


Sandboxing Untrusted JavaScript and other web security stories

John Mitchell
Stanford University


Stanford Computer Security Lab



Some topics I'm working on

- Computing on encrypted data
- Information flow: Dynamic IFC in Haskell, web
- Third-party web tracking
- Practical web security (tools, methods,...)
- Android malware scanning
- Machine learning for CAPTCHAs, security
- Online learning

Seek postdoc with theoretical, practical skills




Outline for Today, Tomorrow

- Background: computer security
- Web fundamentals and browser security
- JavaScript isolation
 - How can trusted and untrusted code be executed in the same environment, without compromising functionality or security?
- Three parts
 - Isolate untrusted application from hosting page
 - Isolate one untrusted application from another
 - Mediated access: reference monitor for critical resources
- Additional topics
 - Operational semantics (covered between parts I and II)
 - Foundations for Web security



Computer Security




Computer Security

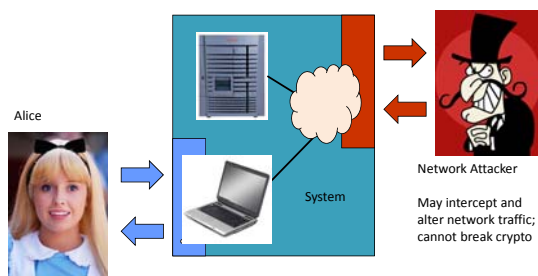
- Security model
 - A system of interest
 - Desired properties of the system
 - Interface and capabilities of an attacker
- Security analysis
 - Can system design and security mechanism it includes guarantee desired the properties, in spite of attacker?

$$\text{Secure}(\text{Sys}, \text{Prop}, \text{Threat}) = \forall U \in \text{UserIn}. \forall A \in \text{Threat}. \forall \text{Runs} \in \text{Sys}(A, U). \text{Prop}(\text{Runs})$$

Inherently analytical problem; not determined by testing



Network security

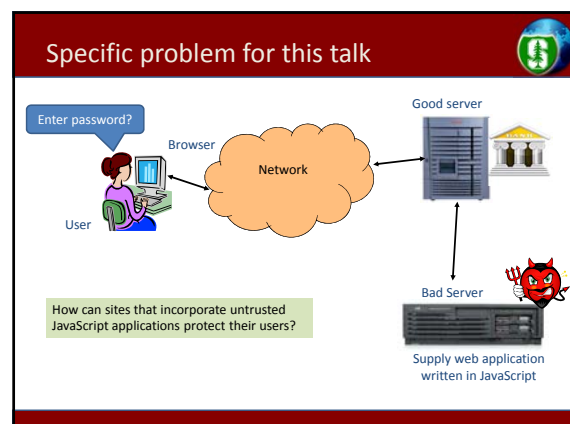
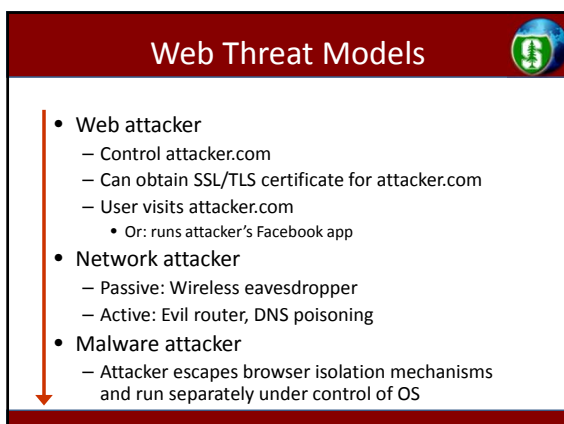
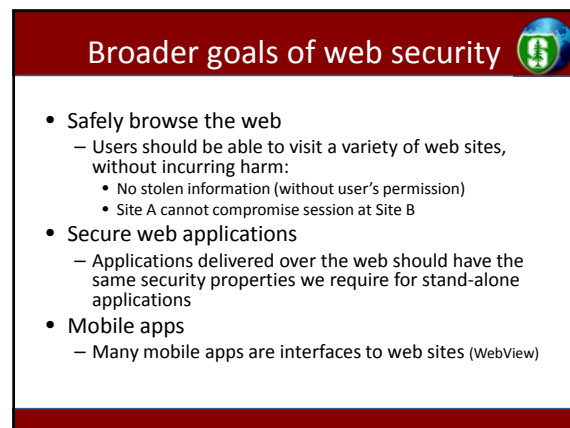
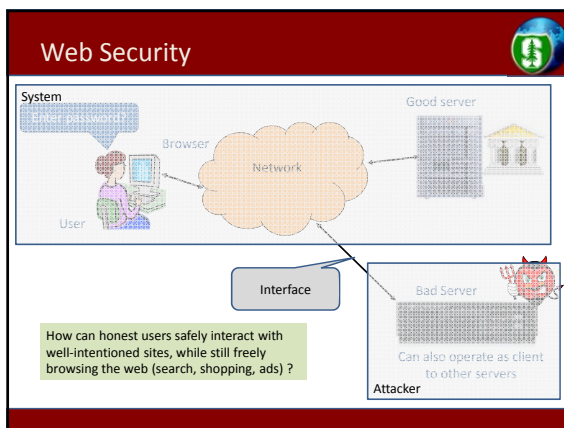
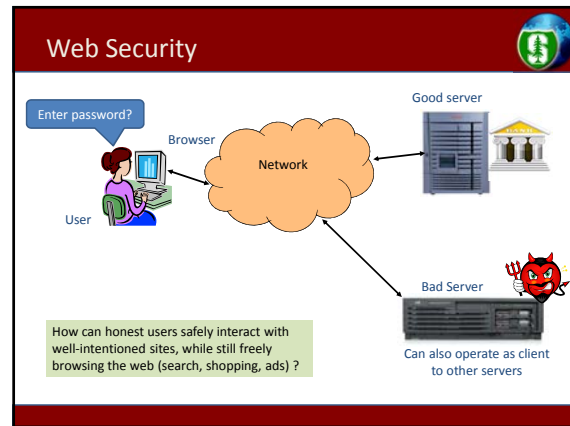
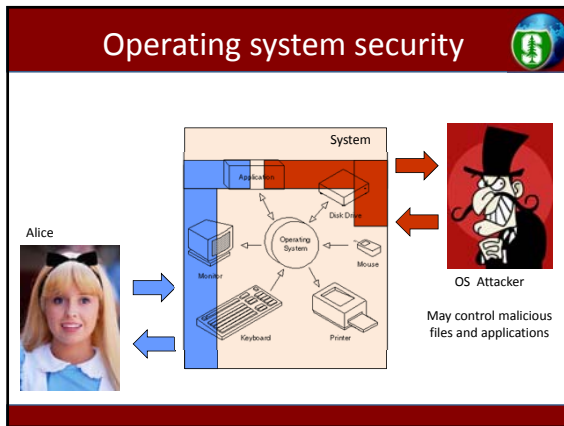


