**Weblogic RCE by only one GET request — CVE-2020–14882 Analysis**

TL;DR



It's been 1 year, this year I was struggling with GadgetInspector, figuring out CVE-2020–2555 and reporting to ZDI.

*If you want to read it, you can always pull it down*

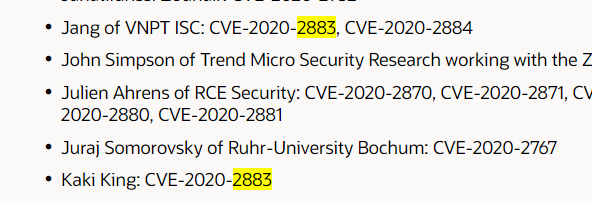
That day, every time Oracle CPU release is to wait, see if my bug has been patched, what level is public PoC ... Finally, after 6 months, Oracle also released a patch for the bug it reported.

Back then it was also a bit naïve, I don't think finding 1 new deserialization chain also counted as a new bug, and numbered a decent 9.8/10...

After CVE-2020–2883 and 2884 (bypass of 2555), I got bored, no longer wanted to pursue gadgetchain search, and repeated the same 1 T3 entrypoint on weblogic.

During that time, it was also in the planning season, so I handed over all this work to my colleagues, who many people probably know =)).

After CVE-2020–2555, many people also realized that library coherence had many cool things to take advantage of. From then on, each time Oracle CPU releases, it adds 1 new gadgetchain of weblogic T3 deserialzation, sometimes there are also CVE's that coincide with 1 someone else' number:



Actually, finding gadgetchain is also very interesting, but only interesting when you are still in the entrypoint.

The prevention is very simple, just disable T3 / IIOP to rest the ̄\\_(ツ)\_/ ̄ game .

.

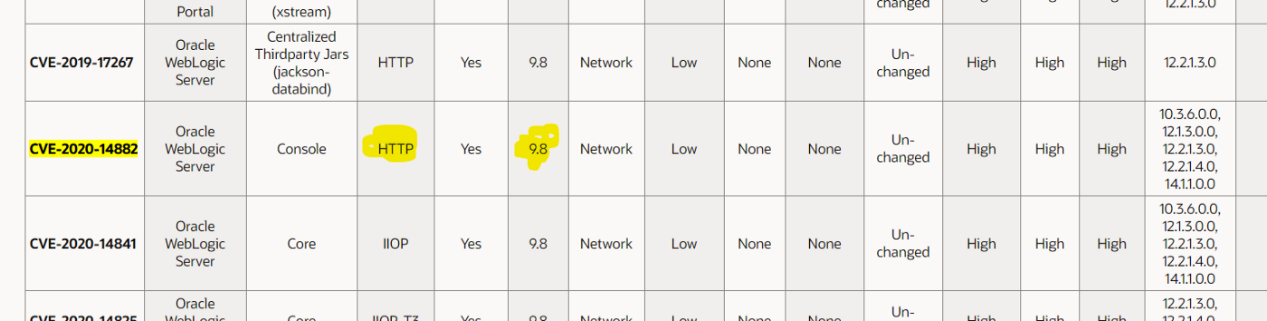
.

.

Until October 21, Oracle CPU Oct brought 1 few big surprises.

Liếc qua weblogic thì cũng vẫn như thường lệ: T3, IIOP… bình thường,

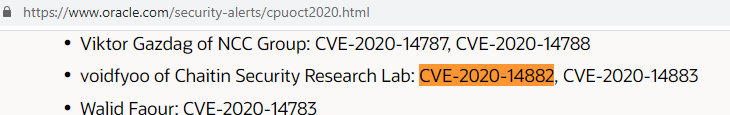
Tuy nhiên có 1 điểm mới ở đây:



**CVE-2020–14882**:

* CVSS: 9.8/10
* Giao thức: **HTTP**
* Ảnh hưởng tất cả các version

Bug này được report bởi 1 thanh niên người tàu:

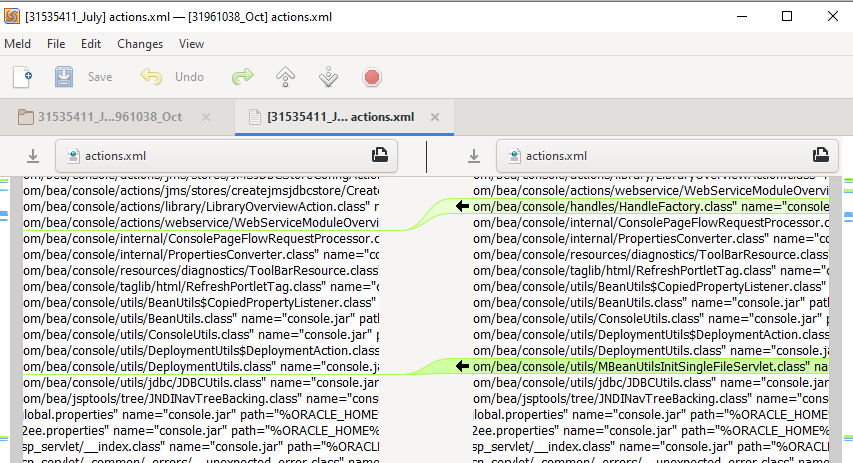


Trước giờ weblogic hầu như chỉ có bug ở chỗ T3 Deser, cứ vá rồi lại bypass, vá rồi lại bypass … (nghĩ đến cũng đủ mệt ròi).

