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



Mới đó mà đã ngót nghét 1 năm, tầm này năm ngoái mình đang loay hoay với GadgetInspector, tìm ra CVE-2020–2555 và report cho ZDI.

*//Tản mạn ltinh xíu, ai muốn đọc thì kéo xuống dưới luôn cũng được*

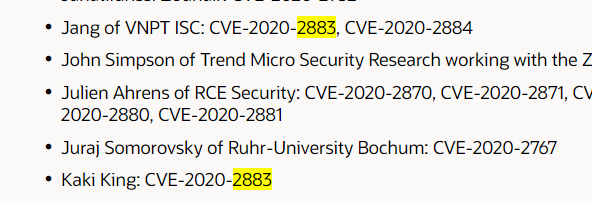
Ngày đó, mỗi lần Oracle CPU release là lại ngóng chờ, xem bug của mình đã được vá chưa, tầm nào thì được public PoC … Cuối cùng, sau 6 tháng, Oracle cũng đã release bản vá cho cái bug mình report.

Hồi đó cũng hơi ngây ngô, ko nghĩ việc tìm ra 1 cái chain deserialization mới cũng được tính là bug mới, và được đánh số 9.8/10 đàng hoàng luôn…

Sau cái CVE-2020–2883 và 2884 (bypass của 2555), thì mình đã chán, không còn muốn theo đuổi công việc tìm kiếm gadgetchain, và lặp lại chung 1 entrypoint T3 trên weblogic nữa.

Trong thời gian đó cũng là vào mùa đồ án nữa, nên là mình bàn giao lại hết công việc này cho đồng nghiệp của mình, là ai thì chắc nhiều người cũng biết =)).

Sau CVE-2020–2555, nhiều người cũng ngộ ra là library coherence có nhiều thứ hay ho để lợi dụng. Từ đó trở đi, mỗi lần Oracle CPU release là lại thêm 1 đống gadgetchain mới của weblogic T3 deserialzation, đôi khi còn có những CVE bị trùng số với 1 ai đó nữa cơ:



Thực ra thì việc tìm gadgetchain cũng rất chi là thú vị, nhưng chỉ thú vị khi còn entrypoint.

Việc ngăn chặn thì rất đơn giản, chỉ cần disable T3/IIOP đi là nghỉ game luôn ¯\\_(ツ)\_/¯.

.

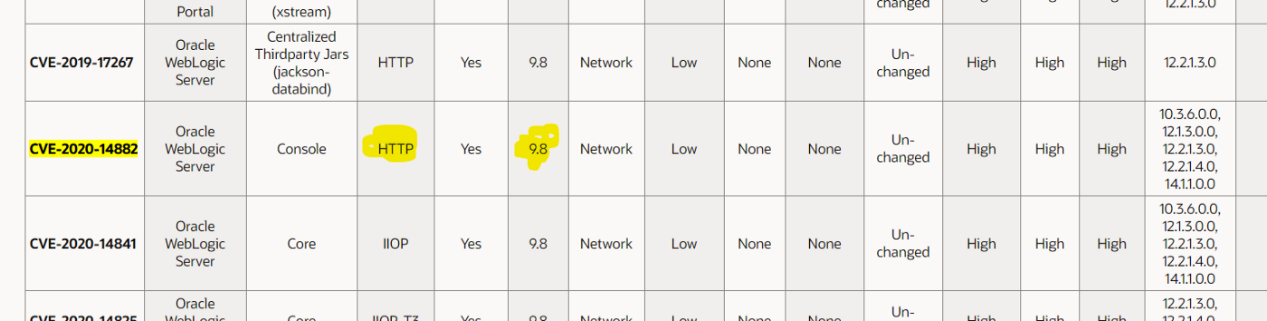
.

.

Cho tới ngày 21/10 vừa rồi, Oracle CPU Oct đã mang tới 1 vài bất ngờ lớn.

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

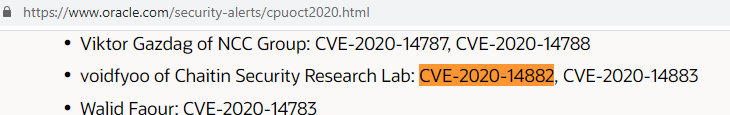
Tuy nhiên có 1 điểm mới ở đây:



**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:

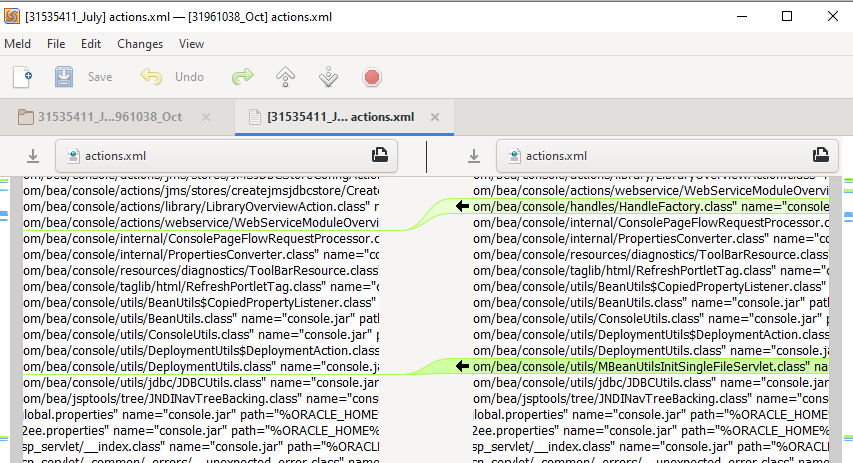


Trước giờ weblogic hầu như chỉ có bug ở chỗ T3 Deser, cứ vá rồi lại bypass, vá rồi lại bypass … (nghĩ đến cũng đủ mệt ròi).

Chậc, lại phải kéo patch về xem thôi …

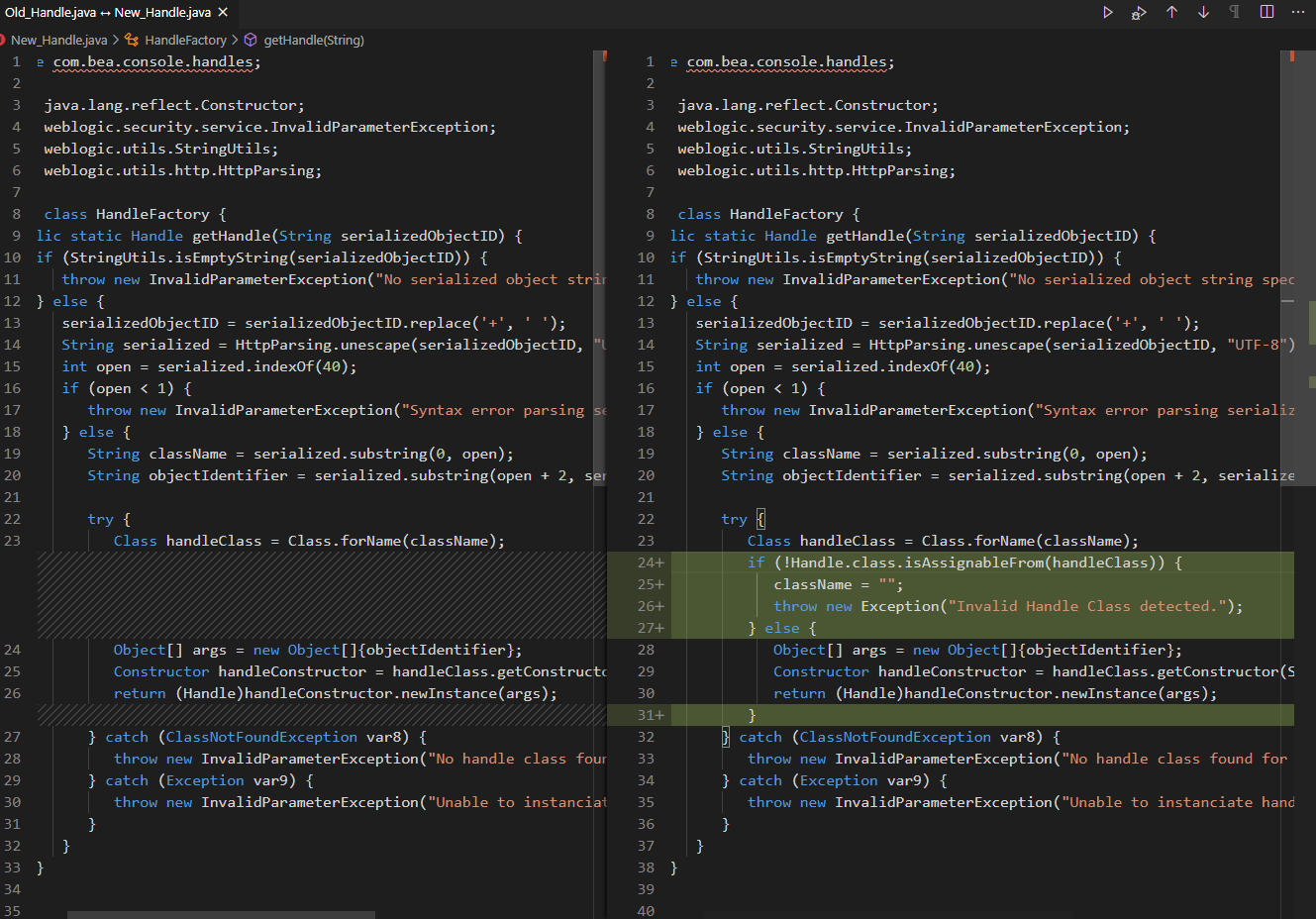
Mình chọn phiên bản để phân tích là Weblogic 12.2.1.3, bản vá Oct đợt này có số 3191038, của July là 31535411.

