Apache Solr RCE—【CVE-2019-0192】

orich1 / 2019-03-13 08:15:00 / 浏览数 4171 安全技术 漏洞分析 顶(3) 踩(0)

前言

第二次分析 solr 了

漏洞详情：https://issues.apache.org/jira/browse/SOLR-13301
漏洞POC：https://github.com/mpgn/CVE-2019-0192

影响版本：5.0.0 ~ 5.5.5、6.0.0 ~ 6.6.5
安全版本：7.0

简要描述

```
ConfigAPI allows to configure Solr's JMX server via an HTTP POST request.By pointing it to a malicious RMI server, an attacker
```

调用栈

Solr 中由配置文件触发 Jmx流程：
org.apache.solr.handler.SolrConfigHandler#handleRequestBody
->
org.apache.solr.handler.SolrConfigHandler.Command#handlePOST
->
org.apache.solr.handler.SolrConfigHandler.Command#applySetProp
->
org.apache.solr.core.CoreContainer#reload
->
org.apache.solr.core.SolrConfig#SolrConfig
->
org.apache.solr.core.SolrCore#reload
->
org.apache.solr.core.SolrCore#SolrCore
->
org.apache.solr.core.SolrCore#initInfoRegistry
->
org.apache.solr.core.JmxMonitoredMap#JmxMonitoredMap
->
javax.management.remote.JMXConnectorServerMBean#start

分析过程

其路由过程不再分析，在 https://xz.aliyun.com/t/1523 这篇文章中分析过

直接到 ReuqestHandler 中，访问路径 /config 可以进入 SolrConfigHandler#handleRequestBody 中，如下图：

```
public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp) throws Exception {

  setWt(req, CommonParams.JSON);
  String httpMethod = (String) req.getContext().get("httpMethod");
  Command command = new Command(req, rsp, httpMethod);
  if ("POST".equals(httpMethod)) {
    if (configEditing_disabled || isImmutableConfigSet) {
      final String reason = configEditing_disabled ? "due to " + CONFIGSET_EDITING_DISABLED_ARG : "because ConfigSet is immutable";
      throw new SolrException(SolrException.ErrorCode.FORBIDDEN, " solrconfig editing is not enabled " + reason);
    }
    try {
      command.handlePOST();
    } finally {
      RequestHandlerUtils.addExperimentalFormatWarning(rsp);
    }
  } else {
    command.handleGET();
  }
}
```

由poc知道我们走的是 POST 的处理流程，跟进 handlePOST 函数

```
private void handlePOST() throws IOException {
  List<CommandOperation> ops = CommandOperation.readCommands(req.getContentStreams(), resp);
  if (ops == null) return;
  try {
    for (; ; ) {
      ArrayList<CommandOperation> opsCopy = new ArrayList<>(ops.size());
      for (CommandOperation op : ops) opsCopy.add(op.getCopy());
      try {
        if (parts.size() > 1 && RequestParams.NAME.equals(parts.get(1))) {
          RequestParams params = RequestParams.getFreshRequestParams(req.getCore().getResourceLoader()
          handleParams(opsCopy, params);
        } else {
          ConfigOverlay overlay = SolrConfig.getConfigOverlay(req.getCore().getResourceLoader());
          handleCommands(opsCopy, overlay);
        }
        break;//succeeded . so no need to go over the loop again
      } catch (ZkController.ResourceModifiedInZkException e) {
```

如上图看见关键词 Config Overlay ，跟进 else 代码块调用的 handleCommands 函数，如下

```
private void handleCommands(List<CommandOperation> ops, ConfigOverlay overlay) throws IOException {
  for (CommandOperation op : ops) {
    switch (op.name) {
      case SET_PROPERTY:
        overlay = applySetProp(op, overlay);
        break;
      case UNSET_PROPERTY:
        overlay = applyUnset(op, overlay);
        break;
      case SET_USER_PROPERTY:
        overlay = applySetUserProp(op, overlay);
        break;
      case UNSET_USER_PROPERTY:
        overlay = applyUnsetUserProp(op, overlay);
        break;
      default: {
        List<String> pcs = StrUtils.splitSmart(op.name.toLowerCase(Locale.ROOT),  separator: '-');
        if (pcs.size() != 2) {
          op.addError(formatString( pattern: "Unknown operation ''{0}'' ", op.name));
```

