RAVAGE

***R**untime **A**nalysis of **V**ulnerabilities*
***A**nd **G**eneration of **E**xploits*

Xiaoran Wang
Yoel Gluck

black hat
USA 2014

AGENDA

- Background
- RAVAGE
- Demo
- Integrations
- Takeaways

BACKGROUND

EXISTING TOOLS

- Static Analysis:
  - AppScan Source (Ounce-IBM), Fortify SCA, Coverity Security Advisor, Checkmarx CxSuite, Veracode SAST, FindBugs, Brakeman
- Blackbox:
  - AppScan, Burp, WebInspect, Hailstorm, ZAP

SCA
CLAIMS

- Scalable

“Scales well -- can be run on lots of software, and can be run repeatedly (as with nightly builds)”

OWASP - Source Code Analysis Tools

SCA
CLAIMS

- Scalable
- Accurate

“tools can automatically find with high confidence […] buffer overflows, SQL Injection Flaws, and so forth”

OWASP - Source Code Analysis Tools

SCA

CLAIMS

- Scalable
- Accurate
- Easy to use

“Output is good for developers -- highlights the precise source files and line numbers that are affected”

OWASP - Source Code Analysis Tools

SCA

REALITY

- Not Scalable
  - Very slow for complex code bases
  - Trims dataflows to make it manageable
- Not Accurate
  - Does not take into account runtime information
  - Prone to issues with interface/implementation/reflection
- Not Easy to use
  - Most UIs are not easy to use
  - For complex dataflows an example URL would be simpler

BLACKBOX

REALITY

- No knowledge of the app:
  - Struggles with wizard-like pages
  - Misses unlinked pages
- Relies on response data/metadata:
  - Misses vulns when nothing changes in output
  - Causes FPs when some changes are detected
- No code coverage
- No dataflows
- Potentially disruptive

"These tools can also be seductive, since they do find lots of potential issues. While running the tools doesn't take much time, each one of the potential problems takes time to investigate and verify."

OWASP Testing Guide v3

RUNTIME ANALYSIS (RTA)

WHAT IS IT?

- Monitors the program at runtime
- Detects when data flows from untrusted sources to sinks
- Can detect:
  - XSS
  - SQLi
  - Static encryption keys
  - Sensitive data leaking via logs
  - App misconfigurations

