Non-LBS Approaches

- Turn off JS/Flash/both
  - kills the revenue model of the internet
- Change JS/Flash VMs and/or browser to fix loopholes and weaknesses
  - requires cooperation and standardization of all client browsers and VMs
  - requires all end-users to update their browsers
- Adopt best coding practices when creating ads
  - assumes ad-creators know anything about coding
- Validate ads at the ad network level
  - Ad networks often never see the ad that the end-user sees!
- What if I’m a page-publisher and I want to protect my visitors, irrespective of which client browsers they may be using? How do I secure my page?

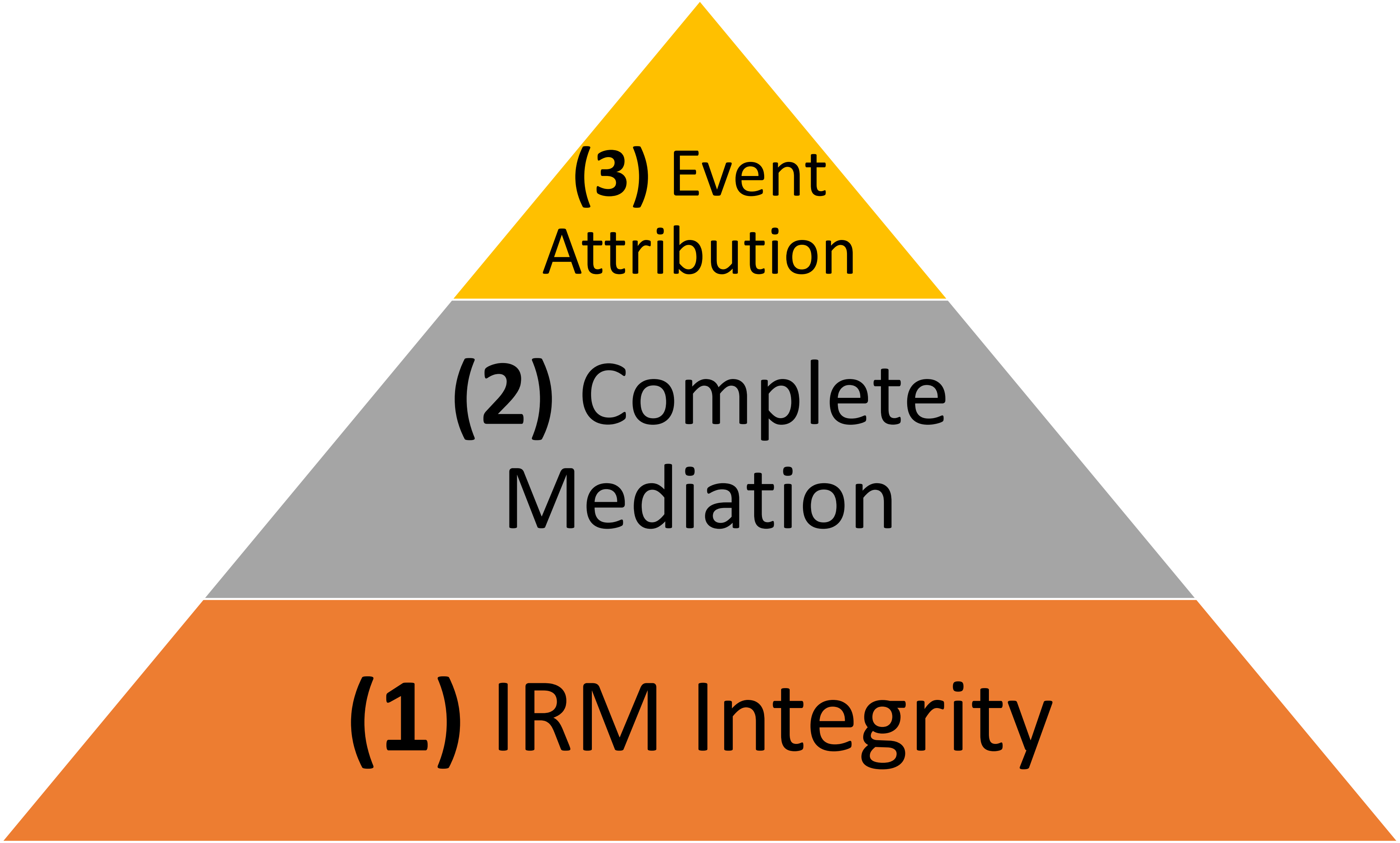
LBS Approach: In-lined Reference Monitors

- Idea: Page-publisher puts a script on her page that rewrites and secures ad code dynamically, as it arrives on the end-user's browser!
  - no change to browsers or VMs required
  - no separate, special software installations for end-users (e.g., no plug-ins)
  - browser-agnostic (use purely standards-compliant JS/Flash code)
  - can enforce *publisher-specific* policies
    - Example: no pop-ups allowed on pages where the page's menu is a pop-up
- Challenges:
  - JS is incredibly dynamic (code constantly generated from strings)
  - JS-Flash interaction is very insecure—hard to completely mediate
  - JS is absurdly mutable (can destructively assign to DOM API functions!!)