Chậc, lại phải kéo patch về xem thôi …

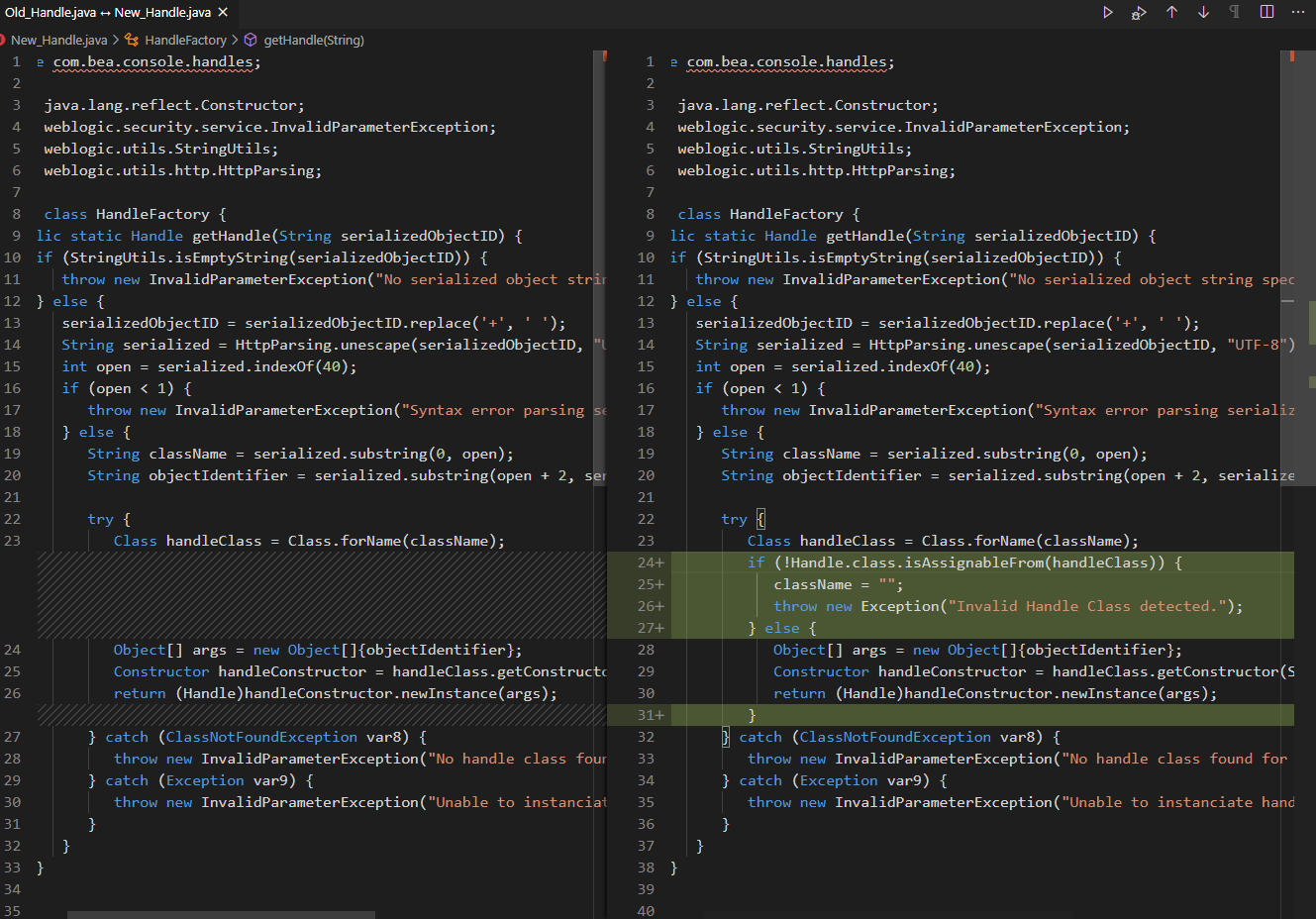
Mình chọn phiên bản để phân tích là Weblogic 12.2.1.3, bản vá Oct đợt này có số 3191038, của July là 31535411.

Diff 2 bản vá và phát hiện ra tại component console, 1 số file class đã bị thay đổi trong bản vá mới: **HandleFactory.class** và **MBeanUtilsInitSingleFileServlet.class**

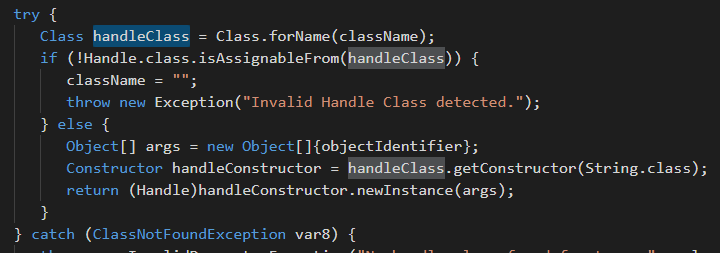


**#The Sink**

Tiếp tục xem sự thay đổi trong các file này, đầu tiên là **com.bea.console.handles.HandleFactory.getHandle():**