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**

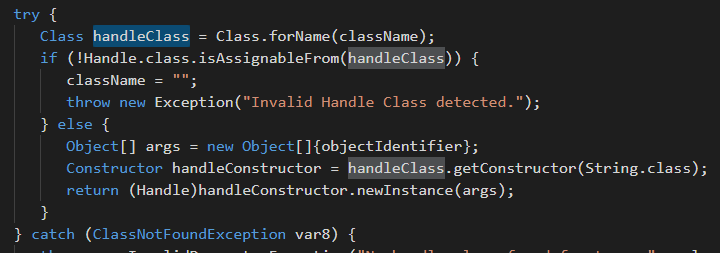


**#The Sink**

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

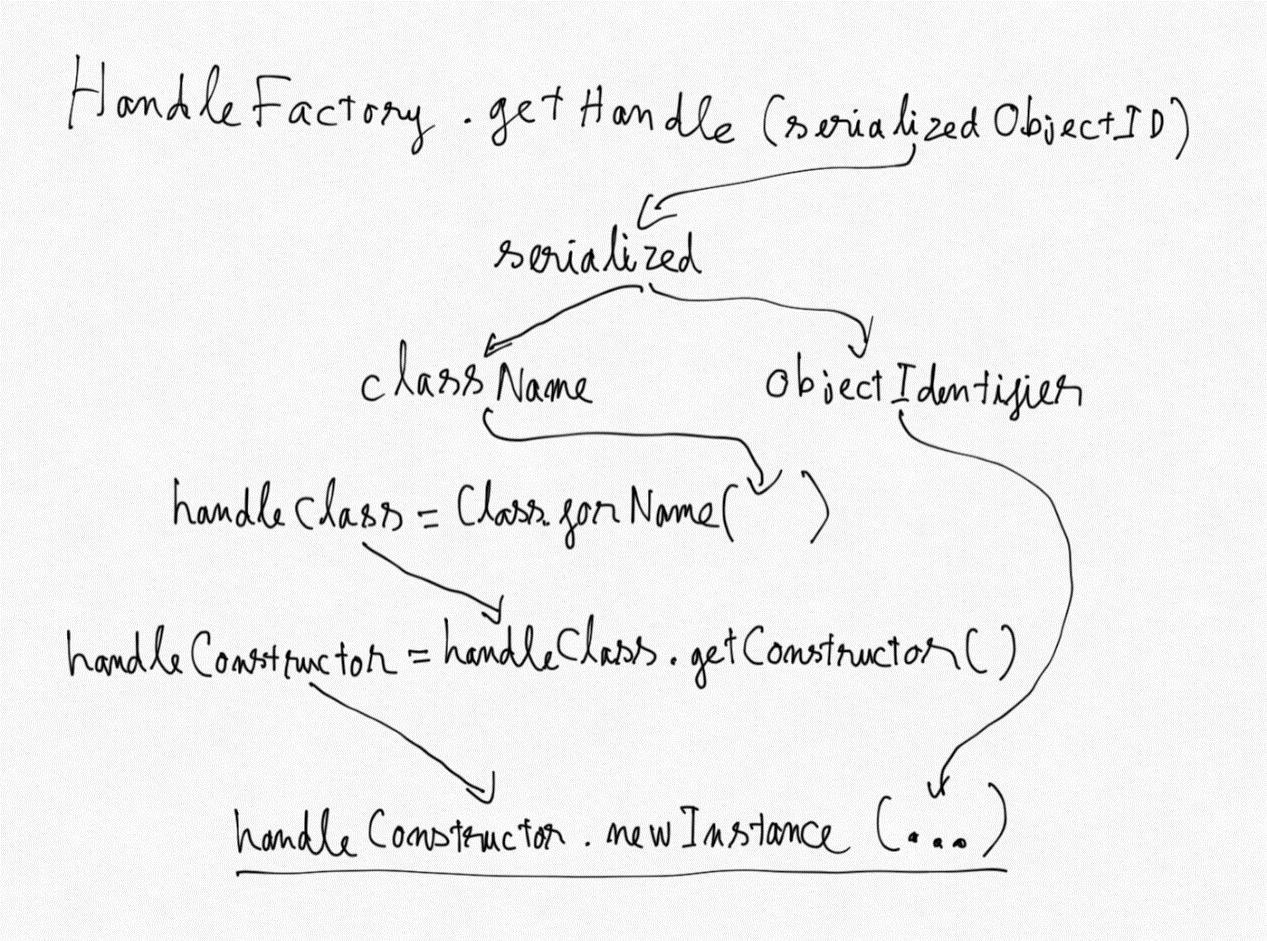


Nhìn sơ qua cũng có thể nhận thấy ngay điểm khác biệt này: bản vá lần này của **HandleFactory**đã kiểm tra lại kiểu của **handleClass**, chỉ cho phép các class là instance của **com.bea.console.handles.Handle** đi qua:



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.

