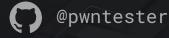
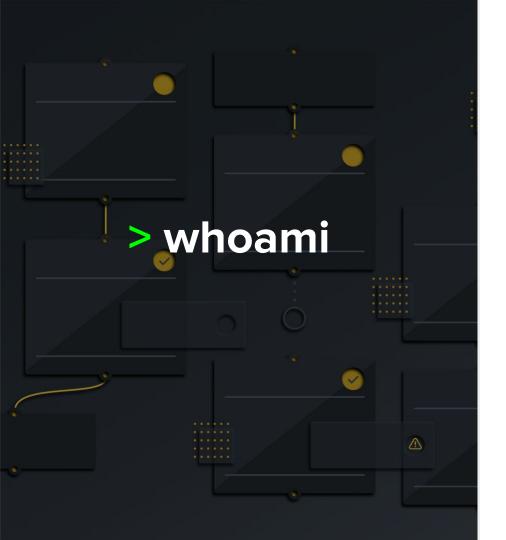




CodeQL as an audit oracle palantir









Alvaro Muñoz @pwntester

Staff Security Researcher with GitHub

Presented at BlackHat, DEFCON, OWASP AppSec, ...

Mostly focused on Java research and web technologies

100+ RCE CVEs :)



GitHub Security Lab

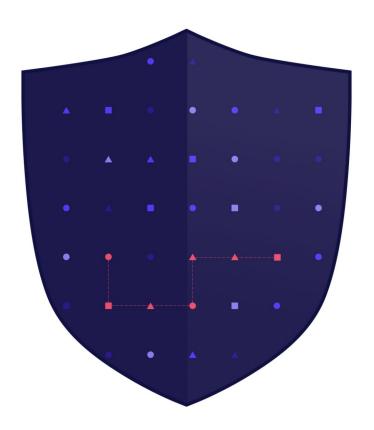
https://securitylab.github.com

GitHub Security Lab

Securing the world's software, together

GitHub Security Lab's mission is to inspire and enable the community to secure the open source software we all depend on.

Follow @GHSecurityLab

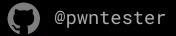




Today we will learn

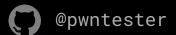
- What CodeQL is
- How to write queries in CodeQL to identify patterns in code
- How to use CodeQL to assist your code review efforts

https://github.com/github/codeql-dubbo-workshop



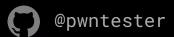
Agenda

- 1. Intro to CodeQL
- 2. Case Study: Apache Dubbo
 - 3. Hands-on exercises



Disclaimer

- I am a CodeQL user but do not work on the CodeQL implementation itself
- My aim is to provide you with a practical and working understanding of how to use CodeQL for code auditing by giving you the tips, tricks, and mental models that we found useful in our own vulnerability research
- Think of this talk as a bootstrap session :)



What is CodeQL?

An expressive query language and engine for code analysis

- Treats code as data
- Lets you describe and find patterns in the code
- Toolchain: CLI and IDE

What can I do with it?





- Find bugs and security vulnerabilities
- Quickly make your analyses more precise
- Assist you during code reviews
- Share security knowledge within your teams using codified, readable and executable queries

CodeQL is...

- Logical
- Declarative no side effects
- Object-oriented
- Read-only
- Equipped with rich standard libraries for analyzing source code

What does a query look like?





Import: lets us reuse logic defined in other libraries

```
import java
```

```
from IfStmt ifStmt, Block block
where
  block = ifStmt.getThen() and
  block.getNumStmt() = 0
select ifStmt, "This if-statement has an empty then block."
```

Query clause: describes what we are trying to find

Building blocks of a query





Predicates

Like functions, but better!

Create reusable logic and give it a name.

Just a query

```
from IfStmt ifStmt, Block block
where
   block = ifStmt.getThen() and
   block.getNumStmt() = 0
select ifStmt
```

Using a predicate

```
predicate isEmpty(Block block) {
  block.getNumStmt() = 0
}

from IfStmt ifStmt
where isEmpty(ifStmt.getThen())
select ifStmt
```

Classes

Describe a set of values.

Using a predicate

```
predicate isEmpty(Block block) {
   block.getNumStmt() = 0
}
from IfStmt ifStmt
where isEmpty(ifStmt.getThen())
select ifStmt
```

Using a class

```
class EmptyBlock extends Block {
  EmptyBlock() {
    this.getNumStmt() = 0
from IfStmt ifStmt
where ifStmt.getThen() instanceof
EmptyBlock
select ifStmt
```

Using a predicate

```
predicate isEmpty(Block block) {
   block.getNumStmt() = 0
}

from IfStmt ifStmt
where isEmpty(ifStmt.getThen())
select ifStmt
```

Using a class

```
class EmptyBlock extends Block {
   EmptyBlock() {
     this.getNumStmt() = 0
   }
}
from IfStmt ifStmt, EmptyBlock block
where ifStmt.getThen() = block
select ifStmt
```