第一个框就是 poc 给出的 POST 数据中指定的 set-property 流程，但是我感觉第二个框也可以，不过具体情况没测试

跟入 applySetProp 函数，如下

```
private ConfigOverlay applySetProp(CommandOperation op, ConfigOverlay overlay) {
     Map<String, Object> m = op.getDataMap();
     if (op.hasError()) return overlay;
     for (Map.Entry<String, Object> e : m.entrySet()) {
       String name = e.getKey();
       Object val = e.getValue();
       Class typ = ConfigOverlay.checkEditable(name, false, null);
[……■■■■■■■ POST ■■■■■……]
       overlay = overlay.setProperty(name, val);
     }
     return overlay;
   }

 }
```

如上，注意力转向 setProperty 的调用，跟进

```
public ConfigOverlay setProperty(String name, Object val) {
  List<String> hierarchy = checkEditable(name,  isXPath: false,  failOnError: true);
  Map deepCopy = (Map) Utils.fromJSON(Utils.toJSON(props));
  Map obj = deepCopy;
  for (int i = 0; i < hierarchy.size(); i++) {
    String s = hierarchy.get(i);
    if (i < hierarchy.size() - 1) {
      if (obj.get(s) == null || (!(obj.get(s) instanceof Map))) {
        obj.put(s, new LinkedHashMap<>());
      }
      obj = (Map) obj.get(s);
    } else {
      obj.put(s, val);
    }
  }

  Map<String, Object> jsonObj = new LinkedHashMap<>(this.data);
  jsonObj.put("props", deepCopy);

  return new ConfigOverlay(jsonObj, znodeVersion);
}
```

这里就是将 POST 参数做处理后，全部用于生成一个新的 ConfigOverlay 对象并将其返回到 handleCommands 函数中的 overlay 变量去接受赋值

标注 handleCommands 中的 overlay 变量，看到如下调用

```
SolrResourceLoader loader = req.getCore().getResourceLoader();
if (loader instanceof ZkSolrResourceLoader) {
  int latestVersion = ZkController.persistConfigResourceToZooKeeper((ZkSolrResourceLoader) loader, overlay.getZnodeVersion(),
      ConfigOverlay.RESOURCE_NAME, overlay.toByteArray(),  createIfNotExists: true);
  log.info("Executed config commands successfully and persisted to ZK {}", ops);
  waitForAllReplicasState(req.getCore().getCoreDescriptor().getCloudDescriptor().getCollectionName(),
      req.getCore().getCoreContainer().getZkController(),
      ConfigOverlay.NAME,
      latestVersion,  maxWaitSecs: 30);
} else {
  SolrResourceLoader.persistConfLocally(loader, ConfigOverlay.RESOURCE_NAME, overlay.toByteArray());
  req.getCore().getCoreContainer().reload(req.getCore().getName());
  log.info("Executed config commands successfully and persited to File System {}", ops);
}
```

If 代码块是处于 zookeeper 分布式状态下的，也能触发成功，不过为了便于分析，我们跟进 else 代码块：单机孤儿模式

可以看见调用了 persistConfLocally 猜测应该是将刚刚生成的 ConfigOverlay 对象的数据做持久存储，其中 ConfigOverlay.RESOURCE_NAME 的值为：

```
public static final String RESOURCE_NAME = "configoverlay.json";
```

数据应该是存入了 configoverlay.json 中
第二步就调用了 reload 函数，重启服务器

其实这里我是反向跟踪的，因为 reload 流程中会有大量的 initial 操作，根本分不清啥时候会加载我们指定的配置，从 ConfigOverlay.RESOURCE_NAME 入手

```
public static final String RESOURCE_NAME = "configoverlay.json";

/*private static final Long STR_ATTR
private static final Long STR_NODE =
private static final Long BOOL_ATTR =
private static final Long BOOL_NODE =
private static final Long INT_ATTR =
private static final Long INT_NODE =
```

Usages of RESOURCE_NAME in All Places (4 usages fou

SolrConfig.java	407	in = loader.openResource(ConfigOverlay.**RESOURCE_NAME**);
SolrConfigHandler.java	500	ConfigOverlay.**RESOURCE_NAME**, overlay.toByteArray(), true);
SolrConfigHandler.java	507	SolrResourceLoader.persistConfLocally(loader, ConfigOverlay.**RESOURCE**
SolrCore.java	2948	final String overlayPath = zkSolrResourceLoader.getConfigSetZkPath() +