FlashJaX Architecture and Workflow



FlashJaX Security Foundations



JavaScript IRM Integrity: Anonymous Closures

```
(function(){
        var principal = “bottom”;
        getPrincipal = function() { return principal; }
}) ();

y = getPrincipal();          // assigns y:=“bottom”;
principal = “root”;          // error: no such variable “principal”!
```

Complete Mediation: Preemptively Hijack the DOM!

```
(function(){
    var principal = "bottom";
    getPrincipal = function() { return principal; }
    var wrap_window = function(w) {
        var o_open = w.open;
        w.open = function() {
            if (isAllowed(principal, "open", arguments))
                return wrap_window(o_open.apply(this, arguments));
            else return null;
        }
        return w;
    }
    wrap_window(window);
}) ();
```

Event Attribution: Shadow Stack of Principals

```
(function(){
        var shadowStack = [];
        …
        var runAs = function(principal, f) {
                shadowStack.push(principal);
                f.apply = js.Function.apply;            // un-hijack f.apply…!
                var r = f.apply(this, js.Aray,prototype.slice.call(arguments, 2));
                shadowStack.pop();
                flush_write(principal);                 // handles runtime code gen
                if (typeof r !== “undefined”) return r;
        }
        …
}) ();
```

Attribution Challenge: Dynamic Code Generation

- Which principal to pass to runAs(principal,f) for each f?
- Static Scripts
  - Publisher labels html subtrees that she “owns” as trusted
  - Publisher labels ad network code blocks as untrusted
  - Multiple ad networks can have mutually distrusting labels (to stop wars)
- Problem: What about runtime generated code?
  - JS scripts regularly generate code from *strings* at runtime (ugh!)
  - Most common (and most general) method: document.write(s);

HTML Document Load Process (simplified)

```
<html>
    <script>
        alert('hello ');
        s = "script>";
        document.write("<"+s+"alert('cruel');</"+s);
    </script>
```

Input Stream (from web server):

```
<html><script>alert('hello ');s="s
```

Output:

HTML Document Load Process (simplified)

```
<html>
    <script>
        alert('hello ');
        s = "script>";
        document.write("<"+s+"alert('cruel');</"+s);
    </script>
```

Input Stream (from web server):

```
<script>alert(' world');</script>…
```

Output:

hello

HTML Document Load Process (simplified)

```
<html>
    <script>
        alert(‘hello ’);
        s = “script>”;
        document.write(“<“+s+”alert(‘cruel’);</”+s);
    </script>
    <script> alert(‘cruel’); </script>
```

Input Stream (from web server):

```
<script>alert(‘cruel’);</script><sc
```

Output:

hello

HTML Document Load Process (simplified)

```
<html>
    <script>
        alert(‘hello ’);
        s = “script>”;
        document.write(“<“+s+”alert(‘cruel’);</”+s);
    </script>
    <script> alert(‘cruel’); </script>
    <script> alert(‘ world’); </script>
```

Input Stream (from web server):

```
<script>alert(‘ world’);</script>…
```

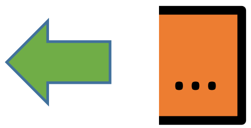
Output:

hello cruel

HTML Document Load Process (simplified)

```
<html>
    <script>
        alert(‘hello ’);
        s = “script>”;
        document.write(“<“+s+”alert(‘cruel’);</”+s);
    </script>
    <script> alert(‘cruel’); </script>
    <script> alert(‘ world’); </script>
```

Input Stream (from web server):



Output:

hello cruel world

Dynamic Codegen Challenges

- First step: Replace document.write with a wrapper
  - use DOM API hijacking again (same as mediation approach)
- But what should the wrapper do?
  - must parse a string into JavaScript code
    - (build our own HTML+JS parser in JS? ugh!)
  - What if the dynamically generated code generates more code dynamically when executed?
    - Turns out almost every ad network actually does this!
- Can’t ignore it – almost all ad networks depend on it and use it

Dynamic Code Generation Solution

```
old_write = document.write;
document.write = function(s) { write_buffer[principal] += s; }        // buffer the writes!

var flush_write = function(principal) {
        var i = document.createElement("ins");
        i.innerHTML = write_buffer[principal];        // invoke the browser's parser!
        write_buffer[principal] = "";
        foreach script element e within i do {
                var newScript = makeFunction(e.textContent);
                e.textContent = "";
                runAs(principal, newScript);
        }
        i.owner = principal;
        document.lastChild.appendChild(i);   // append i to page (without running scripts)
}
```

Attack Scenarios Tested

Attack Scenario	Policy Enforced by FlashJaX
Flash Circumvention of SOP	Principal-specific whitelisting policy
Cross-language Heap-spray Attack	Resource bound policy on heap writes
Cross-Principal Resource Abuse	Principal-specific access control
Wrapper Vulnerabilities	DOM API Aliasing Detection
Confidentiality and Integrity Violations	Principal-specific, fine-grained access control of page real-estate and data
Ad-specific Attacks	Various (see paper)