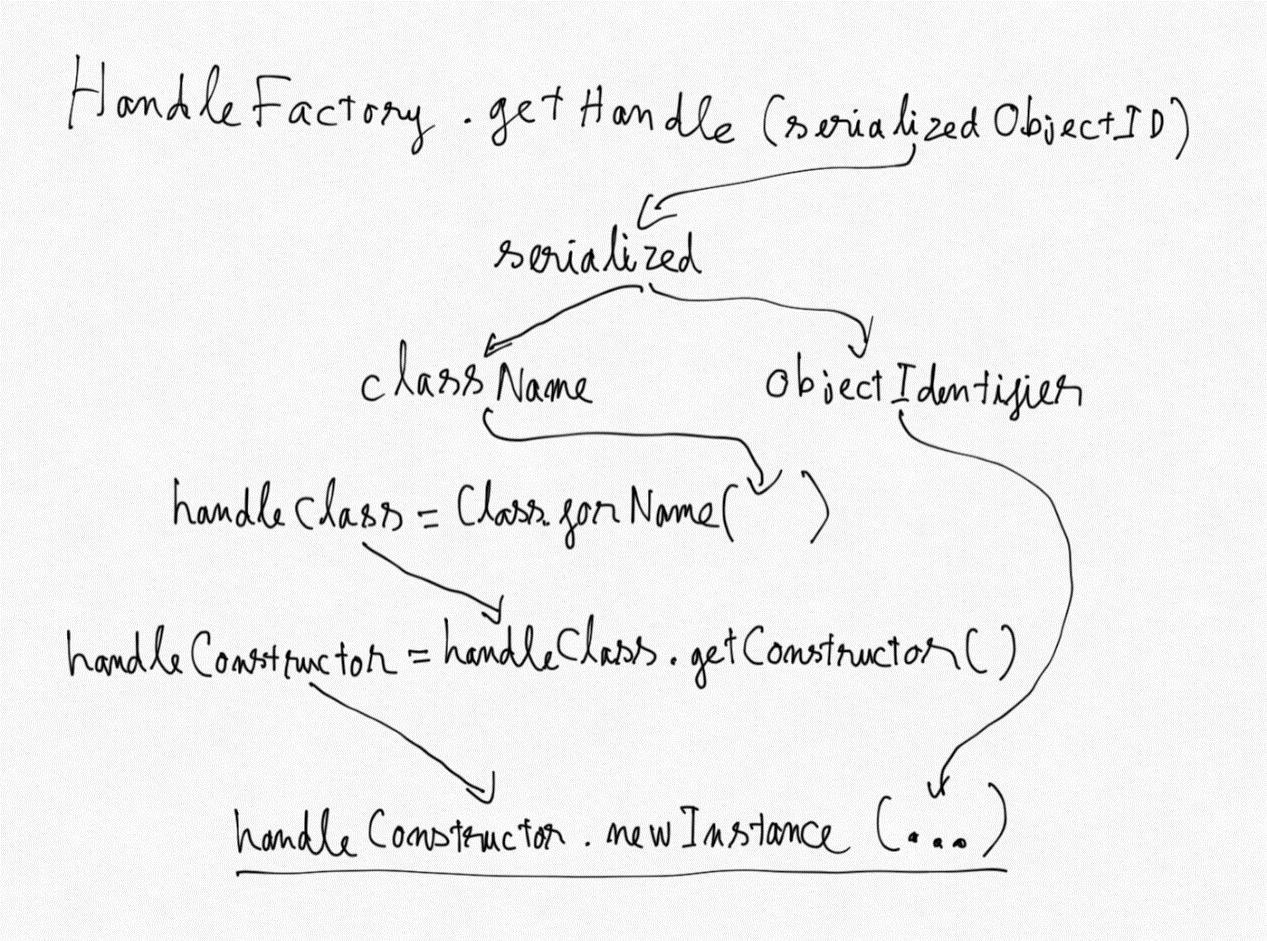


Nhìn sơ qua cũng có thể nhận thấy ngay điểm khác biệt này: bản vá lần này của **HandleFactory**đã kiểm tra lại kiểu của **handleClass**, chỉ cho phép các class là instance của **com.bea.console.handles.Handle** đi qua:



Theo như flow của method **HandleFactory.getHandle()**, dữ liệu nhận vào là 1 String, sau khi qua nhiều lần xử lý thì sẽ đến được **Class.forName(className)** để load class này lên. Tiếp đó sẽ tìm kiếm 1 trong class đó 1 constructor với 1 arg là String để tạo instance mới từ đó.

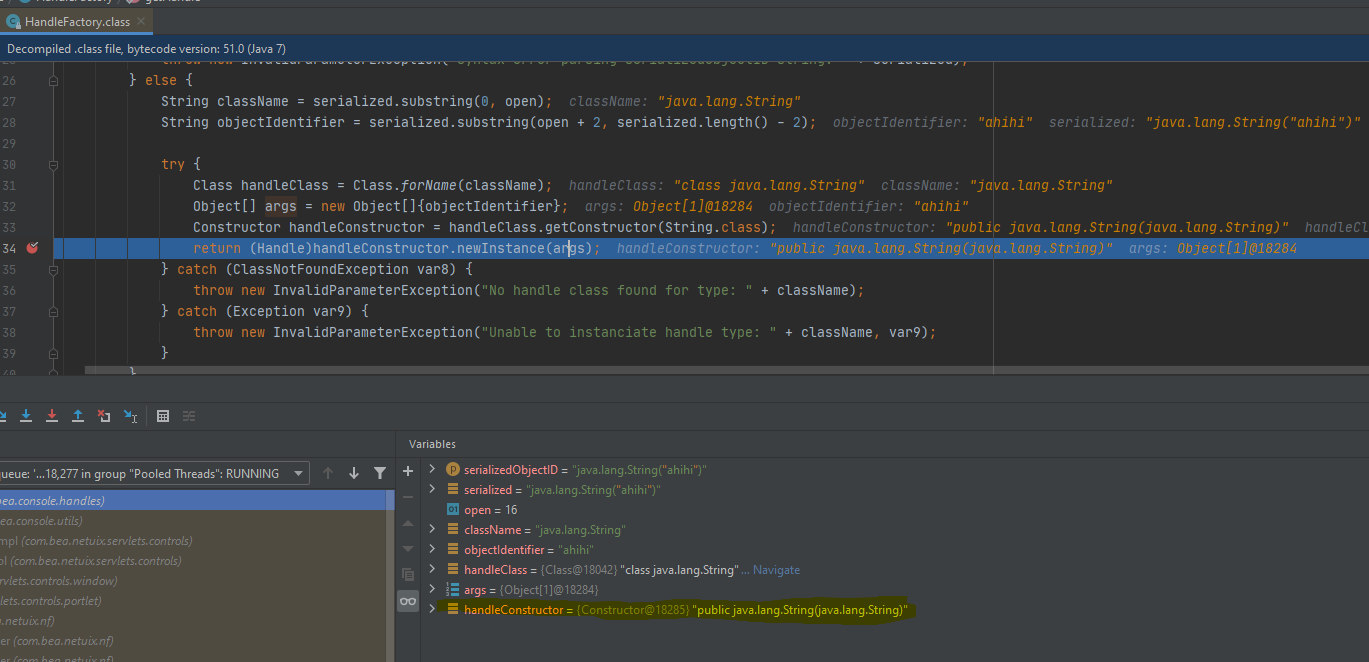
Diễn giải luồng xử lý sẽ như sau:



Có thể trigger entrypoint này bằng 1 GET request sau (tuy nhiên đoạn này vẫn cần authenticated vào trước):

http://<target>/console/console.portal?\_nfpb=true&\_pageLabel=HomePage1&handle=**java.lang.String("ahihi")**

Đặt breakpoint tại HandleFactory.getHandle(), các giá trị của **handleConstructor**, **args**đều đã bị control từ dữ liệu đầu vào.