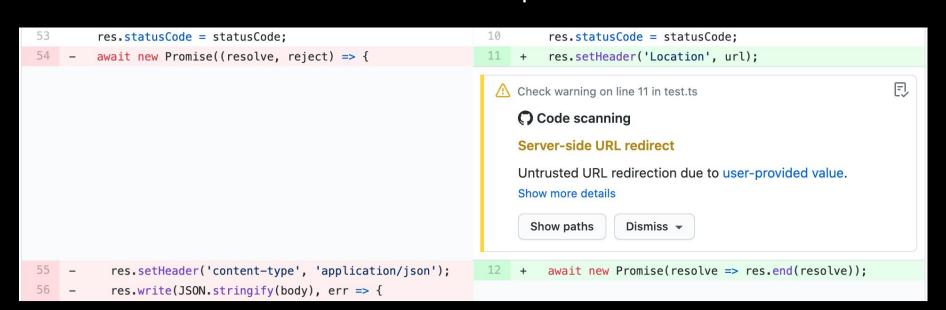
Common uses of CodeQL

- 1. Automated scans with built-in or custom queries
- 2. Variant analysis
- 3. Assist during manual code review



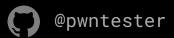
Automated scans

- Toll gates \(\biggriam \) or guardrails \(\daggreen \)
- Ideally integrated into the build pipeline
 - Run on every commit or PR
 - Immediate feedback to developers



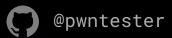
Variant analysis

- Take a known vulnerability and model its characteristics into a CodeQL query
- Run that query across a bunch of projects to find similar vulnerabilities
- Example: <u>Bean Stalking: Growing Java beans into RCE</u>
 - Variant Analysis of CVE-2018-16621 (Bean Validation EL injection) by opening-2018-16621 (Bean Validation EL
 - Multiple RCE (Netflix Conductor, Netflix Titus, Apache Syncope, Sonatype Nexus, Corona-Warn-App, ...)



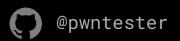
Code Auditing

- The iterative process of answering questions about code to establish vulnerability hypotheses
 - pwndering: verb, "can this or that happen in such and such way that could lead to a vuln?"
- CodeQL can help answer those questions
 - Challenges:
 - Knowing WHAT you want to ask
 - Knowing HOW to ask it
- Clear questions translate into clear queries
- Fuzzy questions translate into fuzzy queries



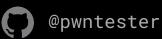
Ask the OraQL





Ask the OraQL



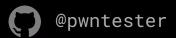


Common Auditing Questions

- What is my attack surface?
- Is this code even executed?
- Where does my user-controlled input end up?
- Which integer arithmetic can I influence?
- Which parts of the code are high in bug density?
- •



Case study



apache/dubbo

Apache Dubbo is a high-performance, java based, open source RPC framework.



AR 355

885

☆ 36k

앟 24k

Forks



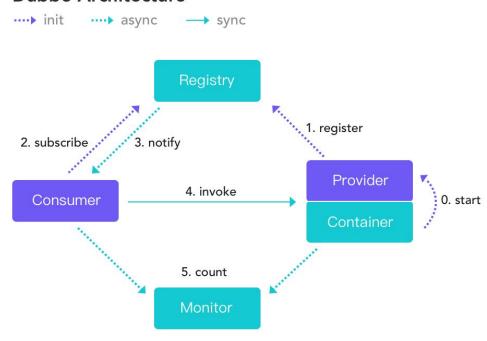
Contributors

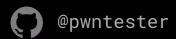
Issues

Stars

@pwntester

Dubbo Architecture





Past vulnerabilities 🐛



CVE-2020-11995

A Hessian2 deserialization vulnerability could lead to malicious code execution. This vulnerability was addressed by establishing a mechanism for users to set deserialization allow/block lists.

CVE-2020-1948

An attacker can send RPC requests with an unrecognized service name or method name along with some malicious parameter payloads. When the malicious parameter is deserialized, it will execute some malicious code.

CVE-2019-17564

Unsafe deserialization occurs within a Dubbo application which has Spring HTTP remoting enabled.



New from this audit

GHSL-2021-035	CVE-2021-25641	Bypass Hessian2 allowlist via alternative protocols	
GHSL-2021-036	60	Pre-auth RCE via multiple Hessian deserializations in the RPC invocation decoder	
GHSL-2021-037	CVE-2021-30179	Pre-auth RCE via Java deserialization in the Generic filter	
GHSL-2021-038	CVE-2021-30179	Pre-auth RCE via arbitrary bean manipulation in the Generic filter	
GHSL-2021-039	CVE-2021-32824	Pre-auth RCE via arbitrary bean manipulation in the Telnet handler	
GHSL-2021-040	CVE-2021-30180	RCE on customers via Tag route poisoning (Unsafe YAML unmarshaling)	
GHSL-2021-041	CVE-2021-30180	RCE on customers via Condition route poisoning (Unsafe YAML unmarshaling)	
GHSL-2021-042	CVE-2021-30181	RCE on customers via Script route poisoning (Nashorn script injection)	
GHSL-2021-043	CVE-2021-30180	RCE on providers via Configuration poisoning (Unsafe YAML unmarshaling)	
GHSL-2021-094	CVE-2021-36162	RCE on customers via MeshApp route poisoning (Unsafe YAML unmarshaling)	
GHSL-2021-095	CVE-2021-36163	Pre-Auth Unsafe Hessian deserialization when Hessian protocol is used	
GHSL-2021-096	33	Pre-Auth Unsafe Java deserialization when RMI protocol is used	
GHSL-2021-097	CVE-2021-37579	Bypass `checkSerialization` security control	