Alice

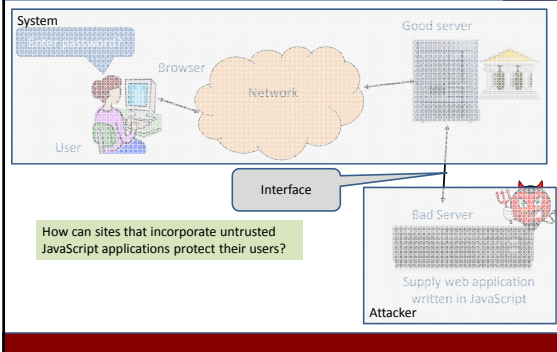
System

Network Attacker

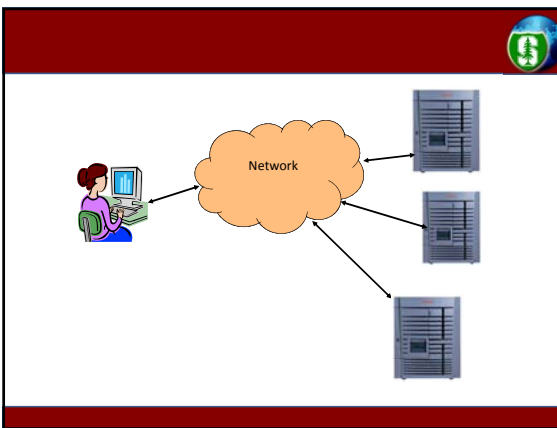
May intercept and alter network traffic; cannot break crypto



Specific problem for this talk



Web fundamentals and browser security



Network request and response

Uniform Resource Locator (URL)

- Global identifier of network-retrievable content
- Example:**
<http://stanford.edu:81/class?name=cs155#homework>

Protocol: http
 Hostname: stanford.edu
 Port: 81
 Path: /class
 Query: ?name=cs155
 Fragment: #homework

- Special characters are encoded as hex:
 - %0A = newline
 - %20 or + = space, %2B = + (special exception)

HTTP Request

| Method | File | HTTP version | Headers |
|--------|-------------|--------------|---|
| GET | /index.html | HTTP/1.1 | Accept: image/gif, image/x-bitmap, image/jpeg, */* Accept-Language: en Connection: Keep-Alive User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95) Host: www.example.com Referer: http://www.google.com?q=dinbats |

Blank line

Data – none for GET

GET : no side effect POST : possible side effect

HTTP Response

HTTP version Status code Reason phrase Headers Data

```

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543
<HTML> Some data... blah, blah, blah </HTML>
  
```

Cookies

Rendering Content

Displaying a page

```

<head>
<title>Washington Post: Breaking News, World, US, DC News .. Analysis</title>
...
</head>
<body class="eidos homepage sectionfront">
  <script type="text/javascript">
    if(self==top&&(top.window.location.pathname).startsWith('/PortalEdito
r')){top.location=self.location;}
  </script>
  ...
  <h2 class="headline"><a href="/world/national-security/nsa-gathered-
thousands-of-americans-e-mails-before-court-struck-down-
program/2013/08/21/146ba4b6-0a90-11e3-b87c-476db8ac34cd_story.html">
Secret court: <br>NSA gathered thousands of domestic e-mails</a>
  ...
  <p class="byline">Ellen Nakashima</p>
  <p class="">
The program unlawfully gathered as many as tens of thousands of e-mails,
according to a 2011 opinion.</p>
  ...
  <div class="hide"></div>
  ...
  Share this video:
  ...
  <div class="facebook_static">
onclick="TWP.Module.SocialButtons.staticSocialPopup('http://www.facebook.com/
sharer.php?u=http://www.washingtonpost.com/posttv/video/thefold/tonight-on-
the-fold-august-21-2013/2013/08/21/36ed282c-0a98-11e3-9941-
6711ed662e71_video.html&3Ffb_ref=3Dom_btn_fb')">
  
```

HTML Image Tags

```


  
```

- Requests, receives, and displays a picture
 - Security issues?

Image tag security issues

- Communicate with other sites
 - ``
- Hide resulting image
 - ``
- Spoof other sites
 - Add logos that fool a user

Important Point: A web page can send information to any site

Browser execution model

- Each browser window or frame
 - Loads content
 - Renders it
 - Processes HTML and scripts to display page
 - May involve images, subframes, etc.
 - Responds to events
- Events can be
 - User actions: OnClick, OnMouseover
 - Rendering: OnLoad, OnBeforeUnload
 - Timing: setTimeout(), clearTimeout()

Document Object Model (DOM)

- Object-oriented interface
 - web page in HTML is structured data
 - DOM provides representation of this hierarchy
- Examples
 - **Properties:** document.alinkColor, document.URL, document.forms[], document.links[], document.anchors[]
 - **Methods:** document.write(document.referrer)

Changing HTML using Script, DOM

- Some possibilities
 - createElement(elementName)
 - createTextNode(text)
 - appendChild(newChild)
 - removeChild(node)
- Example: Add a new list item:

HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

```
var list = document.getElementById('t1')
var newItem = document.createElement('li')
var newText = document.createTextNode(text)
list.appendChild(newItem)
newItem.appendChild(newText)
```

JavaScript onError

- Basic function
 - Triggered when error occurs loading a document or an image
- Example


```

```

 - Runs onError handler if image does not exist and cannot load

http://www.w3schools.com/jsref/jsref_onError.asp

JavaScript timing

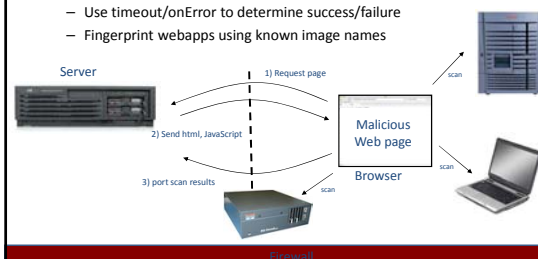
- Sample code


```
<html><body><img id="test" style="display: none">
<script>
var test = document.getElementById('test');
var start = new Date();
test.onerror = function() {
  var end = new Date();
  alert("Total time: " + (end - start));
}
test.src = "http://www.example.com/page.html";
</script>
</body></html>
```

 - When response header indicates that page is not an image, the browser stops and notifies JavaScript via the onerror handler.