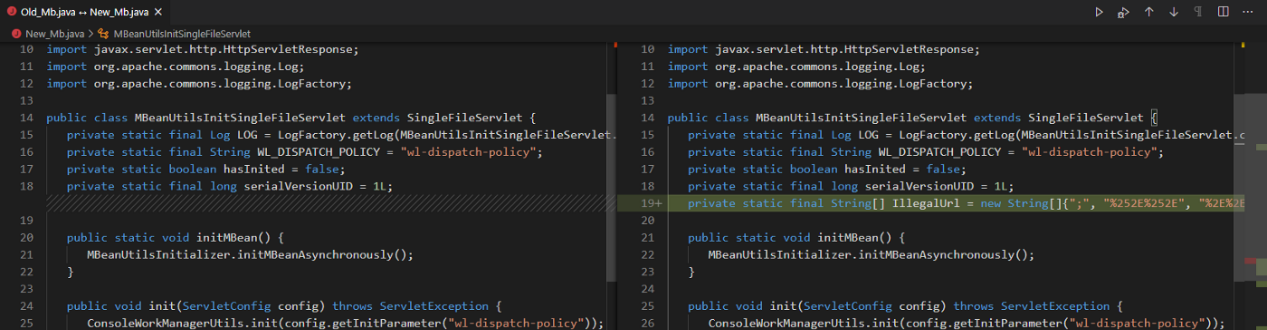
Đây cũng chính là vị trí bị lợi dụng để RCE trong CVE-2020–14882,

Để lợi dụng được entrypoint này thì cần phải tìm ra 1 class có chứa public Constructor với 1 argument = String.

Khi debug tới đây thì mình cũng định để đó, dù sao sink cũng biết rồi, tới khi nào tìm được entrypoint pre-auth thì mới chạy tool để tìm class trigger được bug!

**#The Source**

Với HandleFactory.getHandle() đã là sink của bug này, như vậy thì class bị patch còn lại — MBeanUtilsInitSingleFileServlet có thể sẽ là Source của bug!



Class sau khi được vá sẽ có thêm 1 field “IllegalUrl” như sau:

private static final String[] **IllegalUrl** = new String[]{";", "**%252E%252E**", "**%2E%2E**", "**..**", "%3C", "%3E", "<", ">"};

Đoạn code xử lý của servlet này đơn thuần chỉ là check xem trên url có tồn tại chuỗi “**..**” hay không, nếu có thì sẽ reject request này! Chắc hẳn phải có vấn đề gì đó với dấu “**..**”, nên nó mới bị filter lại như vậy.

Hơn nữa để reach được tới MBeanUtilsInitSingleFileServlet.service(), cần phải qua 1 bước authen nữa ¯\\_(ツ)\_/¯.

“*Thế là post-auth RCE ròi, làm sao có thể là 9.8/10 được chứ …*”. Toi thầm nghĩ và tiếp tục mò mẫm.

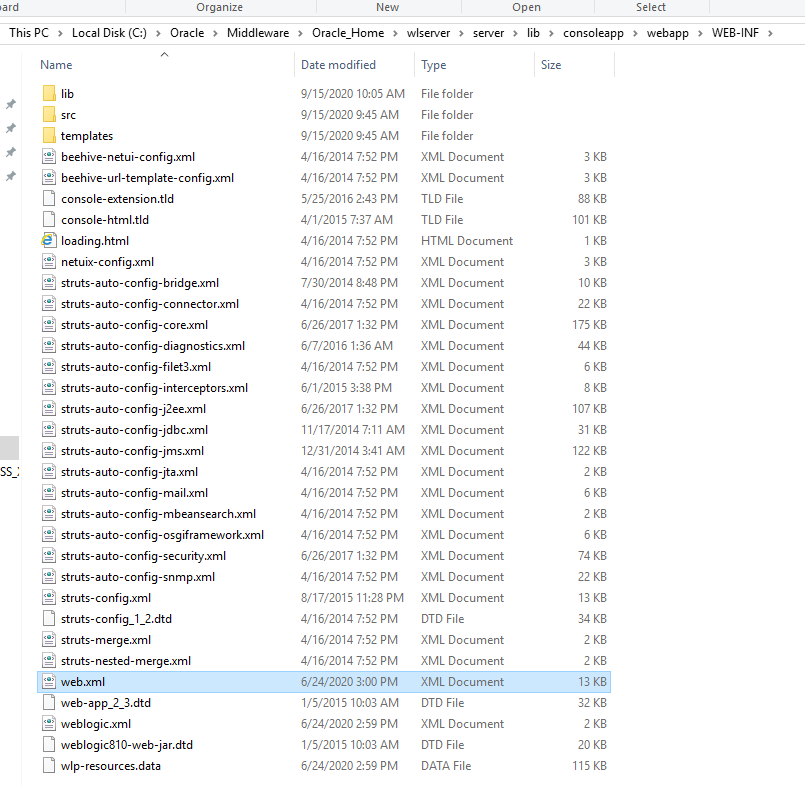
.

.

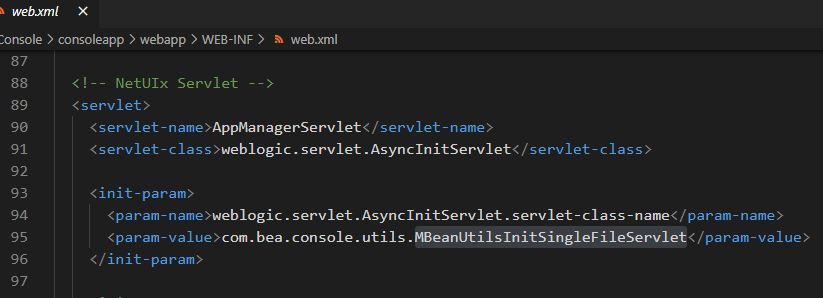
Có lẽ cũng giống như 1 số servlet trên java, weblogic cũng có folder webapp, đối với của console thì đó là:

“*\Middleware\Oracle\_Home\wlserver\server\lib\consoleapp\webapp*”

Và đương nhiên, cũng có các file config web.xml, struts-config.xml như thường lệ:

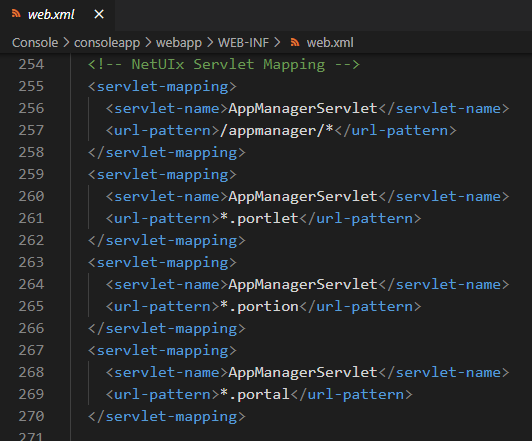


Theo như config trong web.xml, *MBeanUtilsInitSingleFileServlet*là 1 init-param của AppManagerServlet,

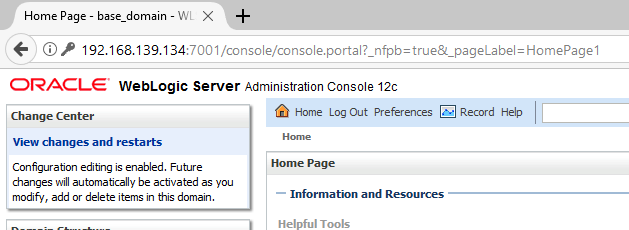


AppManagerServlet được map vào các url pattern sau:

* /appmanager/\*
* \*.portlet
* \*.portion
* \*.portal

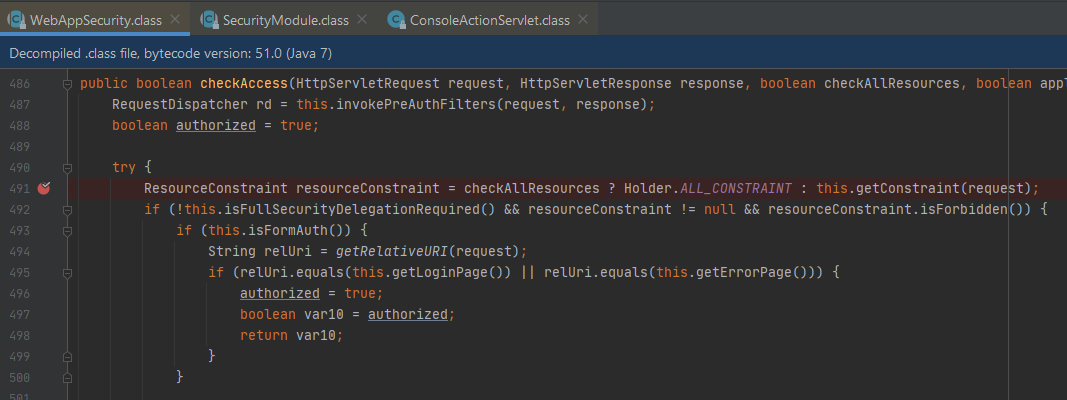


Trong đó có bao gồm cả “*/console/console.portal*”, có thể thấy ngay sau khi login vào console:



…

Sau khi đào bới một hồi thì cũng tìm ra cái class xử lý logic, quyết định xem request này nên authen or non-authen required, trong weblogic 12.2.1.3 là class ***weblogic.servlet.security.internal.WebAppSecurity.checkAccess()***:



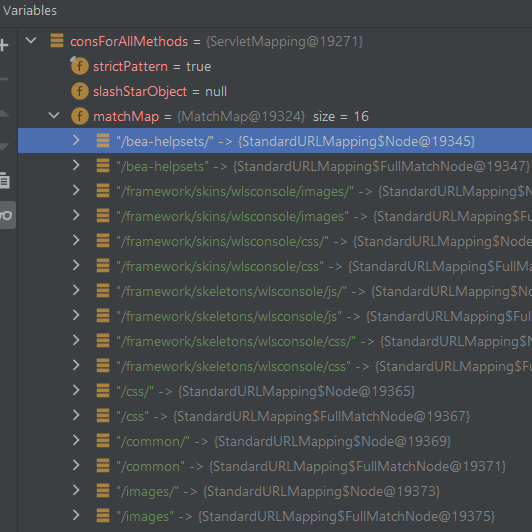
Tại đây,***WebAppSecurityWLS.getConstraint(request)*** sẽ lấy các constraint của request hiện tại, mỗi một constraint có chứa các thông tin như sau:



Trong đó, nếu như constraint có flag *unrestrict=true* thì request đó sẽ được quyết định là unauth, nếu không sẽ trả về page login.

Tiếp tục F7 đi vào ***WebAppSecurityWLS.getConstraint()***, tại đây có thể lấy được tất cả các constraint, dựa vào đó có thể biế t được các url nào sẽ được bỏ qua phần check authentication, list các url pattern được bỏ qua check authen như sau:

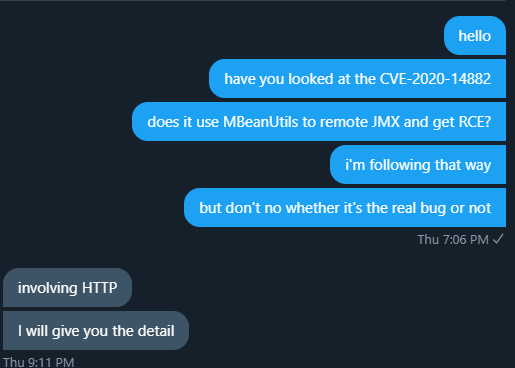
* */bea-helpsets/\**
* */framework/skins/wlsconsole/images/\**
* */framework/skins/wlsconsole/css/\**
* */framework/skeletons/wlsconsole/js/\**
* */framework/skeletons/wlsconsole/css/\**
* */css/\**
* */common/\**
* */images/\**



Dựa vào list các pattern này và thử thêm các trick bypass url trước đó, ví dụ như: “*..;/, /#/../*”, nhưng vẫn ko tìm đc gì thêm hay ho…

Tới đây thì mình stuck luôn, ko nghĩ ra gì mới!