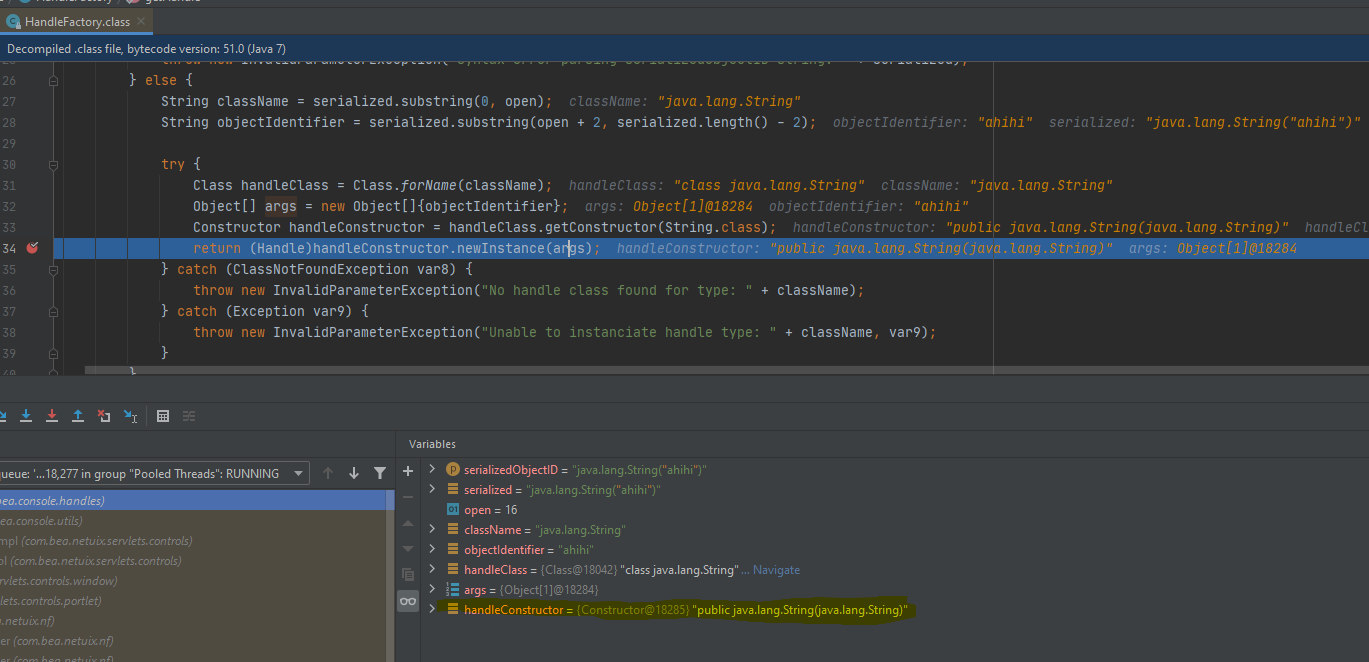
Diễn giải luồng xử lý sẽ như sau:



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.



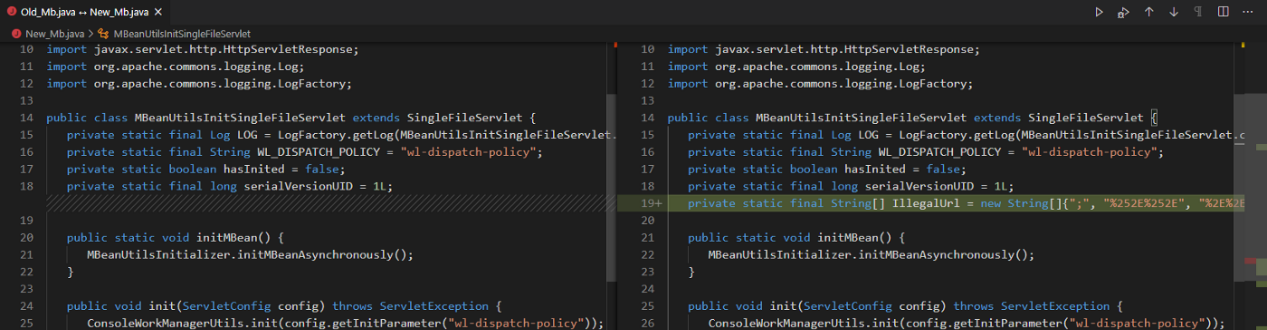
Đây cũng chính là vị trí bị lợi dụng để RCE trong CVE-2020–14882,

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.