Port scanning behind firewall

- JavaScript can:
 - Request images from internal IP addresses
 - Example:
 - Use timeout/onError to determine success/failure
 - Fingerprint webapps using known image names



Remote scripting

- Goal
 - Exchange data between a client-side app running in a browser and server-side app, without reloading page
- Methods
 - Java Applet/ActiveX control/Flash
 - Can make HTTP requests and interact with client-side JavaScript code, but requires LiveConnect (not available on all browsers)
 - XML-RPC
 - open, standards-based technology that requires XML-RPC libraries on server and in your client-side code.
 - Simple HTTP via a hidden IFRAME
 - IFRAME with a script on your web server (or database of static HTML files) is by far the easiest of the three remote scripting options

Important Point: A web can maintain bi-directional communication with browser (until user closes/quits)

See: <http://developer.apple.com/internet/webcontent/iframe.html>

Simple remote scripting example

client.html: RPC by passing arguments to server.html in query string

```
<script type="text/javascript">
function handleResponse() {
  alert("this function is called from server.html")
}
</script>
<iframe id="RSIFrame" name="RSIFrame"
  style="width:0px; height:0px; border: 0px"
  src="blank.html">
</iframe>
<a href="server.html" target="RSIFrame">make RPC call</a>
```

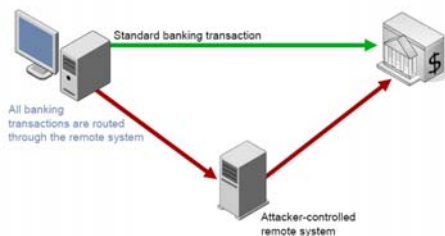
server.html: another page on same server, could be server.php, etc

```
<script type="text/javascript">
window.parent.handleResponse()
</script>
```

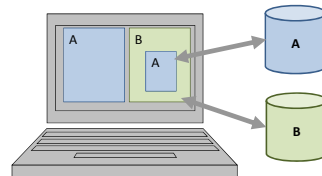
RPC can be done silently in JavaScript, passing and receiving arguments

Continuing trend

- Why ask the user to do something if you can write JavaScript to do it automatically?



Browser security mechanism



- Each frame of a page has an origin
 - Origin = protocol://host:port
- Frame can access its own origin
 - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin

Library import

```
<script src=https://seal.verisign.com/getseal?
  host_name=a.com></script>
```



- Embedded script has privileges of frame, NOT source server
- Can script other pages in this origin, load more scripts
- Other forms of importing



Analogy

Operating system

- Primitives
 - System calls
 - Processes
 - Disk
- Principals: Users
 - Discretionary access control
- Vulnerabilities
 - Buffer overflow
 - Root exploit

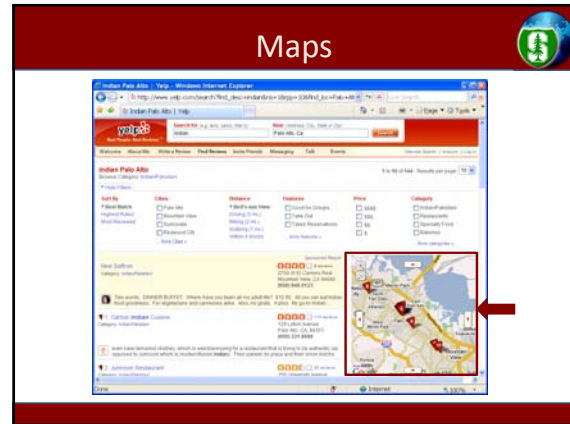
Web browser

- Primitives
 - Document object model
 - Frames
 - Cookies / localStorage
- Principals: "Origins"
 - Mandatory access control
- Vulnerabilities
 - Cross-site scripting
 - Cross-site request forgery
 - Cache history attacks
 - ...

Advertisements



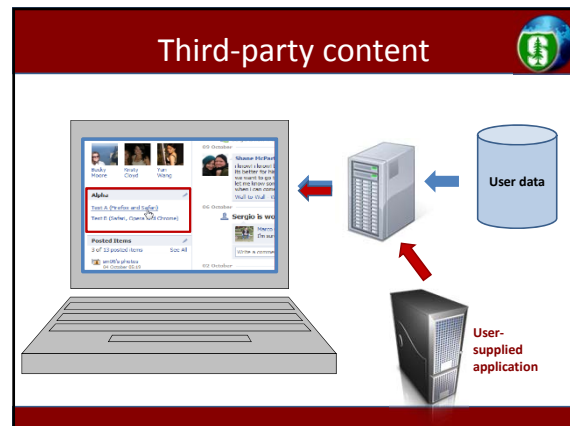
Maps



Social Networking Sites



Third-party content



Secure Web Mashups

• Challenge

- How can trusted and untrusted code be executed in the same environment, without compromising functionality or security?

• Approach

- Programming language semantics
 - Mathematical model of program execution
- Secure sublanguages and security tools
 - Filtering, Rewriting, Wrapping
 - Object-capability model
 - Improved JavaScript standards
 - Automated code analysis tool(s)

• Test cases and paradigms

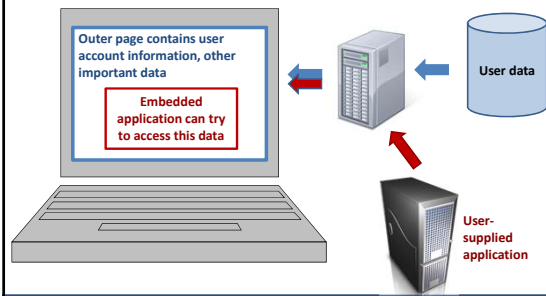
- Facebook JavaScript (FBJS)
 - Allow user-supplied applications
- Yahoo! ADSafe
 - Screen ad content before publisher
- Google Caja
 - Mathematical foundations of object-capability languages
 - Isolation, defensive consistency, ...



Hosting page isolation

Protect hosting page from untrusted applications (executed in the same browser frame)

Hosting page isolation



Facebook FBJS

- Facebook applications either “iframed” or integrated on page
 - We are interested in integrated applications
- Integrated applications are written in FBML/FBJS
 - Facebook subsets of HTML and JavaScript
 - FBJS is served from Facebook, after filtering and rewriting
 - Facebook libraries mediate access to the Document Object Model
- Security goals
 - No direct access to the Document Object Model (DOM)
 - No tampering with the execution environment
 - No tampering with Facebook libraries
- Basic approach
 - FBJS restricts “tricky” parts of JavaScript
 - Blacklist variable names that are used by containing page
 - Prevent access to global scope object, because variables are properties of scope objects

Four “FBJS” Theorems

- Theorem 1:** Subset $J(B)$ of ES-3 prevents access to chosen blacklist B (assuming $B \cap P_{\text{nat}} = \emptyset$)
- Theorem 2:** Subset $J(B)_G$ of $J(B)$ prevents any expression from naming the global scope object
- Theorem 3:** Subset $J(B)_S$ of $J(B)_G$ prevents any expression from naming any scope object
- Theorem 4:** A specific “wrapping” technique preserves Theorem 3 and allows previously blacklisted functions to be safely used

JavaScript Challenges

- Mutable objects with implicit self parameter:
 - `o = {b: function() {return this.a}}`
- Prototype-based object inheritance:
 - `Object.prototype.a = “foo”;`
- Scope can be a first-class object:
 - `this.o === o;`
- Can convert strings into code:
 - `eval(“o + o.b()”);`
- Implicit type conversions, which can be redefined.
 - `Object.prototype.toString = o.b;`

JavaScript can be tricky

- Which declaration of `g` is used?

```
var f = function(){ var a = g();
    function g() { return 1;};
    function g() { return 2;};
    var g = function() { return 3;};
    return a;
}
var result = f();           // has as value 2
```