Lên twitter thì chưa thấy ai đăng PoC gì cả, đành vứt hết liêm sỉ đi xin hint từ những người anh em bên mẫu quốc! (xin phép được giấu tên người này)

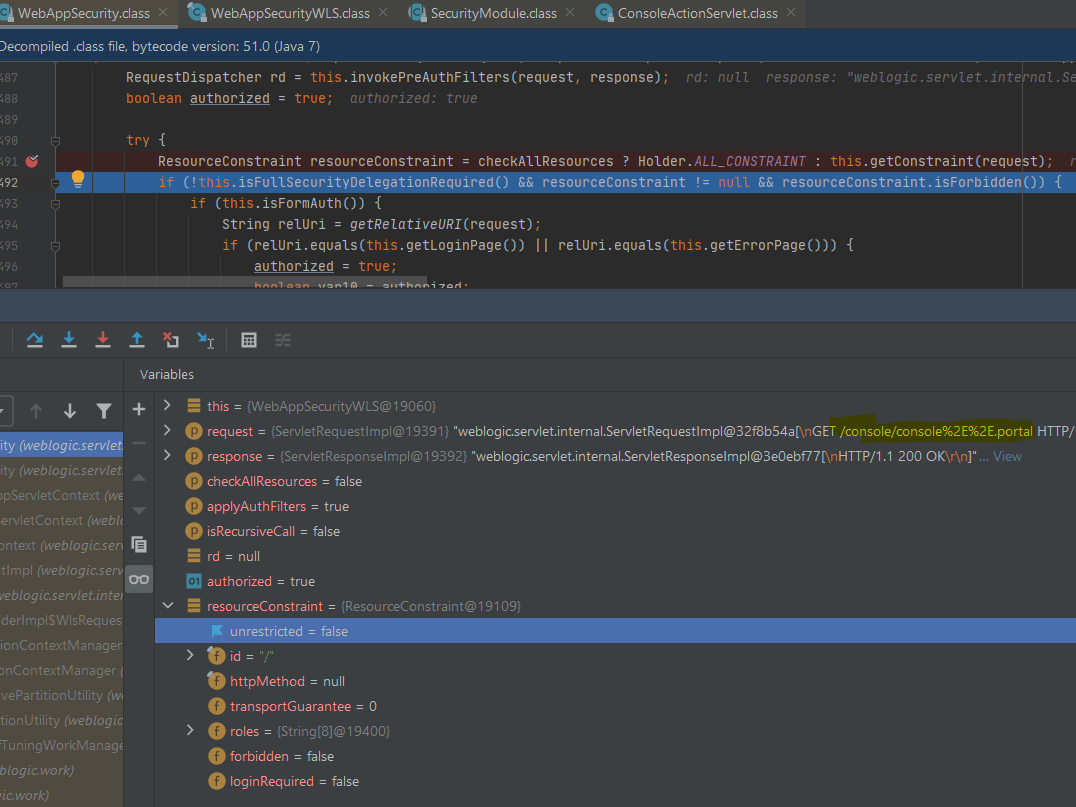


Yeah, và sau đó bác này cho mình hẳn 1 cái pic của PoC luôn, thật là tốt bụng và hào phóng!

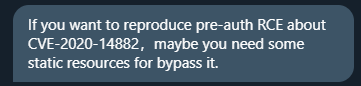


Trong đó thì phần abuse HandleFactory đã biết rồi, còn phần url kia lại là: “*/console/console%2E%2E.portal*” A.K.A “*/console/console…portal*”

Mình đem thử thì thấy ko đúng lắm, với url “*/console/console%2E%2E.portal*” thì constraint trả về vẫn là “**unrestricted=false**” nghĩa là vẫn cần authen mới qua được:



Hỏi lại thanh niên kia thì hắn mới bảo như vậy:



PoC hắn gửi ban đầu chỉ là cái hint mà thôi, để bypass thì cần 1 chút trick, abuse 1 vài resource nào đó mới có thể trở thành unauth RCE được!

Loanh quanh luẩn quẩn lại quay trở về với vị trí ban đầu,

Khi đó thì mình cũng khá là nản rồi, đành kêu thêm mấy đàn em vào ngâm cứu chung cho đỡ. Lại 1 chút flashback về ngày còn chơi CTF, ngày đó cứ mỗi khi stuck lại đem bàn giao idea cho đồng đội để san sẻ suy nghĩ, và hiệu quả lúc nào cũng cao hơn là 1 mình tự chơi …

Nhóm này thì cũng chỉ có 3 người: mình,

[PeterJson](https://medium.com/u/9d0bc610db35?source=post_page-----6e4b09981dbf--------------------------------)

 và @Quynh Le. Chuyên đâm chọt các loại bug xảy ra trên java



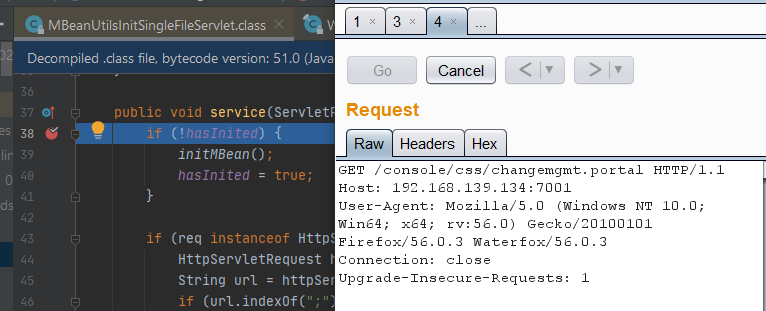
Và rồi thanh niên Đức đã tìm ra thứ cần phải thấy =)), 1 url có thể trigger được ***MBeanUtilsInitSingleFileServlet***mà không phải authen gì cả:

/console/css/changemgmt.portal

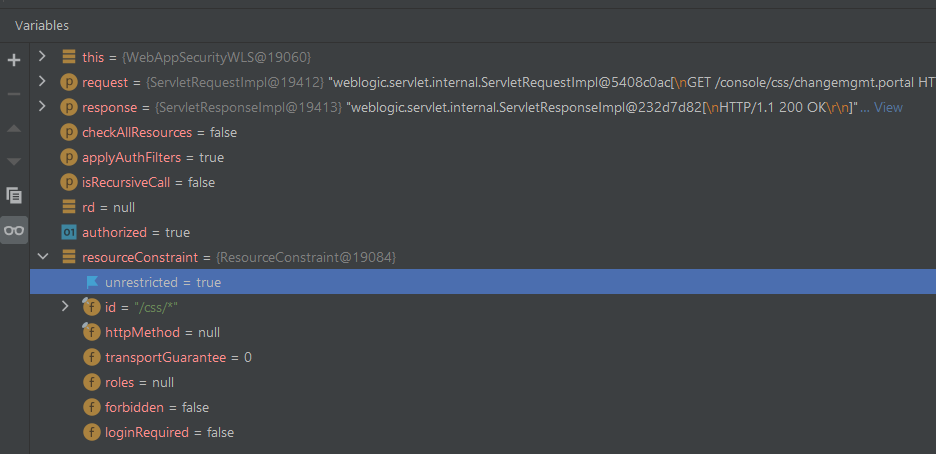
图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