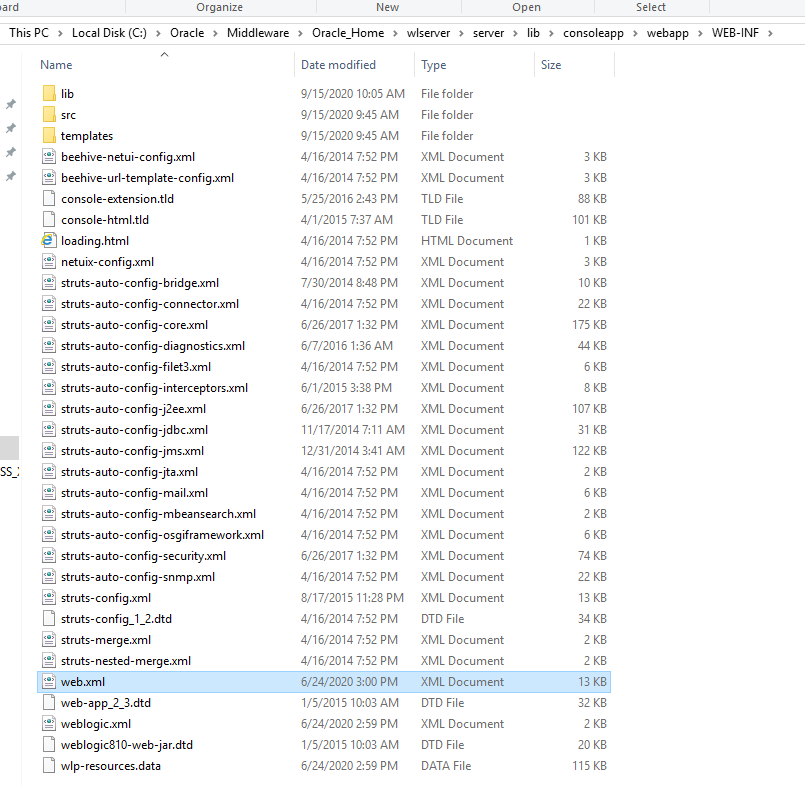
.

.

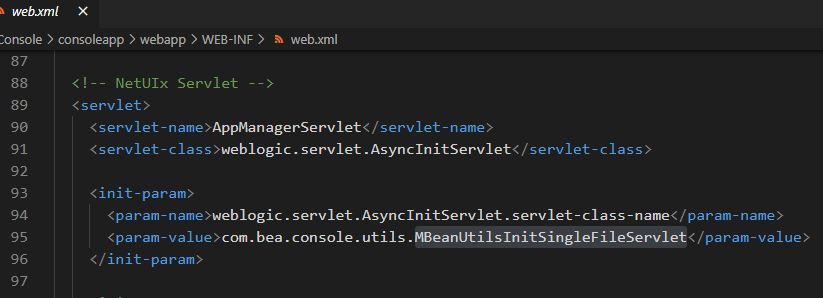
Có lẽ cũng giống như 1 số servlet trên java, weblogic cũng có folder webapp, đối với của console thì đó là:

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

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

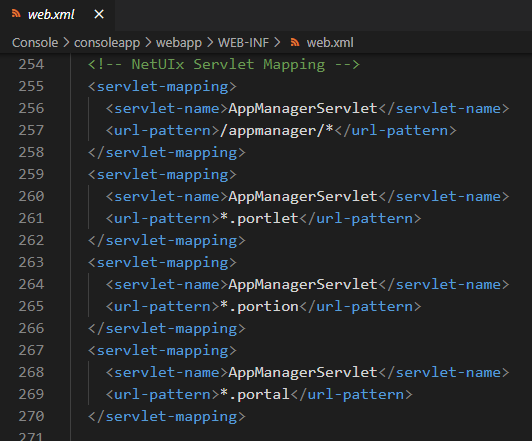


Theo như config trong web.xml, *MBeanUtilsInitSingleFileServlet*là 1 init-param của AppManagerServlet,