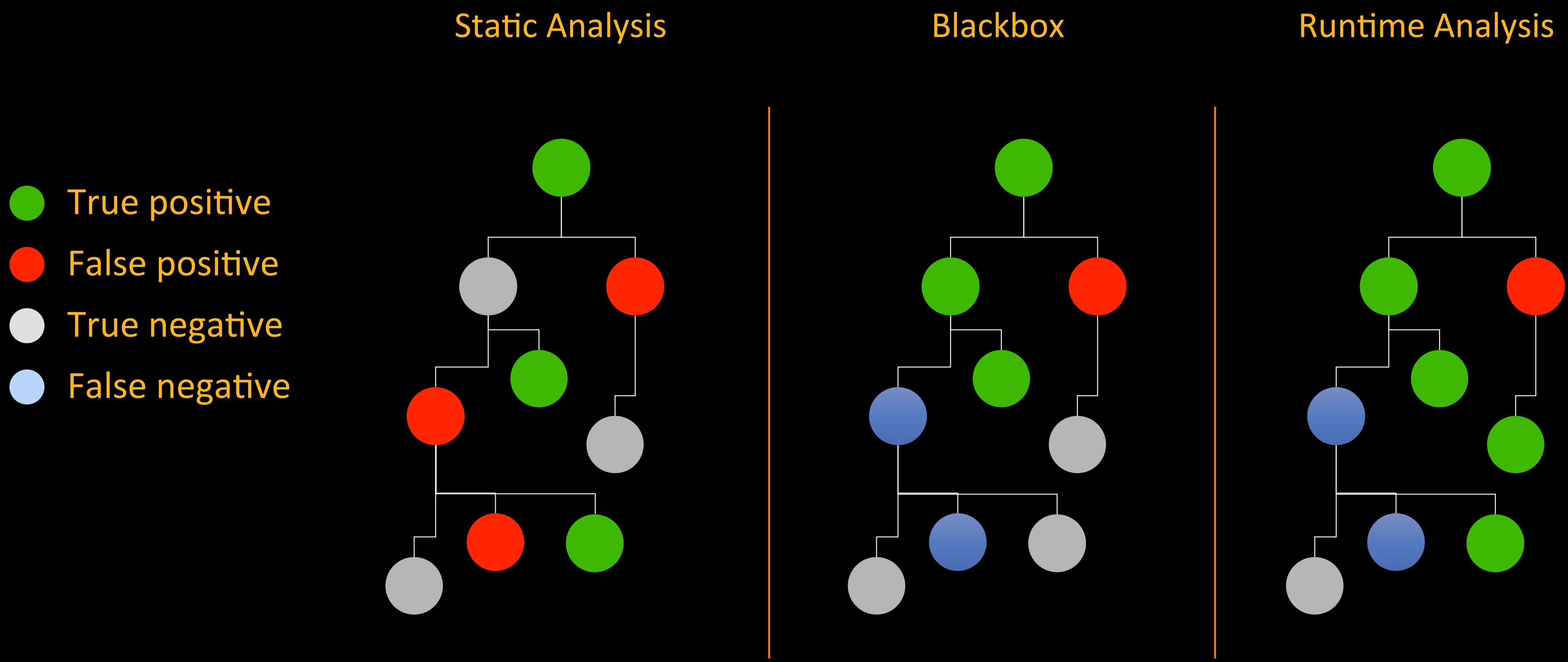
RTA
ADVANTAGES

- Low false positive rate
- No exploit data needed
  - Detects vulnerabilities during standard usage
- No source code needed
  - Though useful when reviewing the results
- Can leverage existing testing
- Complete dataflow

TOOLS
COMPARISON

- Existing runtime analysis tools
  - No dataflow
  - Incorrect dataflow
  - Limited dataflow only on String objects

TOOLS
COMPARISON



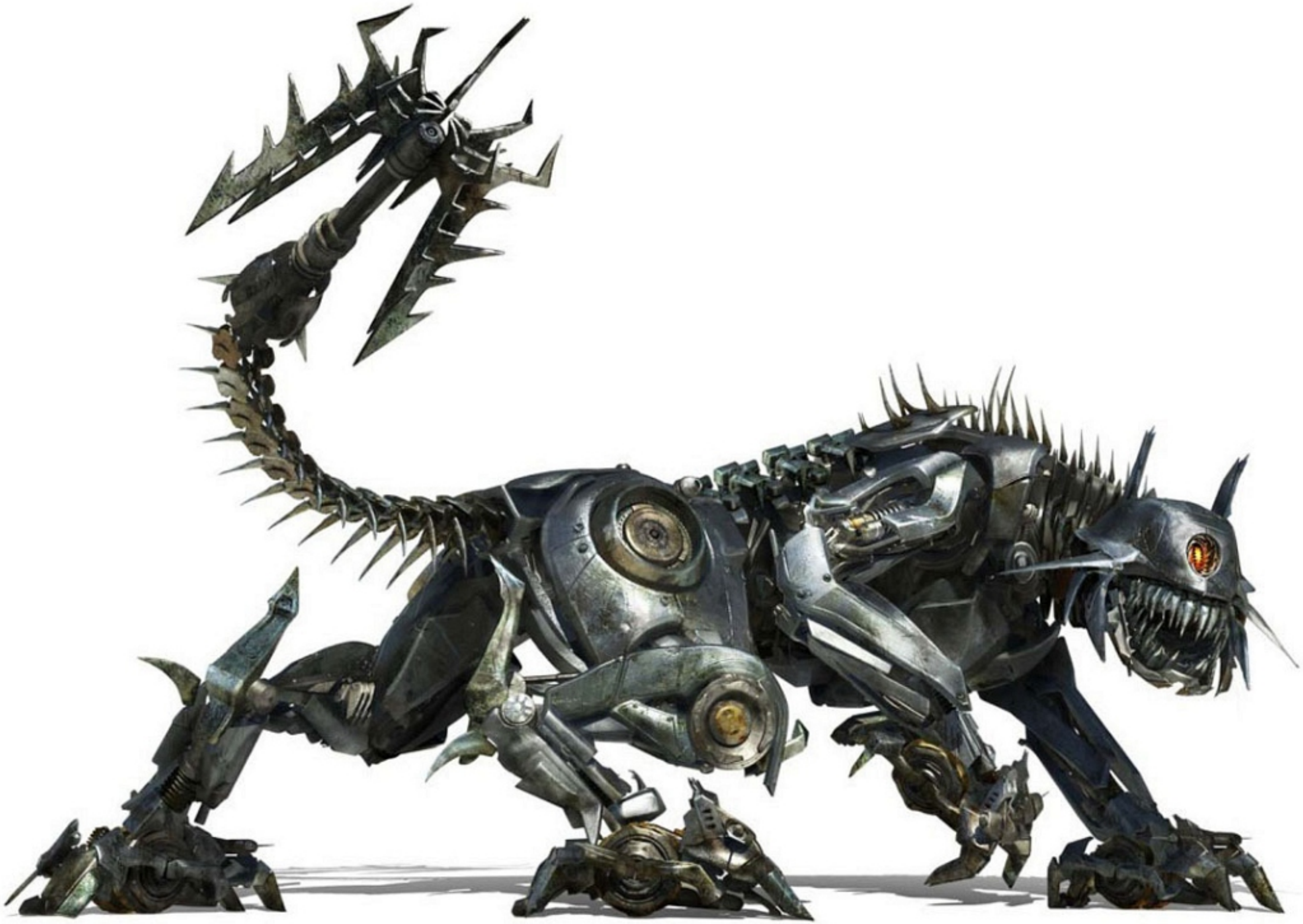
TOOLS

STATISTICS

- We experimented with 3 open source web applications
- Limit the scope of vulnerabilities to be
  - XSS(stored, reflected)
  - Arbitrary Redirection
  - Injections (SQL, XML, CMD)