Ex1: The attack surface



Use semmle.code.java.dataflow.FlowSources.RemoteFlowSource

```
import java
import semmle.code.java.dataflow.FlowSources

from RemoteFlowSource source
select source
```

- Exclude those ones with paths matching */src/test/*
- Select the source, enclosing class and source type



```
import java
import semmle.code.java.dataflow.FlowSources
```

```
from RemoteFlowSource source where
```

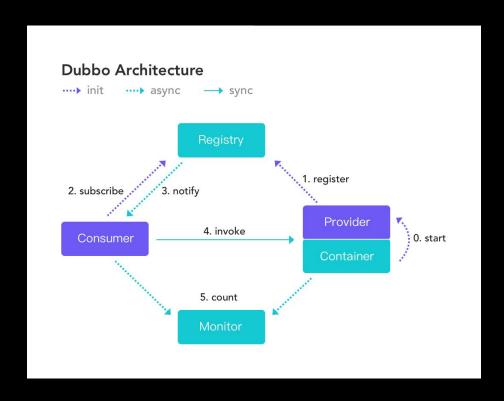
not source.getLocation().getFile().getRelativePath().matches("%/src/test/%")
select source, source.getEnclosingCallable().getDeclaringType(), source.getSourceType()

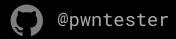
#select ✓				
	source	[1]	[2]	
1	getHostName()	NetUtils	reverse DNS lookup	
2	getHostName()	NetUtils	reverse DNS lookup	
3	getContent()	HttpClientConnection	external	
4	getRequestURI()	HessianHandler	external	
5	getHeaderNames()	HessianHandler	external	
6	getHeader()	HessianHandler	external	
7	getInputStream()	HessianHandler	external	
8	getRequestURI()	InternalHandler	external	
9	getInputStream()	InternalHandler	external	
10	getRequestURI()	InternalHandler	external	



Identifying attack surface

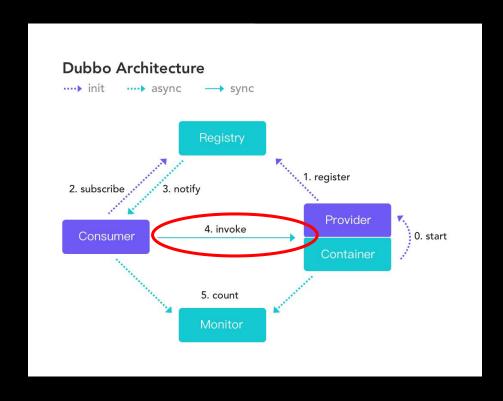


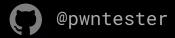




Identifying attack surface

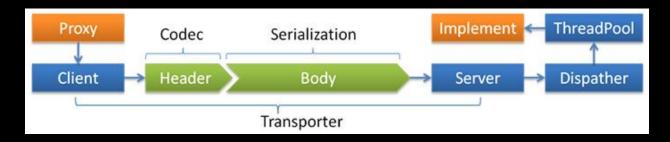


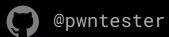




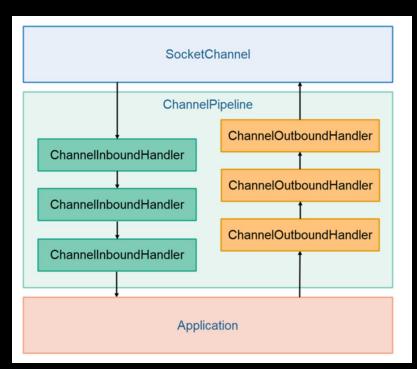


- Dubbo transporters
 - https://dubbo.apache.org/docs/v2.7/dev/impls/remoting/
 - Netty (default one)
 - Mina
 - Grizzly



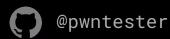


Netty 101



Package	kage io.netty.channel						
Class	ChannelInboundHandler						
Method	channelRead & channelRead0						
Parameters	ChannelHandlerContext ctx Object msg						

Package	io.netty.handler.codec				
Class	ByteToMessageDecoder				
Method	decode & decodeLast				
Parameters	ChannelHandlerContext ctx ByteBuf in List <object> out</object>				