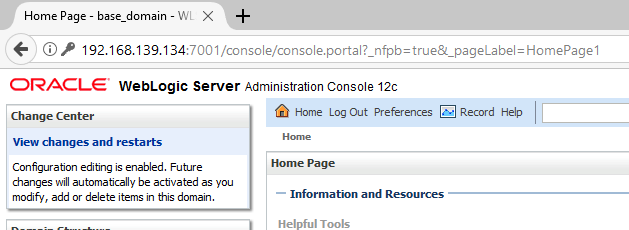


AppManagerServlet được map vào các url pattern sau:

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

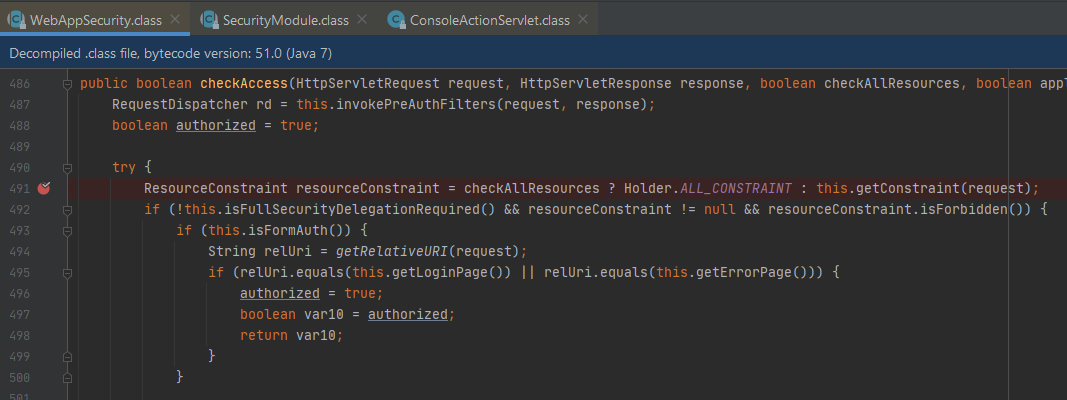


Trong đó có bao gồm cả “*/console/console.portal*”, có thể thấy ngay sau khi login vào console:



…

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()***:



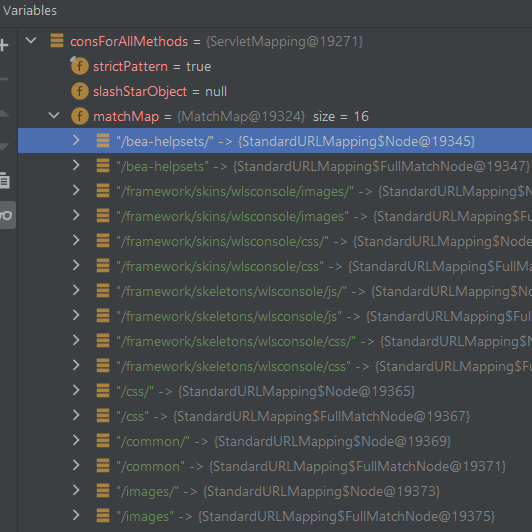
Tại đây,***WebAppSecurityWLS.getConstraint(request)*** sẽ lấy các constraint của request hiện tại, mỗi một constraint có chứa các thông tin như sau:



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:

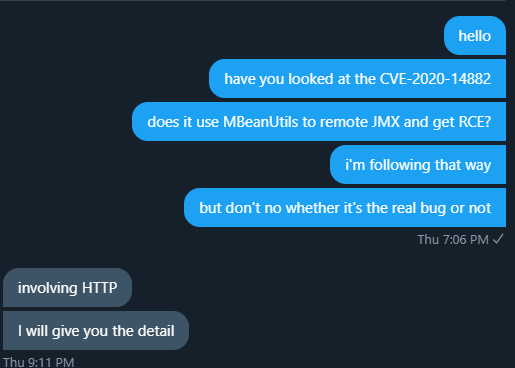
* */bea-helpsets/\**
* */framework/skins/wlsconsole/images/\**
* */framework/skins/wlsconsole/css/\**
* */framework/skeletons/wlsconsole/js/\**
* */framework/skeletons/wlsconsole/css/\**
* */css/\**
* */common/\**
* */images/\**



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)