	Reported Issues #	Real Vulns #	False Positive	False Negative
SCA	150	14	90%	8%
BlackBox	32	4	87%	70%
RAVAGE	13	11	15%	27%

RAVAGE



RAVAGE

Runtime Analysis of Vulnerabilities And Generation of Exploits

- RAVAGE is our implementation of RTA for Java
- Open source
- ~5k LOC
- Written in Java/C++/Assembly

DESIGN

IN THEORY

- Hook relevant operations
  - function invocations e.g. String s =foo(a);
  - field accesses e.g. String s = obj.value;
  - assignments e.g. String s = a;
  - array accesses e.g. String s = args[1];
- Track operations as data points

DESIGN

IN THEORY

- Store each data point as a *Node*
  - File name
  - Line number
  - Context
  - Flags/Info
- Store *Node* in original object
- Split objects at each data point
- Chain Nodes to create a complete dataflow graph

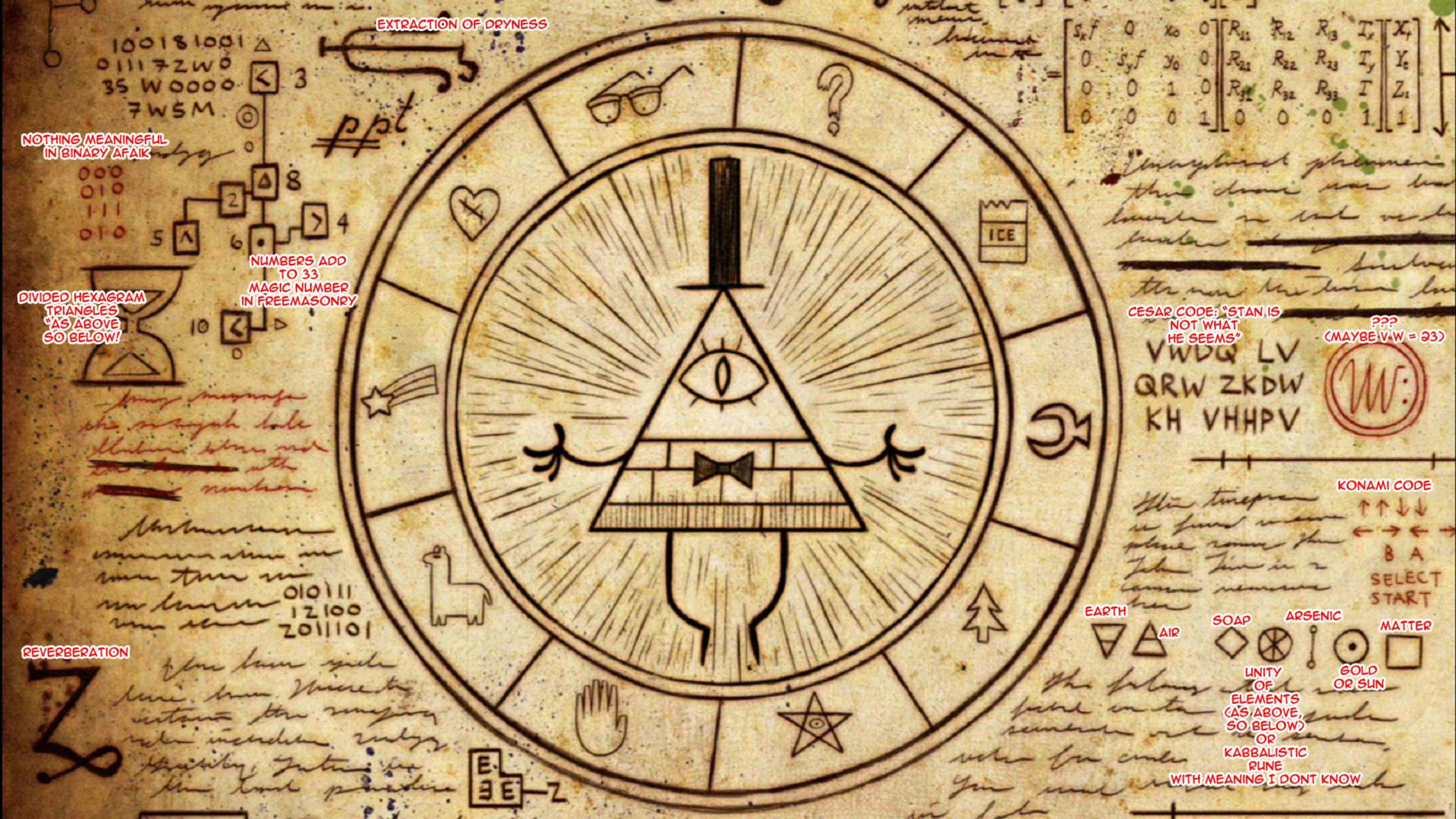
DESIGN

OBJECTIVE

$$R : V \times L \longrightarrow \{\text{true, false}\}$$

$$v_1 \ R \ l_1 \land p \vdash v_1 \rightsquigarrow v_2 \land p \vdash l_1 \triangleright l_2 \Rightarrow v_2 \ R \ l_2$$