- Implicit conversions

```
var y = “a”;
var x = {toString: function(){ return y;}}
x = x + 10;
js> “a10”                  // implicit call toString
```

- Use of *this* inside functions

```
var b = 10;
var f = function(){ var b = 5;
    function g(){ var b = 8; return this.b;};
    g();
}
var result = f();           // has as value 10
```

- String computation of property names

```
var m = “toS”; var n = “tring”;
Object.prototype[m + n] = function(){return undefined};
```

for (p in o){..., eval(...), o[S]}
allow strings to be used as code and vice versa

JavaScript modularity

- Modularity: variable naming and scope
- JavaScript local variables are not “local”
 - Activation records are objects
 - A program can get access to these objects
 - Properties (local variables) can be added, removed
 - These objects have prototypes
 - Properties (local variables) can be added, removed
- Traditional JavaScript (ECMA 2.6.2-3) does not support modularity with information hiding

Operational Semantics

[APLAS'08]

- Three semantic functions $\xrightarrow{e}, \xrightarrow{s}, \xrightarrow{P}$ for expressions, statements and programs.
- **Small step transitions**: A semantic function transforms one state to another if certain conditions (premise) are true.
- General form: $\frac{\text{Premise}}{S \xrightarrow{t} S'}$
- **Atomic Transitions**: Rules which do have another transition in their premise
- **Context rules**: Rules to apply atomic transitions in presence of certain specific contexts.

Basis for JavaScript Isolation

1. All explicit property access has form x , $e.x$, or $e1[e2]$
2. The implicitly accessed property names are: $0, 1, 2, \dots$, `toString`, `toNumber`, `valueOf`, `length`, `prototype`, `constructor`, `message`, `arguments`, `Object`, `Array`, `RegExp`
3. Dynamic code generation (converting strings to programs) occurs only through `eval`, `Function`, and indirectly `constructor`
4. A pointer to the global object can only be obtained by: `this`, native method `valueOf` of `Object.prototype`, and native methods `concat`, `sort` and `reverse` of `Array.prototype`
5. Pointers to local scope objects through `with`, `try/catch`, “named” recursive functions `var f = function g(..){... g(..)...`

Sample Facebook vulnerability



- FBJS `e1[IDX(e2)]` did not correctly convert objects to strings
- Exploit: we built an FBJS application able to reach the DOM.
- Disclosure: we notified Facebook; they promptly patched FBJS.
- Potential for damage is considerable.
 - Steal cookies or authentication credentials
 - Impersonate user: deface or alter profile, query personal information, spam friends, spread virally.

The run time monitor IDX

- We need some auxiliary variables: we prefix them with $\$$ and include them in our blacklist B


```
var $String=String;
var $B={p1:true;...pn:true,eval:true,...,$:true,...}
```
- Rewrite `e1[e2]` to `e1[IDX(e2)]`, where


```
IDX(e) =
  ($=e,{toString:function(){
    return ($=$String($),
    $B[$]?"bad":$)
  }})
```

 - Blacklisting can be turned into whitelisting by inverting the check above `($B[$]?$: "bad")`.
- Our rewriting faithfully emulates the semantics


```
e1[e2] -> val1[e2] -> val1[va2] -> l[va2] -> l[m]
```

Improving our solutions by wrapping

- No need to blacklist `sort`, `concat`, `reverse`, `valueOf`.
 - We can wrap them as follows


```
$OPvalueOf=Object.prototype.valueOf;
Object.prototype.valueOf=
  function(){var $=$OPvalueOf.call(this);
    return ($==Global?null:$)}
```
 - This variant is provably correct.
- Wrapping `eval` and `Function`: possible in principle

Four “FBJS” Theorems

[CSF'09, ESORICS'09]



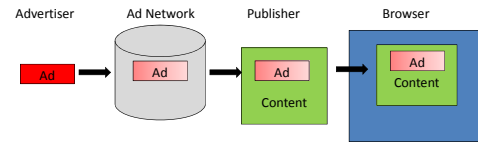
- **Theorem 1:** Subset $J(B)$ of ES-3 prevents access to chosen blacklist B (assuming $B \cap P_{\text{nat}} = \emptyset$)
- **Theorem 2:** Subset $J(B)_G$ of $J(B)$ prevents any expression from naming the global scope object
- **Theorem 3:** Subset $J(B)_S$ of $J(B)_G$ prevents any expression from naming any scope object
- **Theorem 4:** A specific “wrapping” technique preserves Theorem 3 and allows previously blacklisted functions to be safely used

We can prove isolation for a language very similar to FBJS. Success!! ?

Yahoo! AdSafe



- Goal: Restrict access to DOM, global object



- This is a *harder* problem than SNS applications
 - Advertising network must screen advertisements
 - Publishing site is not under control of ad network

Outline for Today, Tomorrow



- Background: computer security
- Web fundamentals and browser security
- JavaScript isolation
 - How can trusted and untrusted code be executed in the same environment, without compromising functionality or security?
- Three parts
 - Isolate untrusted application from hosting page
 - Isolate one untrusted application from another
 - Mediated access: reference monitor for critical resources
- Additional topics
 - Operational semantics (covered between parts I and II)
 - Foundations for Web security

Structured Operational Semantics



Operational Semantics



- Abstract definition of program execution
 - Sequence of actions, formulated as transitions of an abstract machine
- States corresponds to
 - Expression/statement being evaluated/executed
 - Abstract description of memory and other data structures involved in computation

Structural Operational Semantics



- Systematic definition of operational semantics
 - Specify the transitions in a syntax oriented manner using the inductive nature of program syntax
- Example
 - The state transition for $e1 + e2$ is described using the transitions for $e1$ and the transition for $e2$
- Plan
 - SOS of a simple subset of JavaScript
 - Summarize scope, prototype lookup in JavaScript

Simplified subset of JavaScript

- Three syntactic categories
 - Arith expressions : $a ::= n \mid x \mid a + a \mid a * a$
 - Bool expressions : $b ::= a <= a \mid \text{not } b \mid b \text{ and } b$
 - Statements : $s ::= \text{skip} \mid x := a \mid s; s \mid$
if b then s else $s \mid$ while b do s
- States
 - Pair $S = \langle t, \sigma \rangle$
 - t : syntax being evaluated/executed
 - σ : abstract description of memory, in this subset a function from variable names to values, i.e.,
 $\sigma : \text{Var} \rightarrow \text{Values}$

Form of SOS

General form of transition rule:

$$\frac{P_1, \dots, P_n}{\langle t, \sigma \rangle \rightarrow \langle t', \sigma' \rangle} \quad \frac{P_1, \dots, P_n}{\langle t, \sigma \rangle \rightarrow \sigma'} \quad (1)$$

P_1, \dots, P_n are the **conditions** that must hold for the transition to go through. Also called the **premise** for the rule. These could be

- Other transitions corresponding to the sub-terms.
- Predicates that must be true.
- Calls to meta functions like :
 - $\text{get}(\sigma, x) = v$: Fetch the value of x .
 - $\text{put}(\sigma, x, n) = \sigma'$: Update value of x to n and return new store.