Ex2: Model Netty sources

Model channelRead and decode sources. Eg:

```
import semmle.code.java.dataflow.FlowSources
class XXXXXX extends RemoteFlowSource {
    XXXXXX() {
        none()
   override string getSourceType() {
        result = "<Source description>"
```

```
class ChannelInboundHandler extends Class {
  ChannelInboundHandler() {
    this.getASourceSupertype*().hasQualifiedName("io.netty.channel", "ChannelInboundHandler")
/** The ChannelInboundHandlerl.channelRead method */
class ChannelReadMethod extends Method {
  ChannelReadMethod() {
      this.qetName() = ["channelRead", "channelRead0", "messageReceived"] and
      this.getDeclaringType() instanceof ChannelInboundHandler
                                                                     #select >
                                                                                                                            6 results
                                                                                    HttpProcessHandler
                                                                                                           Netty Handler Source
                                                                         msg
                                                                                    QosProcessHandler
                                                                                                           Netty Decoder Source
class ChannelReadSource extends RemoteFlowSource {
                                                                                    TelnetProcessHandler
                                                                                                           Netty Handler Source
                                                                         msq
    ChannelReadSource() {
                                                                                    NettyClientHandler
                                                                                                           Netty Handler Source
                                                                         msq
      exists(ChannelReadMethod m
                                                                                    InternalDecoder
                                                                                                           Netty Decoder Source
                                                                         input
         this.asParameter() = m.getParameter(1)
                                                                                                           Netty Handler Source
                                                                                    NettyServerHandler
                                                                         msg
    override string getSourceType() { result = "Netty Handler Source" }
```

Dubbo abstractions

- Transport layer abstraction.
 - Netty, Mina and Grizzly call:
 - ExchangeCodec.decode()
 - DubboCodec.decodeBody(Channel c, InputStream is, byte[] hdrs)
- Serializer abstraction:
 - Serialization class is initialized for a given serialization type
 - Serialization.deserialize() wraps an input stream with a concrete
 ObjectInput type
 - Eg: JavaObjectInput, KryoObjectInput, etc.
 - Note: It *does not* perform any deserializations
 - ObjectInput.readXXX methods perform the actual deserialization

```
Serialization serialization = CodecSupport.getSerialization(channel.getUrl(), serializationType)
ObjectInput in = serialization.deserialize(channel.getUrl(), inputStream);
in.readObject();
```

Dubbo abstractions

- Transport layer abstraction.
 - Netty, Mina and Grizzly will end up calling
 - ExchangeCodec.decode() -
 - DubboCodec.decodeBody(Channel c, InputStream is, byte[] hdrs)

Our source

- Serializer abstraction:
 - Serialization class is initialized for a given serialization type
 - Serialization.deserialize() wraps an input stream with a concrete
 ObjectInput type
 - Eg: JavaObjectInput, KryoObjectInput, etc.
 - Note: It *does not* perform any deserializations
 - ObjectInput.readXXX methods perform the actual deserialization

```
Serialization serialization = CodecSupport.getSerialization(channel.getUrl(), serializationType)

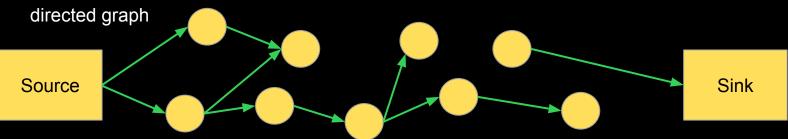
ObjectInput in = serialization.deserialize(channel.getUrl(), inputStream);

in.readObject();

Our sink
(qualifier)

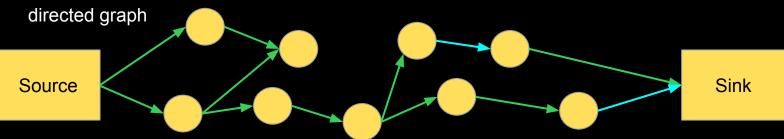
A taint step from param 1 to return
```

- Emulate program run and track data from an origin (source) to a destination (sink)
- Taint Tracking analysis requires models (summaries) for method for which we don't have the source code
- We need to configure the CodeQL TaintTracking module using a TaintTracking::Configuration which is mostly boilerplate to define:
 - Sources (nodes)
 - Sinks (nodes)
 - TaintSteps (optional edges)
 - Sanitizers (optional nodes)
- Dataflow nodes are an abstraction of AST nodes (expressions or parameters) and form a



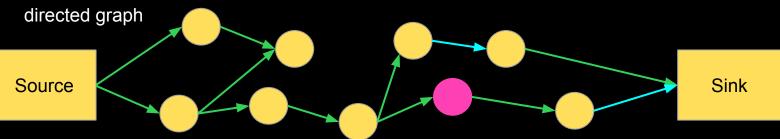


- Emulate program run and track data from an origin (source) to a destination (sink)
- Taint Tracking analysis requires models (summaries) for method for which we don't have the source code
- We need to configure the CodeQL TaintTracking module using a TaintTracking::Configuration which is mostly boilerplate to define:
 - Sources (nodes)
 - Sinks (nodes)
 - TaintSteps (optional edges)
 - Sanitizers (optional nodes)
- Dataflow nodes are an abstraction of AST nodes (expressions or parameters) and form a