$$v \ R \ l_1 \land l_1 \sqsubseteq l_2 \Rightarrow v \ R \ l_2$$



EXTRACTION OF DRYNESS
100181001
01117ZW0
35 W0000
7W5M
NOTHING MEANINGFUL IN BINARY AFAIK
000
010
111
010
NUMBERS ADD TO 33 MAGIC NUMBER IN FREEMASONRY
DIVIDED HEXAGRAM TRIANGLES "AS ABOVE SO BELOW!
ppt
ICE
CESAR CODE: "STAN IS NOT WHAT HE SEEMS"
VWDQ LV
QRW ZKDW
KH VHHPV
???
(MAYBE V W = 23)
KONAMI CODE
B A
SELECT
START
EARTH
AIR
SOAP
ARSENIC
MATTER
UNITY OF ELEMENTS (AS ABOVE, SO BELOW) OR KABBALISTIC RUNE WITH MEANING I DONT KNOW
GOLD OR SUN
010111
12100
2011101
REVERBERATION

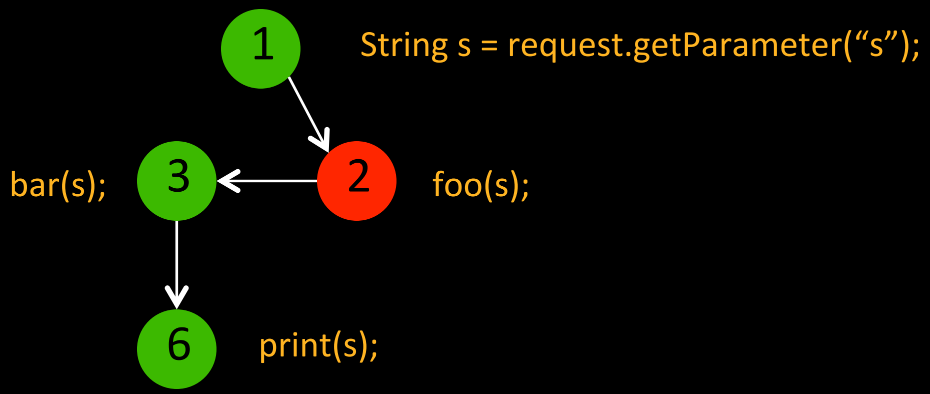
“… a.k.a. track all relevant nodes, but no extra ones”

RAVAGE BH USA 2014

DESIGN

EXAMPLE #1

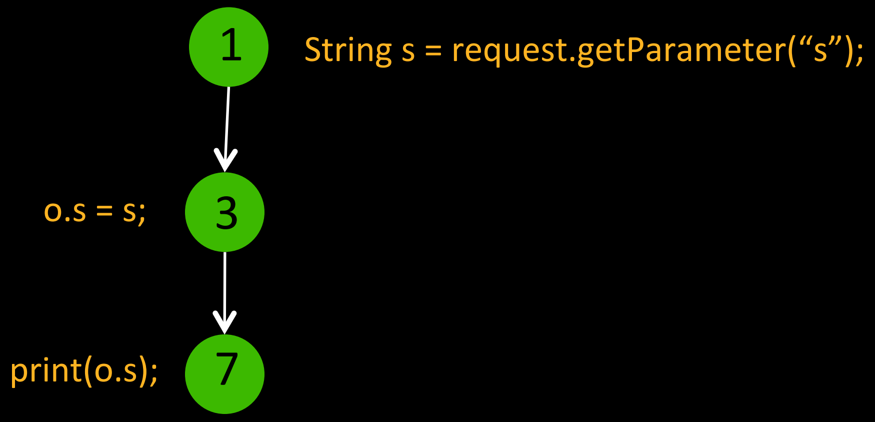
```
1: String s = request.getParameter("s");

2: foo(s);

3: bar(s);


4: void foo(String s) {print("Welcome");}

5: void bar(String s) {

6:   print(s);

7: }
```