Sample operational rules

A rule for Arithmetic Expressions

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle} [A_{3a}] \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle n + a_2, \sigma \rangle \rightarrow \langle n + a'_2, \sigma \rangle} [A_{3b}]$$

How to interpret this rule ?

- If the term a_1 partially evaluates to a'_1 then $a_1 + a_2$ partially evaluates to $a'_1 + a_2$.
- Once the expression a_1 reduces to a value n , then start evaluating a_2

Example :

$$\langle (10 + 12) + (13 + 20), \sigma \rangle \xrightarrow{A_{3a}} \langle 22 + (13 + 20), \sigma \rangle \xrightarrow{A_{3b}} \langle 22 + 33, \sigma \rangle$$

Sample rules

A rule for Statements

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma' \rangle}{\langle x := a, \sigma \rangle \rightarrow \langle x = a', \sigma' \rangle} [C_3] \quad \frac{\sigma' = \text{Put}(\sigma, x, n)}{\langle x := n, \sigma \rangle \rightarrow \langle \sigma' \rangle} [C_2]$$

How to interpret this rule ?

- If the arithmetic exp a partially evaluates to a' then the statement $x = a$ partially evaluates to $x = a'$.
- Rule C_2 applies when a reduces to a value n .
- $\text{Put}(\sigma, x, n)$ updates the value of x to n .

Example : $\langle (x := 10 + 12), \sigma \rangle \xrightarrow{C_3} \langle x := 22, \sigma \rangle \xrightarrow{C_2} \langle \sigma' \rangle$

Conditional and loops

If Then Else

$$\frac{\langle \text{if } tt \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_1, \sigma \rangle [C_{5a}] \quad \langle \text{if } ff \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_2, \sigma \rangle [C_{5b}]}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle \text{if } b' \text{ then } s_1 \text{ else } s_2, \sigma \rangle} [C_{5c}]$$

While

$$\langle \text{while } b \text{ do } s, \sigma \rangle \rightarrow \langle \text{if } b \text{ then } s; \text{ while } b \text{ s else skip end}, \sigma \rangle [C_6]$$

Context Sensitive Rules

Similar rules

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle} [A_{3a}] \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle n + a_2, \sigma \rangle \rightarrow \langle n + a'_2, \sigma \rangle} [A_{3b}]$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 * a_2, \sigma \rangle \rightarrow \langle a'_1 * a_2, \sigma \rangle} [A_{4a}] \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle n * a_2, \sigma \rangle \rightarrow \langle n * a'_2, \sigma \rangle} [A_{4b}]$$

- The above rules have a similar premise :
- Combine them into a **single** rule of the following form :

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{AC(a) \rightarrow AC(a')}$$

where $AC :: _ - + a \mid n + _ - * a \mid n * _ -$

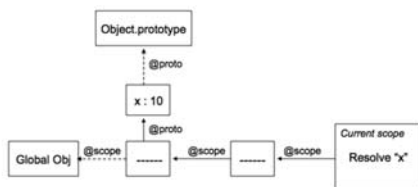
Moving to full JavaScript

- Program state is represented by a triple $\langle H, l, t \rangle$
 - H : program heap, mapping locations (L) to objects
 - l : Location of the current scope object ("activation record")
 - t : expression, statement, or program being evaluated
- Note
 - All definable values (including functions) are either objects or primitive values
 - Activation records are normal JavaScript objects and variable declarations define properties of these objects
 - Instead of a stack of activation records, there a chain of scope objects, called the scope chain

Heap operations

- Each Heap object o has the form
 - $\{p_1 : ov_1, \dots, p_n : ov_n\}$ where p_i are property names and ov_i are primitive values or heap addresses
- Operations on heap objects
 - $\text{Dot}(H, l, p) = l_1$
 - property p of object at location l
 - $\text{Put}(H, l, p, l_v) = H'$
 - Update property p of object at $H(l)$ producing H'
 - $H', l = \text{alloc}(H, o)$
 - Allocate object o at new location l

Scope and prototype lookup



- Every scope chain has the global object at its base
- Every prototype chain has `Object.prototype` at the top, which is a native object containing predefined functions such as `toString`, `hasOwnProperty`, etc

Some notation (based on ECMA Std)

- o *hasProperty* p
 - p is a property of object o or one of the ancestral prototypes of o
- o *hasOwnProperty* p
 - p is a property of object o itself
- A *JavaScript reference type*
 - pair written $l * p$ where l is a heap address, also called the base type of the reference, and p is a property name

Semantics of scope lookup

ECMA 2.62 :

- Get the next object (l) in the scope chain. If there isn't one, goto 4.
- If l "HasProperty" x , return a reference type $l * x$.
- Else, goto 1
- Return $\text{null} * x$.

$$\begin{aligned} \text{Scope}(H, l, "x") &= l_n \\ \langle H, l, x \rangle &\rightarrow \langle H, l, l_n * "x" \rangle \\ \text{HasProperty}(H, l, m) & \\ \text{Scope}(H, l, m) &= l \\ \neg(\text{HasProperty}(H, l, m)) & \\ H(l).@Scope &= l_n \\ \text{Scope}(H, l, m) &= \text{Scope}(H, l_n, m) \\ \text{Scope}(H, \text{null}, m) &= \text{null} \end{aligned}$$

Semantics of prototype lookup

ECMA 2.62 :

- If base type is null, throw a `ReferenceError` exception.
- Else, Call the `Get` method, passing `prop name(x)` and base type l as arguments.
- Return `result(2)`.

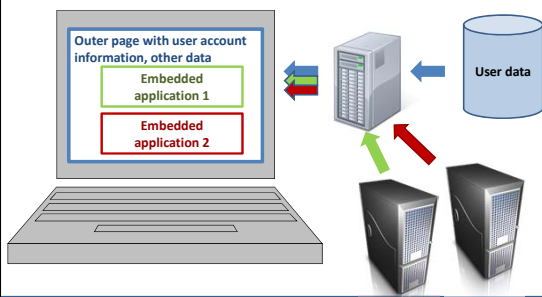
$$\begin{aligned} H_2.l_{\text{exp}} &= \text{alloc}(H, o) \\ o &= \text{newNativeErr}("", \# \text{RefErrProt}) \\ \langle H, l, (\text{null} * m) \rangle &\rightarrow \langle H_2, l, (l_{\text{exp}}) \rangle \\ \text{Get}(H, l, m) &= va \\ \langle H, l, l_n * m \rangle &\rightarrow \langle H, l, va \rangle \\ \text{HasOwnProperty}(H, l, m) & \\ \text{Dot}(H, l, m) &= va \\ \text{Get}(H, l, m) &= va \\ \neg(\text{HasOwnProperty}(H, l, m)) & \\ H(l).@prototype &= l_p \\ \text{Get}(H, l, m) &= \text{Get}(H, l_p, m) \end{aligned}$$

Summary of Operational Semantics

- Abstract definition program execution
 - Uses some characterization of program state that reflects the power and expressiveness of language
- JavaScript operational semantics
 - Based on ECMA Standard
 - Lengthy: 70 pages of rules (ascii)
 - Precise definition of program execution, in detail
 - Can prove properties of JavaScript programs
 - Progress: Evaluation only halts with expected set of values
 - Reachability: precise definition of “garbage” for JS programs
 - Basis for proofs of security mechanisms, variable renaming, ...