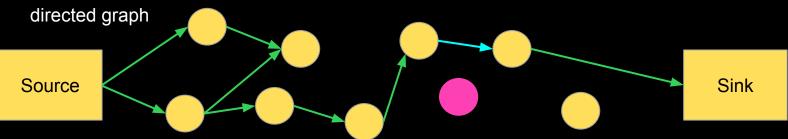


- Emulate program run and track data from an origin (source) to a destination (sink)
- Taint Tracking analysis requires models (summaries) for method for which we don't have the source code
- We need to configure the CodeQL TaintTracking module using a TaintTracking::Configuration which is mostly boilerplate to define:
 - Sources (nodes)
 - Sinks (nodes)
 - TaintSteps (optional edges)
 - Sanitizers (optional nodes)
- Dataflow nodes are an abstraction of AST nodes (expressions or parameters) and form a





- Emulate program run and track data from an origin (source) to a destination (sink)
- Taint Tracking analysis requires models (summaries) for method for which we don't have the source code
- We need to configure the CodeQL TaintTracking module using a TaintTracking::Configuration which is mostly boilerplate to define:
 - Sources (nodes)
 - Sinks (nodes)
 - TaintSteps (optional edges)
 - Sanitizers (optional nodes)
- Dataflow nodes are an abstraction of AST nodes (expressions or parameters) and form a





```
* @kind path-problem
import java
import semmle.code.java.dataflow.TaintTracking
import DataFlow::PathGraph
class MyConfig extends TaintTracking::Configuration {
 MyConfig() { this = "MyConfig" }
  override predicate isSource(DataFlow::Node source) {
          Source
  override predicate isAdditionalTaintStep(DataFlow::Node n1, DataFlow::Node n2) {
        Taint Steps
  override predicate isSink(DataFlow::Node sink) {
           Sink
from MyConfig conf, DataFlow::PathNode source, DataFlow::PathNode sink
where conf.hasFlowPath(source, sink)
```

select sink, source, sink, "source flows into sink"

Ex3: Variant analysis

Source:

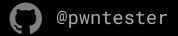
```
org.apache.dubbo.rpc.protocol.dubbo.DubboCodec.decodeBody (param 1,2)
```

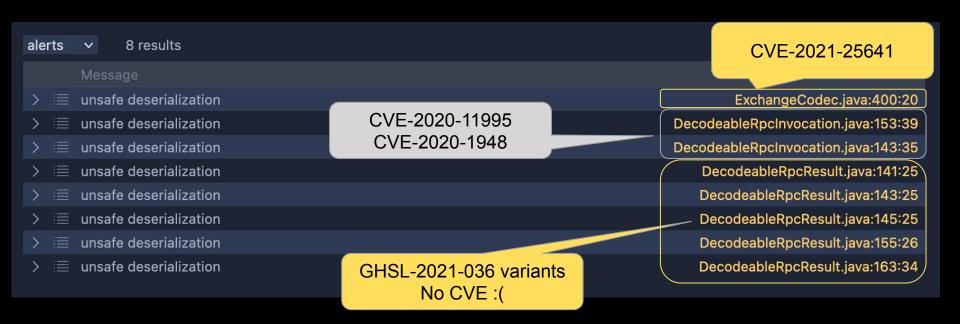
TaintStep:

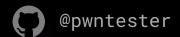
```
org.apache.dubbo.common.serialize.Serialization.deserialize (arg 1 -> return)
```

Sink:

```
org.apache.dubbo.common.serialize.ObjectInput.read* (qualifier)
```





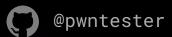


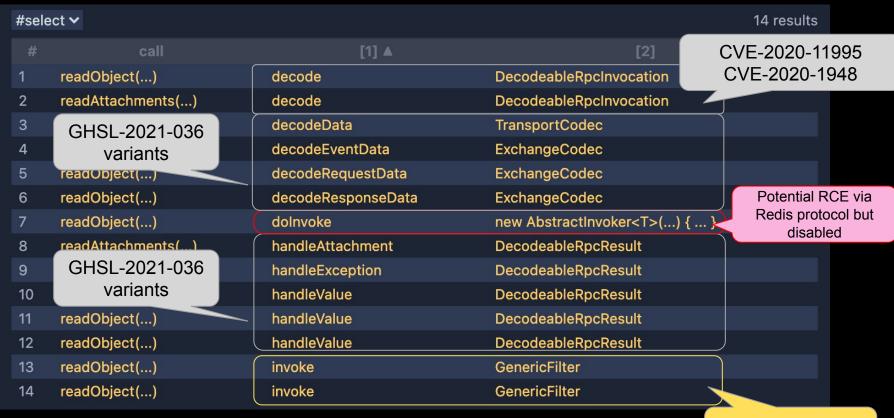
			Dubbo Protoco	1															
Offsets	Octet	0	1				- 83	2							- ;	3			
Octet	Bit	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Magic High	Magic Low	R e q / R e s	2 W a y	g v e n t	Se	eri	ali:		lon		Status		2				
4	32																		
8	64	RPC Request ID																	
12	96	Data Length																	
16	128	Variable length part, in turn, is: dubbo version, service name, service version, method name, parameter types, arguments, attachments																	