DESIGN

EXAMPLE #2

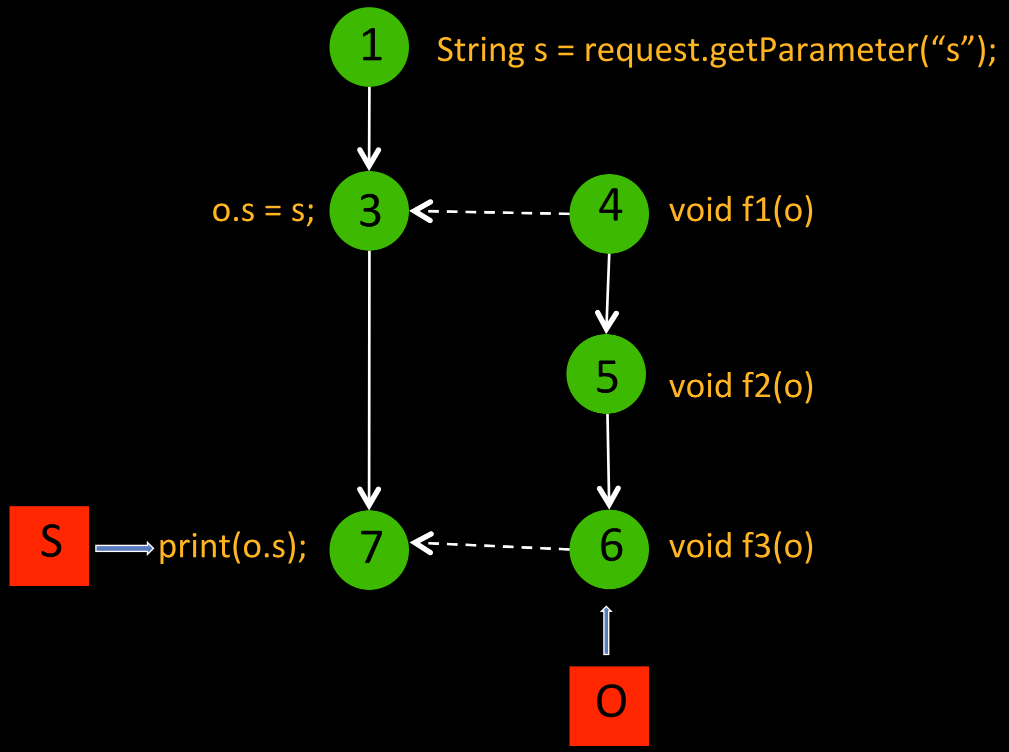
```
1: String s = request.getParameter("s");
2: Object o = new Object();
3: o.s = s;
4: void f1(o)
5:  void f2(o)
6:    void f3(o)
7:        print(o.s);
```



DESIGN

EXAMPLE #2

```
1: String s = request.getParameter("s");

2: Object o = new Object();

3: o.s = s;

4: void f1(o)

5:  void f2(o)

6:    void f3(o)

7:      print(o.s);
```



DESIGN

CAN IT BE DONE?

- Storing stack traces is easy
  - Complete and correct dataflows are not
- Tracking some events is trivial
  - Assignments and control flow are not
    - Performance impact on CPU?
- Splitting Immutable objects (e.g. String) is easy
  - Other objects are not
    - Singleton, Mutable, Resource-bound objects
  - Performance impact on memory?
- Invisible to the program

INTRODUCING …

RAPPER

WRAPPER

DESIGN

STILL IN THEORY

- Wrap each object with a “Wrapper” object
- Record Node info in Wrapper (e.g. file, line, taint)
- Split the Wrapper instead of the object

DEMO

Attempt #1

IMPLEMENTATION

BYTECODE - DESIGN

- Inject RAVAGE hooks into java class files
- Create rules to detect sources/sinks
- At runtime in hooks:
  - On objection creation, create wrapper/nodes
  - Tracks dataflows by chaining nodes
  - Detect vulnerabilities according to rules

IMPLEMENTATION

BYTECODE - FAIL

- Cannot change types of synthetic fields
  - The this$0 object for inner classes
- Cannot change some method signatures
  - Checked by reflection
- Annotations do not always work
- Dynamically defined classes can't be instrumented

Attempt #2

IMPLEMENTATION

JVM - BACKGROUND

- The JVM interprets the bytecode in class files
- OpenJDK HotSpot JVM has two interpreters
  - C++ Interpreter
    - Old, obsolete
    - big switch table
  - Template Interpreter
    - Written in assembly code
    - Dynamically generated at runtime
- We modified the Template Interpreter in JDK 8

IMPLEMENTATION

JVM - DESIGN

- Modify JVM interpreter to add RAVAGE hooks
- Create rules to detect sources/sinks
- At runtime in hooks:
  - On object creation, create wrapper/nodes
  - Wrappers invisible to rest of system
    - e.g. unwrap before comparing two string objects
  - Tracks dataflows by chaining nodes
  - Detect vulnerabilities according to rules

IMPLEMENTATION

JVM - CHALLENGES

- HotSpot JVM is complicated
  - 250,000 LOC
  - Many optimizations and tweaks to:
    - GC, Cache, context switches, stub codes
  - Painful to debug assembly code
    - No source code mapping, no context
    - Using GDB to debug a Java program is challenging