Isolation *Between* Untrusted Applications

Isolation between applications



FBJS limitations

- Authority leak
 - Can write/read properties of native objects
 - `var Obj = {};`
 - `var ObjProtoToString = Obj.toString;`
- Communication between untrusted apps
 - First application
 - `Obj.toString.channel = "message";`
 - Second application
 - `var receive_message = Obj.toString.channel;`

Defeat Sandbox

```
<a href="#" onclick="break()">Attack FBJS!</a> <script>
function break(){
  var f = function(){};
  f.bind.apply =
    (function(old){return function(x,y){
      var getWindow = y[1].setReplay;
      getWindow(0).alert("Hacked!");
      return old(x,y)}
    })(f.bind.apply)
}</script>
```

- Redefine bind method used to Curry functions
- Interferes with code that uses `f.bind.apply(e)`

How to isolate applications?

- Capability-based protection
 - Traditional idea in operating systems
 - Capability is “ticket” granting access
 - Process can only access through capabilities given
- If we had a capability-safe subset of JavaScript:
 - Give independent apps disjoint capabilities
- Problem: Is there a capability-safe JavaScript?

Foundations for object-capabilities

[S&P 2010]

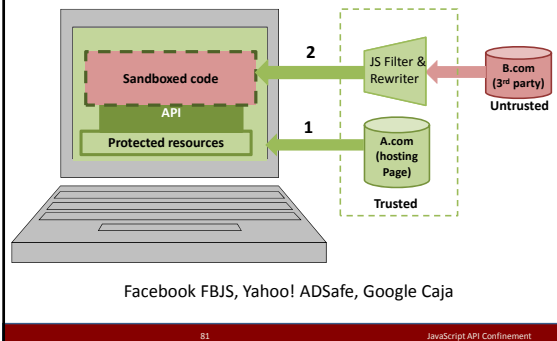
- Object-capability model [Miller, ...]
 - Intriguing, not formally rigorous
 - Examples: E (Java), JoeE (Java), Emily (Ocaml), W7 (Scheme)
- Authority safety
 - Safety conditions sufficient to prevent
 - Authority leak ("only connectivity begets connectivity")
 - Privilege escalation ("no authority amplification")
 - Preserved by program execution
 - Eliminates basis for our previous attacks
- Capability safety
 - Access control model sufficient to imply authority safety
- Theorems:** Cap safety \Rightarrow Auth safety \Rightarrow Isolation
 - Accepted examples satisfy our formal definitions



Mediated Access

How can trusted code provide access to security-critical resources?

Mediated Access



81

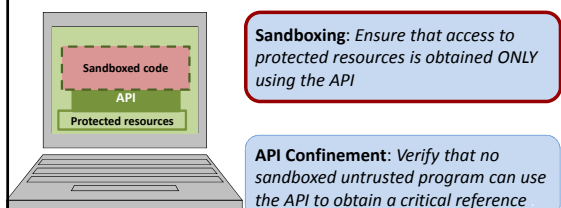
JavaScript API Confinement

ECMA Script 5 Strict Mode

- Restricted subset of JavaScript with "safer" semantics, e.g.
 - Assignment to an undeclared identifier does not create a property in the global object
 - Strict mode eval code cannot instantiate variables or functions in the variable environment of the caller to eval.
 - A *this* value of null or undefined is not converted to the global object ...
 - Strict mode code may not include a WithStatement
 - ...
- Goals
 - Provide language framework for code isolation
 - Protect trusted strict code against untrusted unstrict code

Language standard captures many properties explored in our research

Two Problems



83

JavaScript API Confinement

Evolution of JavaScript

- ES3: JS based on ECMA-262 3rd edition spec
- ES5: JS based on ECMA-262 5th edition spec (released in Dec 2009)
- ES5S: strict mode of the ES5 language
- SESlight: Secure ECMAScript, a subset of ES5S
 - Basis for our tool
 - SES is currently under proposal by the ECMA committee.

Stanford Security Workshop 2011

84

From ES3 to ES5S

| Restriction | Rationale |
|--|------------------------------|
| No delete on variable names | Achieving Static Scoping |
| No prototypes for scope objects | Achieving Static Scoping |
| No with | Achieving Static Scoping |
| No this coercion | Plugging Encapsulation Leaks |
| No .callee , .caller on argument objs | Plugging Encapsulation Leaks |
| No .callee , .arguments on function objs | Plugging Encapsulation Leaks |
| No arguments and formal parameters aliasing | Plugging Encapsulation Leaks |

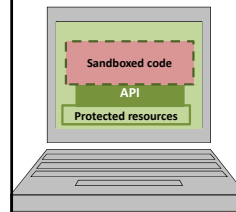
Property: ES5S is a lexically scoped language

9/7/2013

Stanford Security Workshop 2011

85

Second Problem



Sandboxing: Ensure that access to protected resources is obtained **ONLY** using the API

API Confinement: Verify that no sandboxed untrusted program can use the API to obtain a critical reference

86

JavaScript API Confinement

API Confinement Problem

PointsTo(x, P): Set of values that get stored in variable x during the execution of program P.

Confine(P, F): For all programs s in SES_{light} , If untrusted s has access to API defined by P, can s gain access to any forbidden reference in F?

9/7/2013

Techniques

- Points-to Analysis
 - Context Insensitive: Single Activation Record per function.
 - Flow Insensitive: Insensitive to the order of statements.
 - Only track references, ignore primitive values
- Expressed in Datalog (Whaley et al)
 - Collect Facts about the program: **Assign(x,y)**, **Load(x,y,z)**,...
 - Encode semantics of the program as Horn clauses:
 $Stack(x,l) :- Assign(x,y), Stack(y,l)$
 - Generate all possible consequence facts under the rules.
 - Tricky Case: eval, capturing implicit code execution.
- Scalable and well studied techniques for C, Java

88

Stanford Security Workshop 2011

9/7/2013

201

Simple Example

Program

```
var y = {};
var z = {};
var x = y;
x.f = z;
x = y.f;
```

encode

Facts

```
Stack(y, ly)
Stack(z, lz)
Assign(x, y)
Store(x, "f", z)
Load(x, y, "f")
```

9/7/2013

89

Datalog Predicates

| Predicates for encoding the term | | Predicates for encoding the heap-stack | |
|----------------------------------|-----------------|--|--------------|
| Assign(x, y) | Throw(l, x) | Heap(l, x, m) | Stack(x, l) |
| Load(x, y, z) | Catch(l, x) | Prototype(l, m) | FuncType(l) |
| Store(x, y, z) | TP(l, x) | ObjType(l) | ArrayType(l) |
| FormalArg(l, i, x) | FormalRet(l, x) | NotBuiltin(l) | |
| Actual(x, i, z, y, l) | Instance(l, x) | | |