这里有个获取流数据操作，八九不离十就是这里，但是为了调用流程直观，我们还是从 reload 流程中讲述

```
public void reload(String name) {
  SolrCore core = solrCores.getCoreFromAnyList(name,  incRefCount: false);
  if (core != null) {

    // The underlying core properties files may have changed, we don't really know. So we have
    // CoreDescriptor and we need to reload it from the disk files
    CoreDescriptor cd = reloadCoreDescriptor(core.getCoreDescriptor());
    solrCores.addCoreDescriptor(cd);
    try {
      solrCores.waitAddPendingCoreOps(cd.getName());
      ConfigSet coreConfig = coreConfigService.getConfig(cd);
      log.info("Reloading SolrCore '{}' using configuration from {}", cd.getName(), coreConfig.
      SolrCore newCore = core.reload(coreConfig);
      registerCore(cd, newCore,  registerInZk: false,  skipRecovery: false);
```

注意到了这一句，重新加载的话，所有配置文件也会重新加载一次，既然之前写入了 configoverlay.json 文件中，那也算作是配置文件，跟进 getConfig 函数

```
public final ConfigSet getConfig(CoreDescriptor dcore) {

  SolrResourceLoader coreLoader = createCoreResourceLoader(dcore);

  try {

    // ConfigSet properties are loaded from ConfigSetProperties.DEFAULT_FILENAME file.
    // ConfigSet flags are loaded from the metadata of the ZK node of the configset.
    NamedList properties = createConfigSetProperties(dcore, coreLoader);
    NamedList flags = getConfigSetFlags(dcore, coreLoader);

    boolean trusted =
        (coreLoader instanceof ZkSolrResourceLoader
            && flags != null
            && flags.get("trusted") != null
            && !flags.getBooleanArg( name: "trusted")
            ) ? false: true;

    SolrConfig solrConfig = createSolrConfig(dcore, coreLoader);
    IndexSchema schema = createIndexSchema(dcore, solrConfig);
    return new ConfigSet(configName(dcore), solrConfig, schema, properties, trusted);
```

很显眼 createSolrConfig 函数流程肯定是重加载配置，一直跟踪到 SolrConfig#SolrConfig 中，如下：

```
*/
public SolrConfig(SolrResourceLoader loader, String name, InputSource is)
    throws ParserConfigurationException, IOException, SAXException {
  super(loader, name, is,  prefix: "/config/");
  getOverlay();//just in case it is not initialized
  getRequestParams();
  initLibs();
```

看见 overlay 了，跟进到 getConfigOverlay 函数如下

```
public static ConfigOverlay getConfigOverlay(SolrResourceLoader loader) {
  InputStream in = null;
  InputStreamReader isr = null;
  try {
    try {
      in = loader.openResource(ConfigOverlay.RESOURCE_NAME);
    } catch (IOException e) {
```

这里就刚好和前面搜索到的流数据操作吻合，那么继续在 SolrConfig 的构造函数中查看，找到如下

```
280
281     Node jmx = getNode( path: "jmx",  errifMissing: false);
282     if (jmx != null) {
283       jmxConfig = new JmxConfiguration( enabled: true,
284           get( path: "jmx/@agentId",  def: null),
285           get( path: "jmx/@serviceUrl",  def: null),
286           get( path: "jmx/@rootName",  def: null));
287
288     } else {
289       jmxConfig = new JmxConfiguration( enabled: false,  agentId: n
290     }
```

上图中看见了 poc 中指定的三个参数，并且用他们创建了 JmxConfiguration 对象赋值给 jmxConfig
这里还没完，JmxConfiguration 仅仅是存储信息的，并没有执行任何操作，跟踪 jmxConfig 参数如下

```
jmxConfig;
```

Usages of jmxConfig in All Places (9

```
SolrConfig.java   283  jmxConfig = new JmxConfiguration(true,
SolrConfig.java   289  jmxConfig = new JmxConfiguration(false, null, null, null);
SolrConfig.java   861  if (jmxConfig != null) result.put("jmx", jmxConfig);
SolrConfig.java   887  if (jmxConfig != null) result.put("jmx", jmxConfig);
SolrCore.java    1169  if (config.jmxConfig.enabled) {
SolrCore.java    1170  return new JmxMonitoredMap<String, SolrInfoMBean>(na
```