Optimator
SPATEN
MUNICH
Third Coast Beer
MILK STOUT
Reserve Ale
BOULEVARD
PRAGUE

IMPLEMENTATION

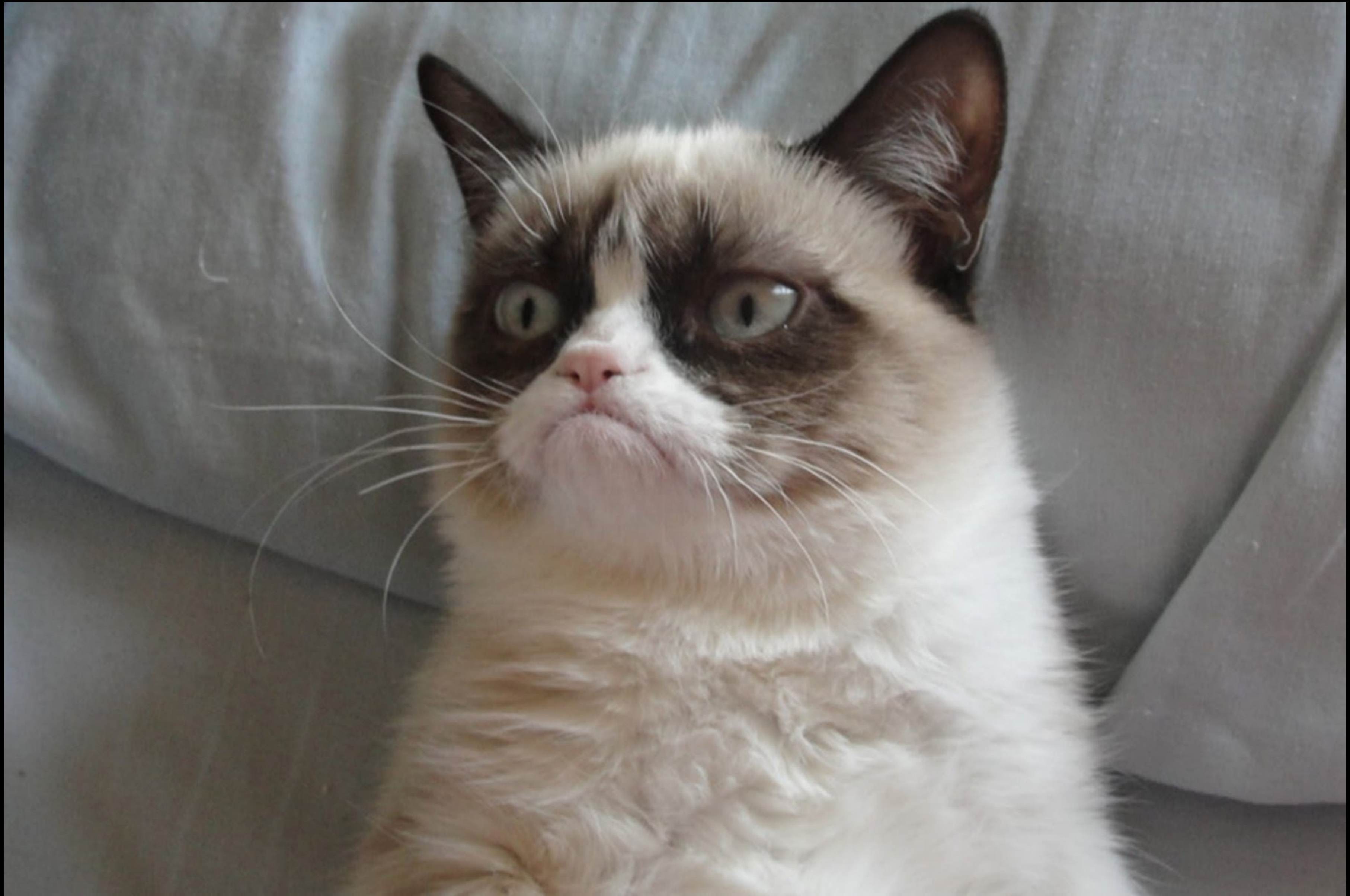
PERFORMANCE

- **At what layer to implement RAVAGE?**

black hat USA 2014

IMPLEMENTATION

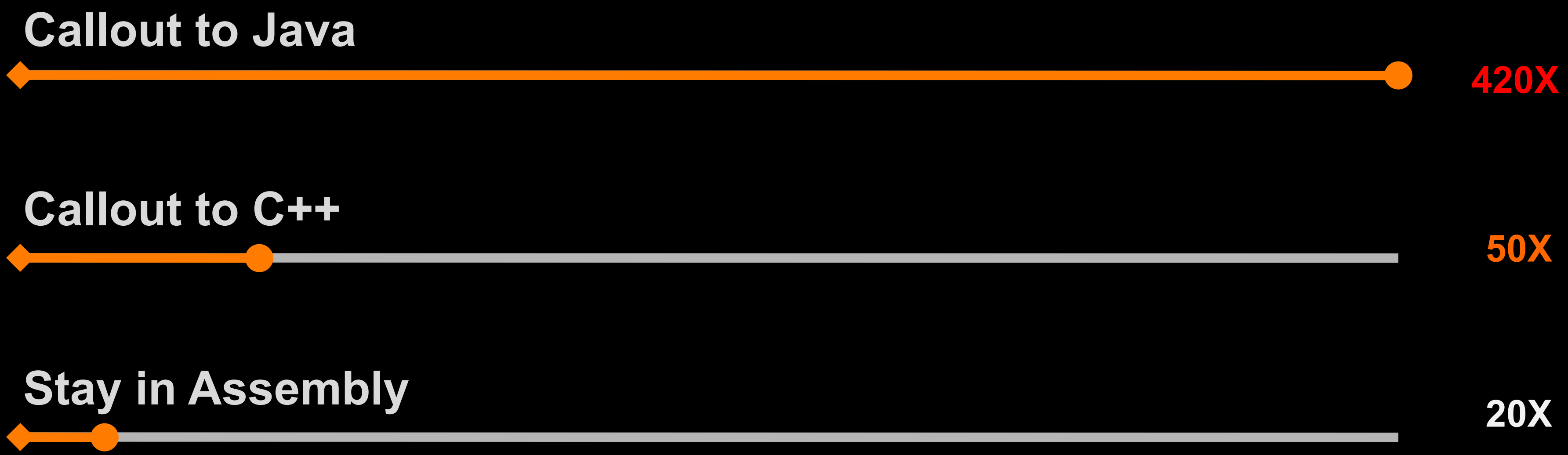
PERFORMANCE



IMPLEMENTATION

PERFORMANCE

- At what layer to implement RAVAGE?