Mình cũng hơi bất ngờ khi thấy request này, nhưng test lại thì đúng là nó có thể trigger được thật …



Kiểm tra lại trong debugger, constraint đúng là được map vào “/css/” nên được **unrestrict**thật,

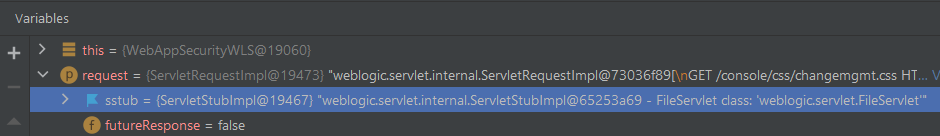


Và bất ngờ hơn, khi kiểm tra lại servlet handle cái entrypoint này, lại thấy là ***AsyncInitServlet***đang handle, chứ không phải là ***FileServlet***như mình nghĩ!

图形用户界面, 网站

描述已自动生成

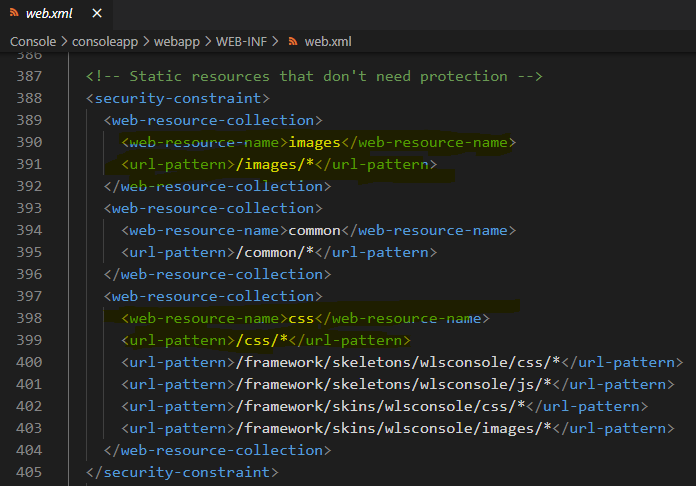
So sánh với 1 request thông thường (có đuôi .css), **FileServlet**sẽ handle request này.



Đó giờ mình cứ nghĩ là có điều gì magic ở FileServlet, nên cứ cắm đầu vào tìm lỗi của nó … :(

Nhưng thực tế điều magic lại không nằm ở đó … nó nằm ở web.xml cơ!

Nhìn lại phần khai báo static resource của web.xml:



Trong đó:

* Url pattern “*/common/\**” được xử lý bởi *JSPCServlet*(handle các request tới file jsp)
* Các url pattern “*/framework/\**” được xử lý bởi *FileServlet*
* …

Tuy nhiên với các url pattern*“****/images/\*****”* và *“****/css/\*****”* thì lại không được khai báo servlet nào sẽ handle cả!

Do đó khi request với url “***/console/css/aaasdasdasd.portal***”, ***AppManagerServlet***sẽ handle request này.

Tóm lược lại như sau:

* “*/css/\**” để bypass authen
* “*\*.portal*” để trigger

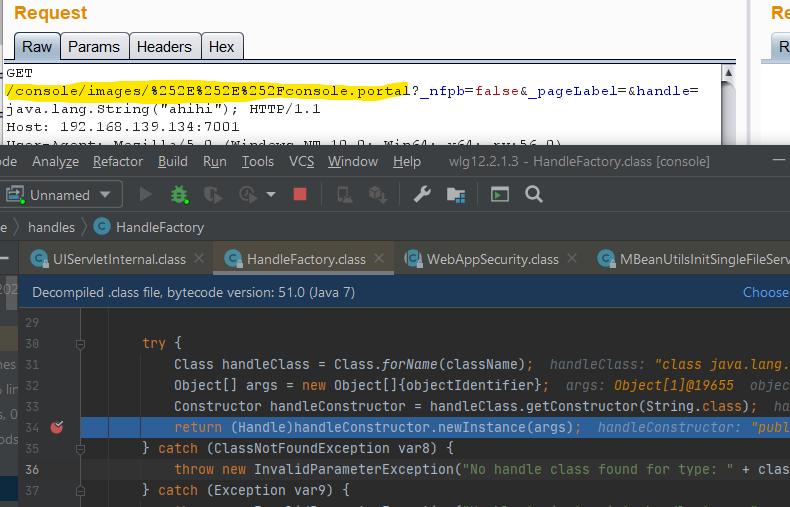
Tới đây thì đã có thể reach được ***MBeanUtilsInitSingleFileServlet.service()***mà không cần phải authen gì nữa

Tuy nhiên request vẫn chưa thể reach tới được ***HandleFactory.getHandle()*** do url pattern chưa match được với portlet!