跟入红框所示位置

```
private Map<String,SolrInfoMBean> initInfoRegistry(String name, SolrConfig config) {
  if (config.jmxConfig.enabled) {
    return new JmxMonitoredMap<String, SolrInfoMBean>(name, String.valueOf(this.hashCode()), config.jmxConfig);
  } else {
    log.debug("JMX monitoring not detected for core: " + name);
    return new ConcurrentHashMap<>();
```

JmxMonitoredMap 的构造函数中就进行了 JMX 监控操作，可以触发 rmi 序列化

但是这里属于 initInfoRegistry 函数中，还不清楚这个函数在哪儿调用，反向查找在 SolrCore 的构造函数中被调用到

```
896     // Initialize JMX
897     this.infoRegistry = initInfoRegistry(name, config);
```

SolrCore 的构造函数由 CoreContainer#reload 调用的 SolrCore#reload 触发

至此服务端触发jmx流程已经完整

链接

漏洞详情：https://issues.apache.org/jira/browse/SOLR-13301
漏洞POC：https://github.com/mpgn/CVE-2019-0192
https://issues.apache.org/jira/secure/attachment/12961503/12961503_SOLR-13301.patch
https://lucene.apache.org/solr/guide/6_6/config-api.html#ConfigAPI-CommandsforCommonProperties

上一篇：从5道堆题看堆利用 下一篇：RCE——从一个错别字到获取域管理...

1. 8 条回复

0c**** 2019-03-13 10:16:57

JmxMonitoredMap 的构造函数中就进行了 JMX 监控操作，可以触发 rmi 序列化

具体是怎么触发的rmi序列化的 ？ 看漏洞说明中有提调用了bind方法,但是我没找到。

0 回复Ta


r00t4dm 2019-03-13 10:33:33

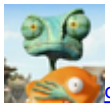
为啥JMX可以触发RMI反序列化？

0 回复Ta


王天 2019-03-13 14:08:29

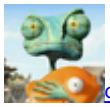
这个漏洞利用起来会鸡肋吗？影响范围广吗？

0 回复Ta


orich1 2019-03-13 14:50:21

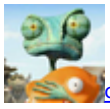
@0c**** 看 JmxMonitoredMap 里的 else 代码块，调用到了 JMXConnectorServer#start ，运行时实则转向 javax.management.remote.rmi.RMIConnectorServer#start，在判断 address 的时候（也就是指定的恶意serviceUrl），如果 jndi 字符串在协议开头，就调用 javax.management.remote.rmi.RMIConnectorServer#bind ，意味着可以进行远程 server 绑定操作，这个底层走的 rmi 交互

1 回复Ta


orich1 2019-03-13 14:52:41

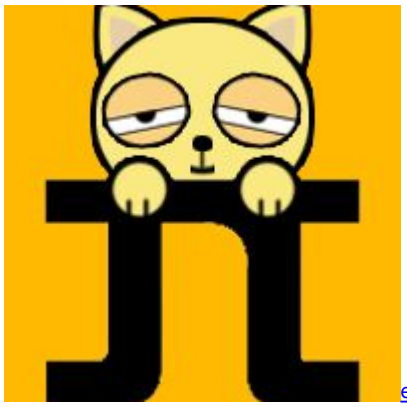
@r00t4dm 因为 JMX 提供的连接或者创建方式之一就是使用的 rmi

1 回复Ta


orich1 2019-03-13 14:56:25

@王天 简单来说会出现流量外连恶意服务器的情况，并且成功rce也需要 jdk 和各种存在漏洞的依赖包，影响范围未调查

1 回复Ta


elliot 2019-03-13 16:30:13

jdk-7u21
solr:5.5.1
./solr -e techproducts -Dcom.sun.management.jmxremote
复现进行POST请求时，出现Timeout 该怎解决，求师傅解答

0 回复Ta


47235****@qq.com 2019-03-24 21:26:39

@elliot 我用的6.0.0.可以的！但是shell反弹不出来！

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板