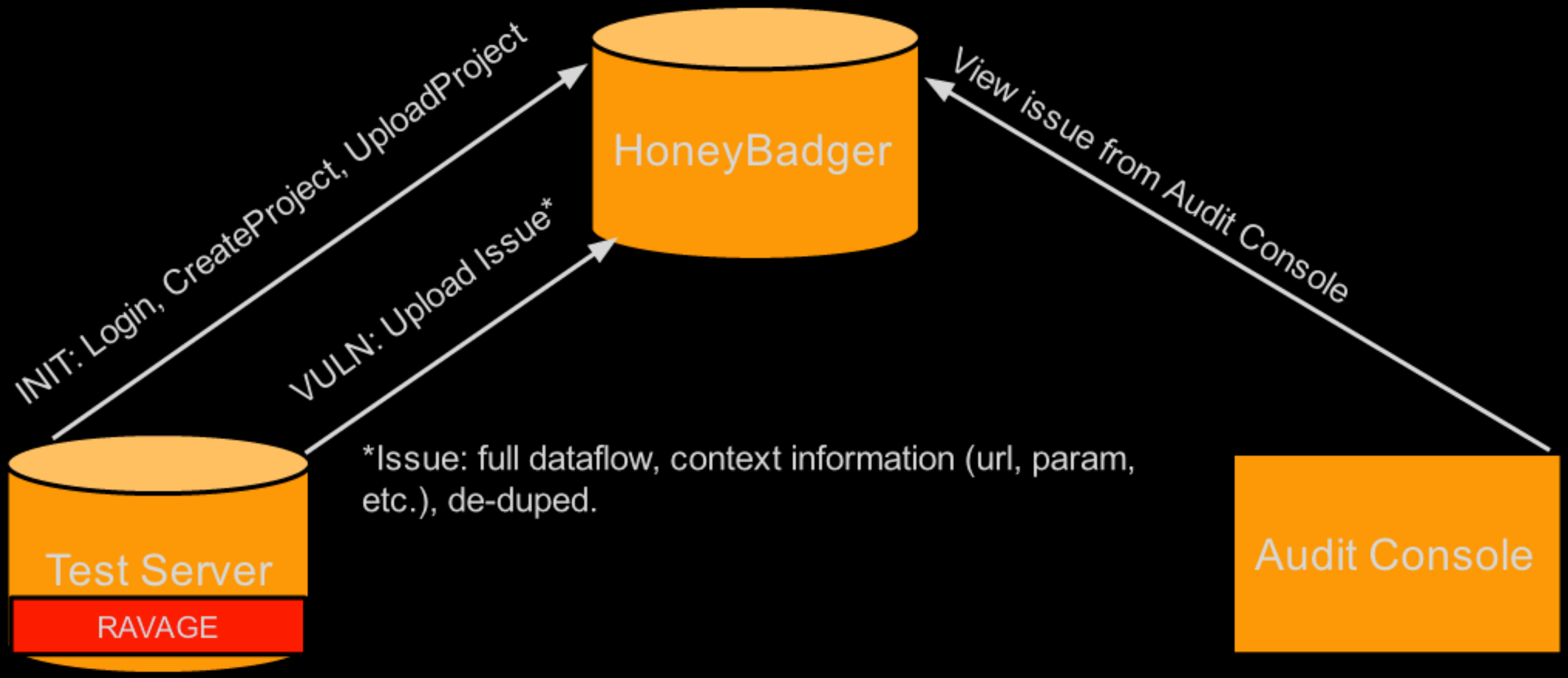
IMPLEMENTATION

PERFORMANCE

- Still significant slow down
  - Target 10X slowdown after JIT modifications

IMPLEMENTATION

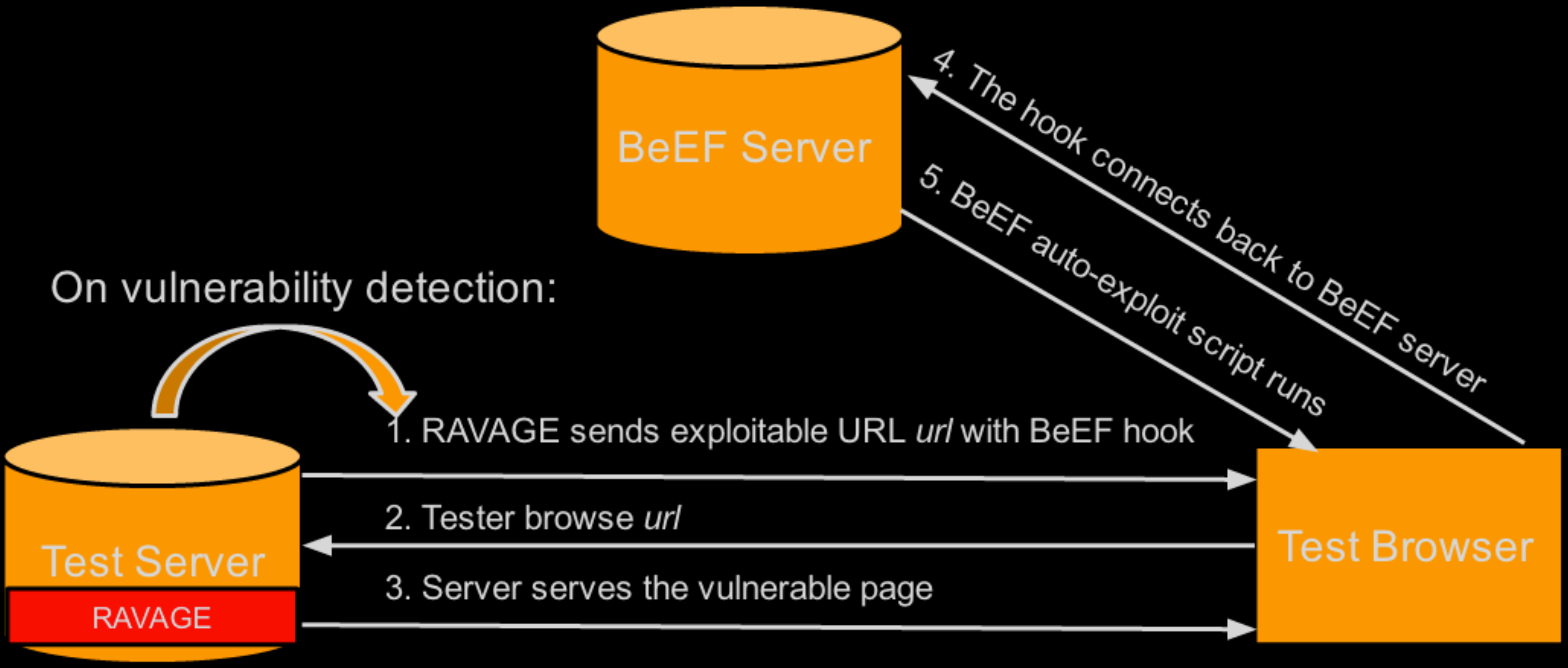
PLUGINS - HoneyBadger



*Issue: full dataflow, context information (url, param, etc.), de-duped.

IMPLEMENTATION

PLUGINS – BeEF



DEMO

INTEGRATIONS

INTEGRATIONS

TOOLS / ENVIRONMENTS

- Static analysis
  - Run RAVAGE and SCA, then cross verify the results
  - Run SCA, verify the results with RAVAGE
- Blackbox testing
  - Gather dataflows of vulnerabilities detected by Blackbox

INTEGRATIONS

TOOLS / ENVIRONMENTS

- Testing framework
  - Use Unit testing, Selenium testing, and other automation frameworks to drive RAVAGE
- Testing environment
  - Leverage RAVAGE to detect vulnerabilities
- Production
  - Block attacks in real-time

TAKEAWAYS

TAKEAWAYS

AND FUTURE WORK