Ngay lúc đó, dựa vào hint ban đầu mà thanh niên người tàu kia gởi:

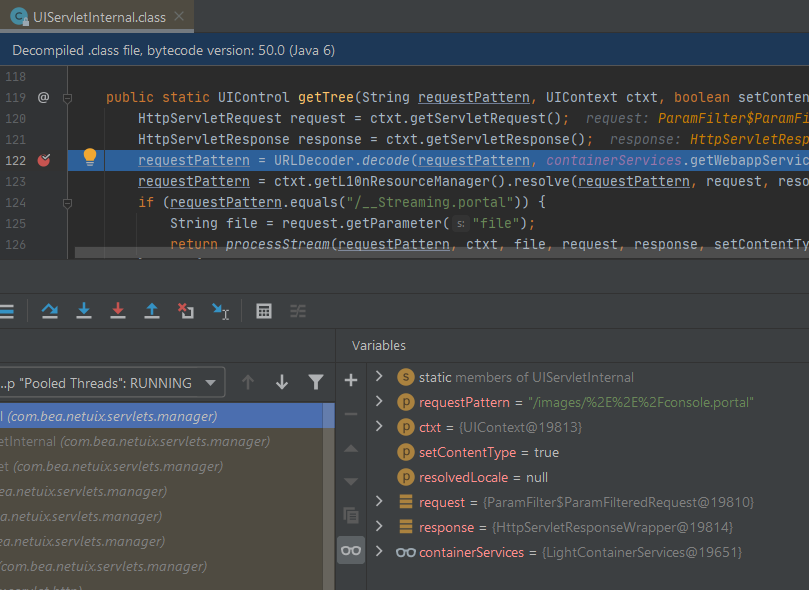


Vậy ra dấu “..” ở đây, là ý muốn hint sử dụng “..” double encode để trigger bug, thử lại thì đúng như thế thật:



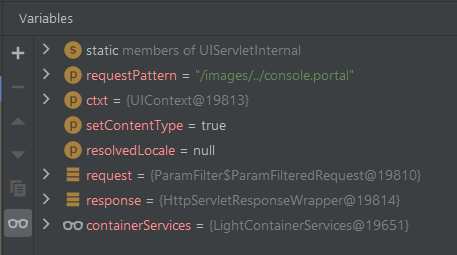
At this point, I can already write a complete PoC, but I would like permission to analyze this paragraph further,

After going through MBeanUtilsInitSingleFileServlet.service() and adding 1 more number of long paragraphs, at UIServletInternal.getTree(), the url pattern is extracted again and decode URL 1 again:



Before decode

And after decode:



This explains why the use of double encoded urls can both bypass the original normalized url processing paragraph, and still make the later servlet handle work properly!

**#Trigger RCE**

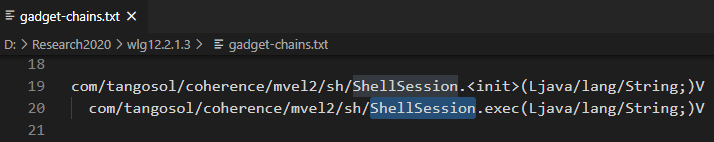
In the begging PoC, this young man uses ***com.bea.core.repackaged.springframework.context.support.FileSystemXmlApplicationContext()***to trigger RCE, however this chain requires an outbound.

I am a perfectionist, moreover in reality, quite a few servers block outbound, so I decided to find 1 new chain, no outbound required, RCE in one hit! 🤣

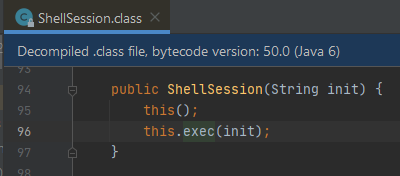
When I was a kid, I edited the GadgetInspector tool a lot, which served to trace the code.

By this time, it came into play again, pulled it out and slightly edited the conditions in the code to suit the new context, running about 5p, it managed to get quite a few results.

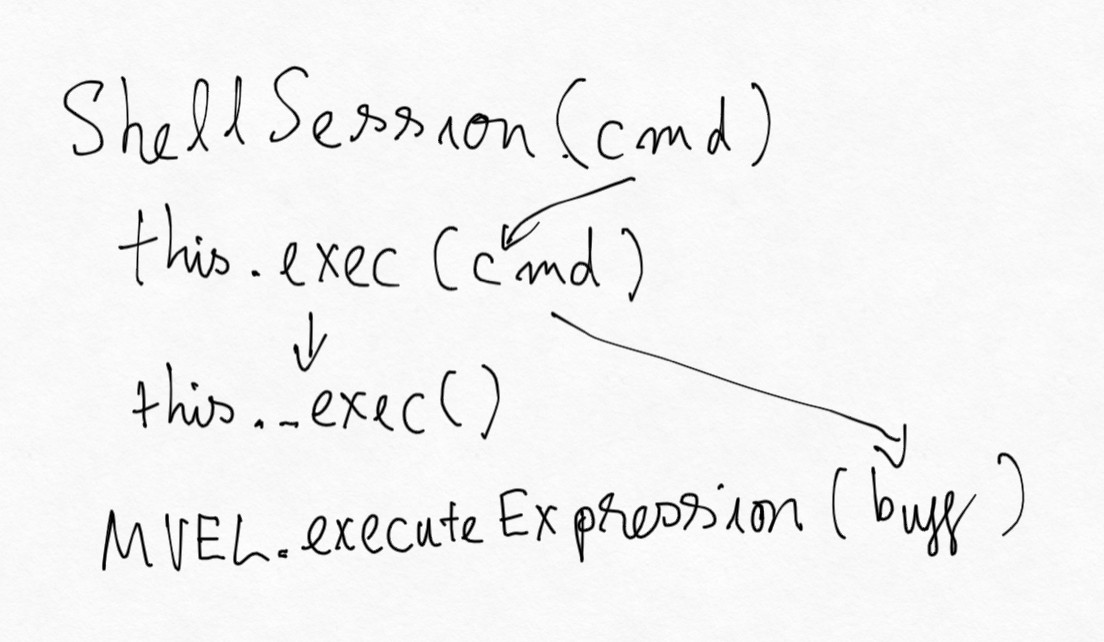
Of which there are 1 quite potential chains:



This chain after receiving the arg from the constructor will call *ShellSession.exec()* to execute with this arg always:



The chain behind *ShellSession.exec()*is still long to be executed, I summarize it as follows:



Thus, it is possible to execute the MVEL expression arbitrarily (regarding MVEL, it seems that I have also mentioned it in previous articles).

*In fact, until poc succeeds with this new chain, I have not been surprised, I do not believe that there are things available to take advantage of like that. I'm not sure if this is some 1 backdoor left in the code =))).*

After combining all of the above, adding a little salt sauce, I wrote the PoC execute command completely and then had a response =)) to be healthy. The last PoC was perfected by Germany, not himself,

PoC video: <https://youtu.be/JFVDOIL0YtA>

=)) Whoever needs PoC will do their own research in this, I don't have ready-to-eat food here.