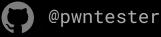


Ex4: Semantic matches

- Taint tracking analysis requires modelling of all APIs involved in the taint propagation
- Out of the shelve libraries may not model all required APIs
- As part of a manual code review we want to verify dangerous ops regardless of user control evidences
- Find all calls to ObjectInput.read*() methods semantically
- Exclude calls to read* within the ObjectInput class itself
- Exclude calls on files with a path matching */src/test/*
- Output the call, the enclosing method and the enclosing class

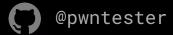


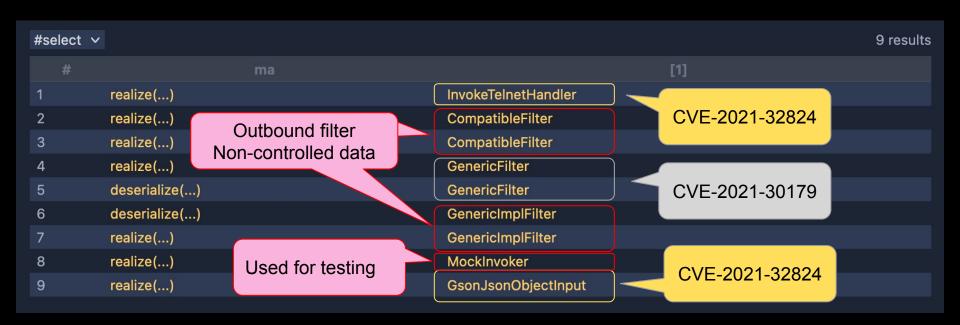


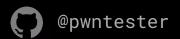


Ex5: Scaling manual findings

- GenericFilter was also vulnerable to arbitrary setter calls via PojoUtils and JavaBeanSerializeUtil
- Are they used anywhere else?
- Find (semantically) all uses of:
 - PojoUtil.realize()
 - JavaBeanSerializeUtil.deserialize()
- As usual exclude results on test files

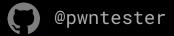






Ex6: Semantic sinks heatmap

- A best practice is to look for semantic (non-dataflow) matches of the most important taint tracking sinks
- This will give us a heatmap of where dangerous operations take place and therefore a map of files to audit more carefully
- Find all calls to an unsafe deserialization sinks known to CodeQL
 - Reuse UnsafeDeserializationSink class
 - From semmle.code.java.security.UnsafeDeserializationQuery
- Select sink class, method and call enclosing class



#select > 23 results Krvo readObjectOrNull(...) KryoObjectInput2 2 Kryo readObjectOrNull(...) KryoObjectInput2 Krvo readObjectOrNull(...) KryoObjectInput2 Hessian2Input readObject(...) Hessian2ObjectInput 4 Hessian2ObjectInput Hessian2Input readObject(...) ObjectInputStream readObject(...) NativeJavaObjectInput ObjectInputStream readObject(...) **JavaObjectInput** 8 Hessian2Input readObject(...) Hessian2ObjectInput 9 Hessian2Input readObject(...) Hessian2ObjectInput readClassAndObject(...) KryoObjectInput2 Krvo Kryo readClassAndObject(...) KryoObjectInput **JSON** parseObject(...) 12 FastJsonObjectInput **JSON** parseObject(...) FastJsonObjectInput **JSON** parseObject(...) MockInvoker 14 **JSON** MockInvoker parseObject(...) 16 **JSON** parseObject(...) new Function<String,DefaultServiceInstance>(...) { ... } **JSON** parseObject(...) NacosDynamicConfiguration 18 **JSON** parse(...) FastJsonObjectInput 19 MockInvoker **JSON** parse(...) JSON 20 parse(...) ServiceInstanceMetadataUtils 21 Yaml **TagRuleParser** load(...) CVE-2021-30180 ConditionRuleParser Yaml load(...) Yaml load(...) ConfigParser

Bonus track: reverse flow

 In many occasions, we want to hoist a sink to figure out what other methods also behave as sinks

```
public void foo(String str) {
 bar(str);
public void bar(String str) {
  baz(str);
public void baz(String str) {
  sink(str);
```



Bonus track: reverse flow

 In many occasions, we want to hoist a sink to figure out what other methods also behave as sinks

```
public void foo(String str) {
  bar(str); \checkmark
public void bar(String str) {
  baz(str); \checkmark
public void baz(String str) {
  sink(str);
```