- x, y, z are variable/property names
- l, m, n are object creation site labels (abstract locations)

9/7/2013

90

Inference Rules

Assignment, Load, Store

$Stack(x, l) \vdash Stack(y, l), Assign(x, y)$
 $Stack(x, n) \vdash Load(x, y, f), Stack(y, l), Prototype(l, m), Heap(m, f, n)$
 $Heap(l, f, m) \vdash Store(x, f, y), Stack(x, l), Stack(y, m), NotBuiltin(l)$
 $Prototype(l, n) \vdash Prototype(l, m), Prototype(m, n)$

Function Calls

$Assign(x, x) \vdash Actual(f, i, x, y, k), Stack(f, l), FormalArg(i, i, z)$
 $Assign(y, z) \vdash Actual(f, i, x, y, k), Stack(f, l), FormalRet(i, z)$

ToPrimitive Conversions

$Actual(n, 0, x, 'sd', l) \vdash TP(x, l), Stack(x, l), Prototype(l, m), Heap(m, 'toString', n), FuncType(n)$
 $Actual(n, 0, x, 'sd', l) \vdash TP(x, l), Stack(x, l), Prototype(l, m), Heap(m, 'valueOf', n), FuncType(n)$

9/7/2013

91

Procedure

Procedure D(P, F):

1. Build P' from P by adding "test" variable
2. $D_1 \leftarrow \text{Encode}(P') + \text{Encoding of built-in objects.}$
3. $D_2 \leftarrow \text{All possible consequences of } D_1$
4. Return *confined* iff there is *no* $l \in F$ such that $Stack("test", l) \in D_2$

Theorem: Procedure $D(P, F)$ is *sound*. If the procedure says *confined* then the API is actually safe.

9/7/2013

92

Yahoo! ADSafe

- ADSafe API does not use setters/getters
- can be *de-sugared to SES_{light}*
- **Annotations:**
 - $\$Num$: Added to patterns of the form $for(...i...)\{ o[i, \$Num] \}$
 - $\$Reject$: Added to patterns of the form $if(!reject(name))\{ ...object[name, \$Reject]... \}$
- Approx 2000 LOC.

Analysis tool output: NOT CONFINED if the methods `ADSAFE.lib` and `ADSAFE.go` are exposed to untrusted code.

9/7/2013

93

Feasible Attack !

```
<div id="test">
<script>
"use_strict";
```

- `ADSAFE.lib` defined as
`function (name, f){adsafe_lib[name] = f(adsafe_lib)}`
- ADSafe API protects DOM objects by hiding them behind `"__nodes__"` property of certain fake object (**Bunch objects**).
- **Assumption:** Sandboxed code can never write to `"__nodes__"` property of any object

```
</script>
</div>
```

`lib.v0;` as Bunch object to DOM wrapper

9/7/2013

Stanford Security Workshop 2011

94

Fixing the vulnerability

- Replace `ADSafe.lib` with the following
- ```
ADSafe.lib = function(name, f){
 if(reject_name(name)){
 adsafe_lib[name] = f(adsafe_lib)
 }
}
```

**Tool analysis result:** API is CONFINED.

*ADSafe API is confined under the provided annotations and the SES<sub>light</sub> threat model*

Repair adopted by Yahoo! ADSafe.

9/7/2013

95

## General Foundations for Web Security

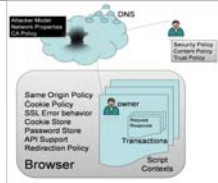


## Broader Foundations for Web Security

- Problem: Web platform and application security are not based on precise model

- Solution: Foundational model of web macro-platform supporting rigorous analysis

- Apply formal modeling techniques and tools, e.g., network security  $\Rightarrow$  web
- Precise threat models: web attacker, active network attacker, gadget attacker
- Support trustworthy design of browser, server, protocol, web application mechanisms



### Initial case studies

- Origin header
- Cross-Origin Resource Sharing
- Referer Validation,
- HTML5 forms
- WebAuth

- Find attacks, verify repairs

## A formal model of the web

- Includes browser, servers, and network
- Threat models include:
  - web attacker with no special network privilege
  - network attacker that can eavesdrop and/or modify unencrypted traffic at will
- Security goals include:
  - Security invariants
    - Assumptions about how today's Web works
    - Example: no DELETE in cross-origin HTTP requests
  - Session integrity
    - Attacker does not participate in the HTTP transaction

## Alloy

[Jackson]

- Concepts expressed by relations
  - General logical language
- User inputs “scope” to determine precision
  - Tool converts logical expressions to finitary form by allocating specific number of bits to each possible value
- SAT-based analysis
  - Satisfiability checker used to see if formula is satisfiable
  - Leverage large community working on efficient SAT checkers

## Web modeling examples

```

abstract sig NetworkEvent extends Event {
 from: NetworkEndpoint,
 to: NetworkEndpoint }
abstract sig HTTPEvent extends NetworkEvent {
 host: Origin }
sig HTTPRequest extends HTTPEvent {
 method: Method,
 path: Path,
 headers: set HTTPRequestHeader }
sig HTTPResponse extends HTTPEvent {
 statusCode: Status,
 headers: set HTTPResponseHeader
}

```

## Principals

- A Principal is an entity that controls a set of NetworkEndpoints and owns a set of DNSLabels, which represent fully qualified host names:

```

abstract sig Principal {
 servers: set NetworkEndpoint,
 dnslabels: set DNS
}
abstract sig PassivePrincipal
 extends Principal {} {
 servers in HTTPConformist
}

```

## Browsers

- A Browser is an HTTPClient together with a set of trusted CertificateAuthorities and a set of ScriptContexts. (For technical convenience, we store the Browser as a property of a ScriptContext.)

```

abstract sig Browser
 extends HTTPClient {
 trustedCA: set CertificateAuthority
}
sig ScriptContext {
 owner: Origin,
 location: Browser,
 transactions: set HTTPTransaction
}

```



## Cookies (modeled as a fact)

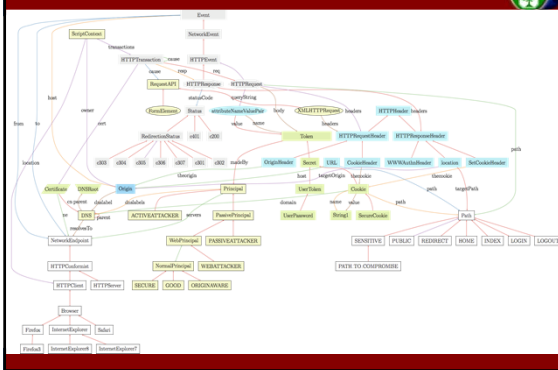
- Idea: HTTPRequests from Browsers contain only appropriate cookies from previous SetCookieHeaders
- ```
fact {
  all aReq:HTTPRequest | {
    aReq.from in Browser
    hasCookie[aReq]
  } implies all acookie: reqCookies[aReq] |
  some aresp: getBrowserTrans[aReq].resp | {
    aresp.host.dnslabel = aReq.host.dnslabel
    acookie in respCookies[aresp]
    happensBeforeOrdering[aresp,aReq]
  }
}
```