- RAVAGE is a Java Runtime Analysis tool to automatically
  - Detect vulnerabilities
  - Report vulnerabilities
  - Generate exploits for vulnerabilities
- We need your help improving its performance
- Contribute back to Java as a developer option
- We need your help building a rich rule set
- We encourage you to implement RAVAGE for other languages

YOEL GLUCK / salesforce.com

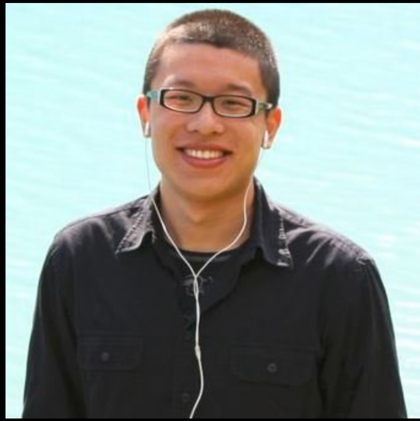
@GluckYoel

yoel.gluck2@gmail.com

www.ravagesecurity.com

black hat USA 2014

XIAORAN WANG / salesforce.com



@0x1a0ran

xiaoran@x1a0ran.com

THANK YOU!

References

- http://www.splendidwallpaper.com/wp-content/uploads/2009/08/transformers_2_ravage_1600x900.jpg
- http://www.wallpapermania.eu/images/data/2013-08/5673_Beautiful-big-fluffy-animal-sad-cat.jpg
- http://blog.drunkandfull.com/wp-content/uploads/2009/12/beer.jpg
- http://www.theangryhoneybadger.com/