 We can consider the followings as sinks:

```
o foo()
```



Bonus track: reverse flow

• Hoist the Yaml.load() method to see where its coming from

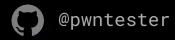
#sele	ect 🗸			12 results
	dist	[1]	[2]	n ▲
1	3	NacosConfigListener	innerReceive	configInfo
2	3	ConfigChangedEvent	ConfigChangedEvent	content
3	3	ConfigChangedEvent	ConfigChangedEvent	content
4	1	ListenableRouter	process	event [content]
5	1	TagRouter	process	event [content]
6	3	AbstractConfiguratorListener	process	event [content]
7	3	Optional	orElse	p0
8	1	ConfigParser	parseConfigurators	rawConfig
9	2	AbstractConfiguratorListener	genConfiguratorsFromRawRule	rawConfig
10	5	AppResponse	AppResponse	result
11	3	new Consumer <string>() { }</string>	accept	V
12	3	CacheListener	dataChanged	value



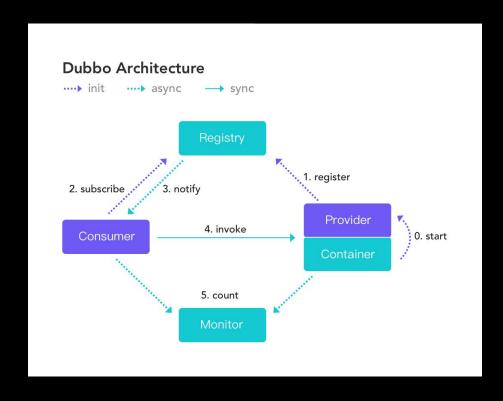
Configuration Centers

- Dataflows to the insecure YAML.load() are originated from Configuration Centers (CC) listeners.
- Dubbo CC currently supports ZooKeeper, Nacos, Etcd, Consul and Apollo
- An attacker being able to modify arbitrary configuration objects on the CC may trigger unsafe deserialization ops.
- Hint:
 - Nacos authentication was bypassable (CVE-2021-29441)
 - https://securitylab.github.com/advisories/GHSL-2020-325_326-nacos/
 - Many ZooKeeper instances run with no authentication



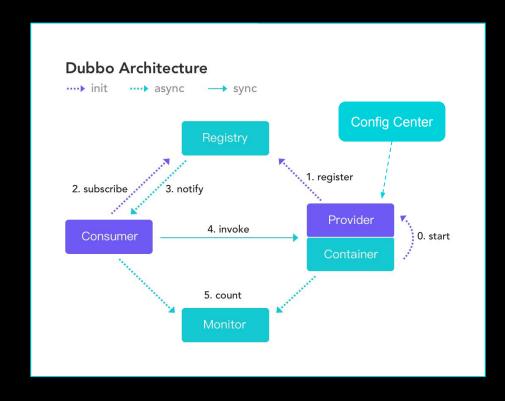






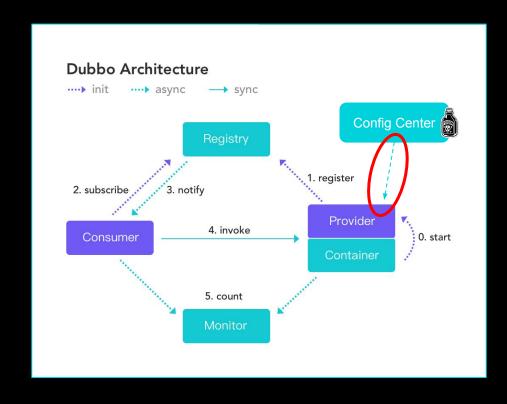






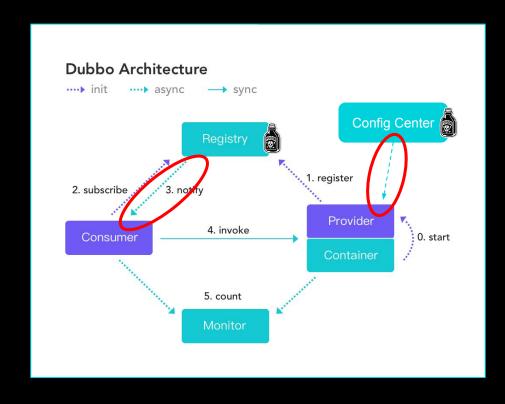














Ex7: Configuration Centers

- Dubbo uses one more abstraction for CCs and another for the Registry
- Model Dubbo CC abstraction as a new source

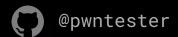
Package	org.apache.dubbo.common.config.configcenter
Class	ConfigurationListener
Method	process
Parameters	ConfigChangedEvent

Model Dubbo Registry abstraction as a new source

Package	org.apache.dubbo.registry
Class	NotifyListener
Method	notify
Parameters	List <url></url>

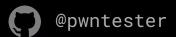


		[1] ▲	[2]
1	urls	notify	ReverseCompatibleNotifyListener
2	urls	notify	RegistryDirectory
3	urls	notify	OverrideListener
4	arg0	notify	new NotifyListener() { }
5	urls	notify	MultipleNotifyListenerWrapper
6	urls	notify	SingleNotifyListener
7	event	process	ListenableRouter
8	event	process	TagRouter
9	event	process	new ConfigurationListener() { }
10	event	process	AbstractConfiguratorListener