Property: session integrity

```
fun involvedServers[t:HTTPTransaction] :
  set NetworkEndpoint {
    (t.*cause & HTTPTransaction).resp.from
    + getTransactionOwner[t].servers
  }

pred webAttackerInCausalChain[t:HTTPTransaction] {
  some (WEBATTACKER.servers & involvedServers[t])
}
```

Alloy metamodel



Sample case studies

- HTML5 Forms
- Referer validation
- WebAuth protocol

Referer Validation

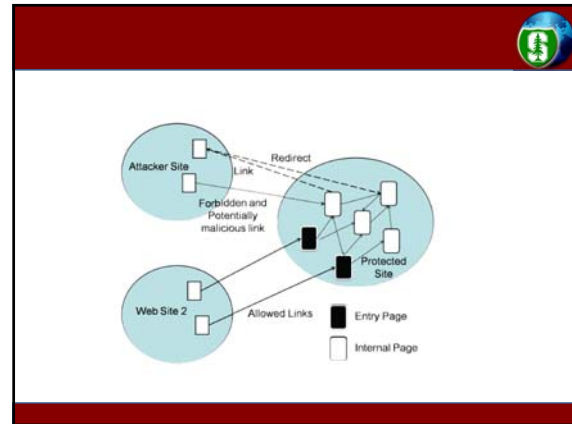
- A proposed defense against Cross-Site Request Forgery (CSRF) and Cross-Site Scripting (XSS) [F. Kerschbaum, 2007]
- Websites reject a request unless
 - the referer header is from the same site, or
 - the request is directed at an "entry" page vetted for CSRF and XSS vulnerabilities

Modeling

- Referer header already part of model
- Add check: RefererProtected principals allow HTTP requests with external Referers only on the "LOGIN" page:


```
fact {
    all aReq:HTTPRequest | {
      (getTransaction[aReq].resp.from
      in RefererProtected.servers )
      and isCrossOrigin[aReq]
    } implies aReq.path = LOGIN
  }
```


Referer header and redirects



Referer Validation - Countermeasure

- **Exploitation**
 - CSRF and XSS can be carried out on websites protected with Referer Validation
- **Countermeasure**
 - This vulnerability is difficult to correct as Referer header has been widely deployed
 - Websites can try to suppress all outgoing Referer headers using, for example, the `noreferrer` relation attribute on hyperlinks.
- **Web extension**
 - Origin header records path of redirects

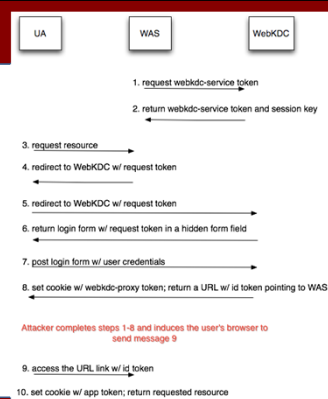
WebAuth

- Web-based Single Sign-On protocol
- WebAuth and a similar protocol, Central Authentication Service (CAS), are deployed at over 80 universities worldwide
- Although we analyze WebAuth specifically, we have verified the same vulnerability exists in CAS

WebAuth Protocol



WebAuth Attack



WebAuth Analysis

- Attack
 - An insider can share privileged web resources with unprivileged users without sharing login credentials
 - Attacker can steal sensitive user information by logging users into attacker's account
- Countermeasure
 - Store a nonce in a host cookie to bind messages 3 and 9, and splice in messages in between by including the nonce in the request and id tokens.
 - Verified the fix up to a finite size in our model

Statistics for the case studies

| Case Study | Lines of new code | No. of CNF clauses | CNF gen. time (sec) | CNF solve time (sec) |
|--------------------|-------------------|--------------------|---------------------|----------------------|
| HTML5 Form | 20 | 976,174 | 27.67 | 73.54 |
| Referer Validation | 35 | 974,924 | 30.75 | 9.06 |
| WebAuth | 214 | 355,093 | 602.4 | 35.44 |

- The base model contains some 2,000 lines of code
- Tests were performed on an Intel Core 2 Duo 3.16GHz CPU with 3.2 GB memory

Conclusion of case studies

- Identified previously unknown attacks in HTML5 Forms, Referer validation, and WebAuth
- Proposed countermeasures to the attacks
- These attacks are identified based on a formal model the Web that is implemented in Alloy
- Modeling approach can discover practical new attacks and verify the security of alternate designs, up to a certain size of the model

Challenge

- Does CSP prevent XSS?

Goals and Challenges Ahead

- Language-based isolation
 - Better understanding of object-capability model
 - Apply to JavaScript and other languages: E, Joe-E, Emily, W7, ES 3 \Rightarrow ES 5
 - Better tools for working with secure JavaScript
 - Wider recognition and deployment through standards, browser implementations
- Web platform security
 - Formalize additional properties of web platform
 - Browser same-origin
 - Cookie policies
 - Headers, ...
 - Prove correctness of accepted defenses
 - Improve design of central components
 - Improve design of new features (e.g., native client)

Conclusions

- The web is an exciting area for Computer Science
- Isolating untrusted JavaScript
 - Isolate untrusted application from hosting page
 - Isolate one untrusted application from another
 - Confinement: mediate access to critical resources
- Many more Web security problems
 - Define precise model of web application platform
 - Analyze protocols, conventions, attacks, defenses
 - Range of information flow problems
 - Look at correctness, precision of web application development tools

References



- With A. Taly, S. Maffei:
 - Operational semantics of ECMA 262-3 [APLAS'08]
 - Language-Based Isolation of Untrusted JavaScript [CSF'09]
 - Run-Time Enforcement of Secure JavaScript Subsets [W2SP'09]
 - Isolating JavaScript with Filters, Rewriting, and Wrappers [ESORICS'09]
 - Object Capabilities and Isolation of Untrusted Web Applications [S&P'10]
 - Automated Analysis of Security-Critical JavaScript APIs [S&P'11] (with T. + Google group)

Additional related work



[Yu,Chander,Islam,Serikov'07] *JavaScript instrumentation for browser security*.
Rewriting of JavaScript to enforce security policies based on edit-automata.

[Sands,Phung,Chudnov'09] *Lightweight, self protecting JavaScript*.
Aspect-oriented wrapping of DOM to enforce user-defined safety policies.

[Jensen,Møller,Thiemann'09] *Type analysis for JavaScript*.
Abstract-interpretation based analysis to detect basic type errors.

[Chugh,Meister,Jhala,Lerner'09] *Staged information flow for JavaScript*.
Static information flow analysis plus run-time checks for integrity and confidentiality.

[Livshits,Guarnieri'09] *GateKeeper: Mostly static enforcement of security and reliability policies for JavaScript code*.
Enforcing policies by filtering and rewriting based on call-graph and points-to analysis.

Web Sandbox (Scott Isaacs). Based on BrowserShield.
Rewriting and run-time monitoring with performance penalty.