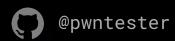


Ex8: Script Injection

- Does the new attack surface lead to new issues?
- You may want to run an scan with the complete (default + experimental) set of CodeQL queries and add the new sources and taint steps you developed during the workshop
- Copy the ScriptInjection query
 - https://github.com/github/codeql/blob/main/java/ql/src/experimental/Security/CWE/CWE-0
 94/ScriptInjection.ql
- Import sources from Exercise 7
 - o import dubbo
- Import few JDK models not yet merged to CodeQL repo
 - import models



				OVE 0004 00404	
		Message		CVE-2021-30181	
~		Java Scrip	ot Engine evaluate user input.		ScriptRouter.java:73:24
	~	Path			
		1	urls : List		RegistryDirectory.java:229:37
		2	categoryUrls : Map		RegistryDirectory.java:239:32
		3	getOrDefault() : Object		RegistryDirectory.java:239:32
		4	routerURLs : Object		RegistryDirectory.java:240:19
		5	urls : Object		RegistryDirectory.java:371:46
		6	url : URL		RegistryDirectory.java:386:58
		7	url : URL		ScriptRouterFactory.java:41:29
		8	url : URL		ScriptRouterFactory.java:42:33
		9	url : URL		ScriptRouter.java:65:25
		10	url : URL		ScriptRouter.java:70:24
		11	getRule(): String		ScriptRouter.java:70:16
		12	rule		ScriptRouter.java:73:43
	>	Path			
>		Java Scrip	t Engine evaluate <mark>user input</mark> .		ScriptRouter.java:73:24



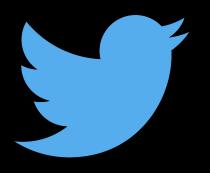
Contribute your own queries and make some & https://securitylab.github.com/get-involved



\$184.000 bounties so far



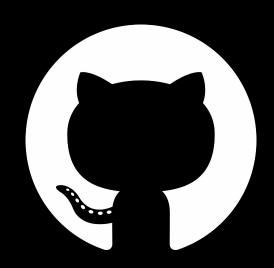
Thank you!



Reach out on twitter:

- @pwntester
- @ghsecuritylab





Tips & Tricks



Start small

- If you have a great idea test it first in a small synthetic example.
- This will avoid you a lot of head-scratching.
- Complex queries can take hours to finish.
 - Make sure your query finds what you are looking for in an example database.



Logic issues

- A good amount of issues come from trying to express what we want in logical terms.
- Doing this can be confusing.
- Make sure to undust your logic book and properly understand quantifiers, etc.



Debugging

- There is no "gdb" for codeql.
- Best tool you have is to break complex queries into small pieces.
- Each small piece can be executed in isolation.



AST Viewer

- The AST viewer allows you to see how codeqle represents your code.
- This allows you to quickly identify the components involved in the pattern you are trying to query.
- Once you know the type in the AST you can import it in your CodeQl query.
- Support is limited to a handful of languages but soon it will work across all the supported languages.



AST Viewer

```
V AST VIEWER
                                       三 句
                                                 1497
 AST for TcpDiscoverySpi.java
                                                              /** {@inheritDoc} */
                                                1498
                                                              @Override public IgniteSpiContext getSpiContext() {
   ∨ [Method] getSpiContext Line 1499
                                                 1499
                                                                   if (ctxInitLatch.getCount() > 0) {
    > (Javadoc)
                                                 1500
    > (Annotations)
                                                                        if (log.isDebugEnabled())
                                                 1501
      [TypeAccess] IgniteSpiContext Line 1499
                                                                             log.debug("Waiting for context initialization.
                                                 1502

    [BlockStmt] stmt Line 1499

                                                 1503
     ∨ [IfStmt] stmt Line 1500
                                                 1504
                                                                        trv
      > [GTExpr] ... > ... Line 1500
                                                 1505
                                                                             U.await(ctxInitLatch);
        [BlockStmt] stmt Line 1500
                                                 1506
       > [IfStmt] stmt Line 1501
                                                 1507
                                                                             if (log.isDebugEnabled())
       > [TryStmt] stmt Line 1504
                                                                                  log.debug("Context has been initialized.");
                                                 1508
     ∨ [ReturnStmt] stmt Line 1515
                                                1509
      [MethodAccess] getSpiContext(...) Line 1515
                                                                        catch (IgniteInterruptedCheckedException e) {
         [SuperAccess] super Line 1515
                                                1510
     [Method] getLocalNode Line 1519
                                                1511
                                                                             U.warn(log,
     [Method] setListener Line 1524
                                                 1512
     [Method] setDataExchange Line 1529
                                                 1513
     [Method] setMetricsProvider Line 1534
                                                 1514
```