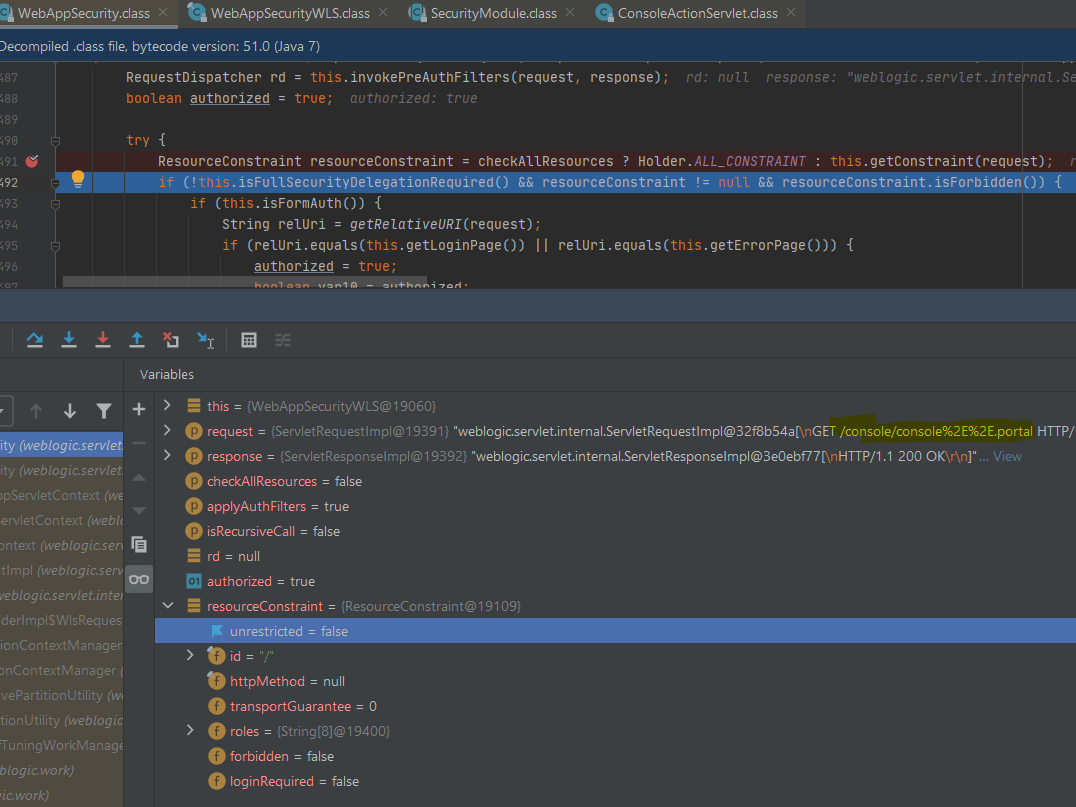


Yeah, và sau đó bác này cho mình hẳn 1 cái pic của PoC luôn, thật là tốt bụng và hào phóng!

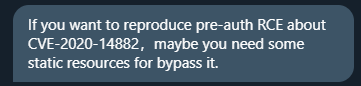


Trong đó thì phần abuse HandleFactory đã biết rồi, còn phần url kia lại là: “*/console/console%2E%2E.portal*” A.K.A “*/console/console…portal*”

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,

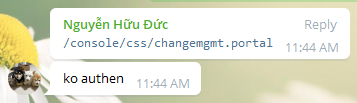
[PeterJson](https://medium.com/u/9d0bc610db35?source=post_page-----6e4b09981dbf--------------------------------)

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

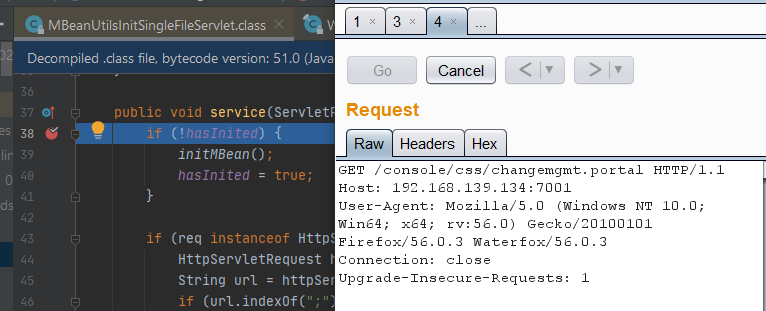


Và rồi thanh niên Đức đã tìm ra thứ cần phải thấy =)), 1 url có thể trigger được ***MBeanUtilsInitSingleFileServlet***mà không phải authen gì cả:

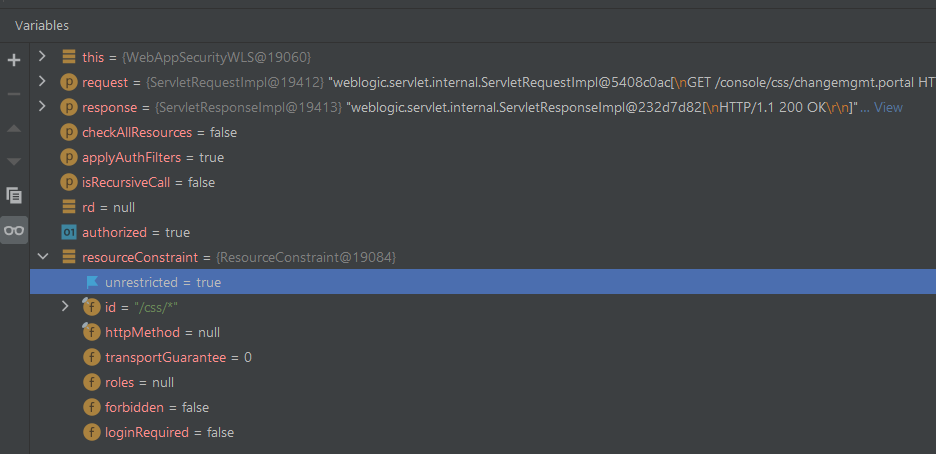
/console/css/changemgmt.portal



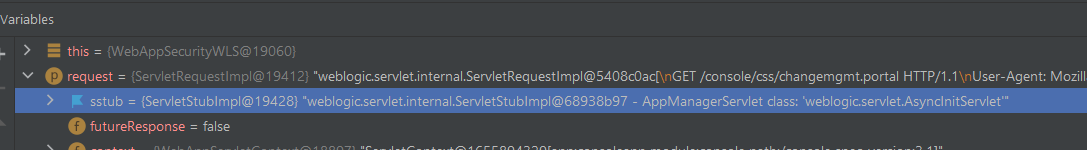
Mình cũng hơi bất ngờ khi thấy request này, nhưng test lại thì đúng là nó có thể trigger được thật …



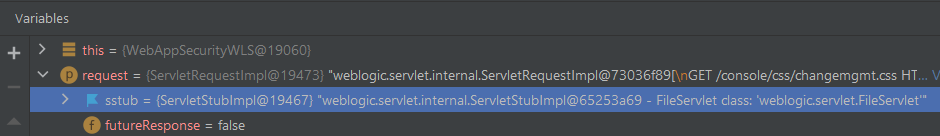
Kiểm tra lại trong debugger, constraint đúng là được map vào “/css/” nên được **unrestrict**thật,



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!



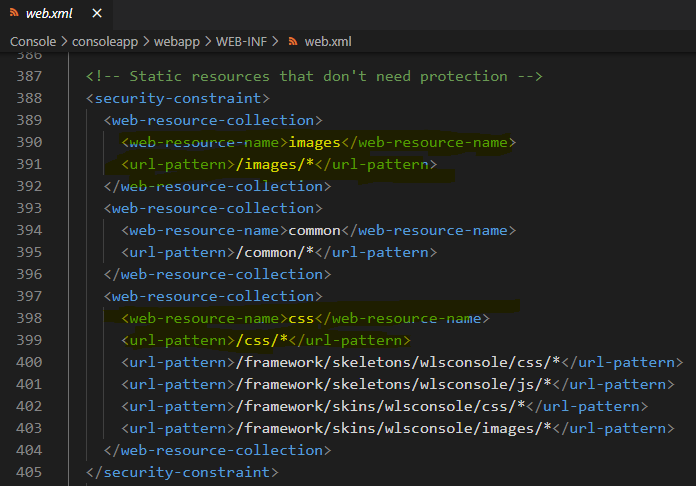
So sánh với 1 request thông thường (có đuôi .css), **FileServlet**sẽ handle request này.



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

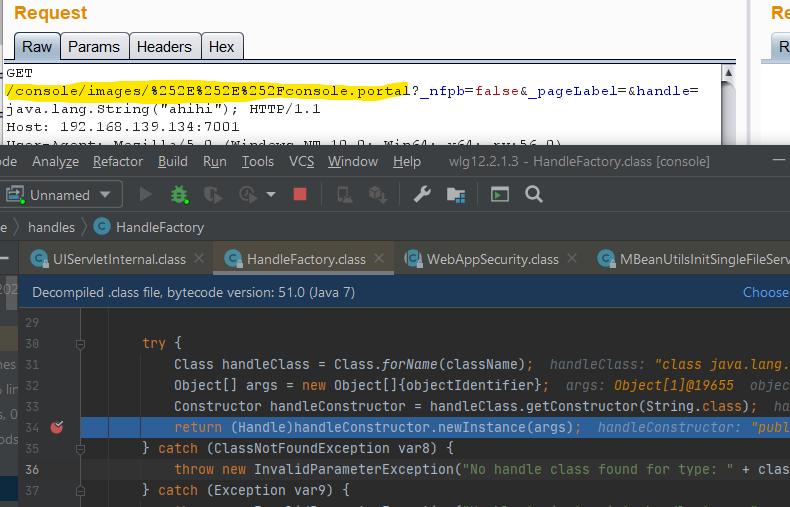
Tới đây thì đã có thể reach được ***MBeanUtilsInitSingleFileServlet.service()***mà không cần phải authen gì nữa

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:

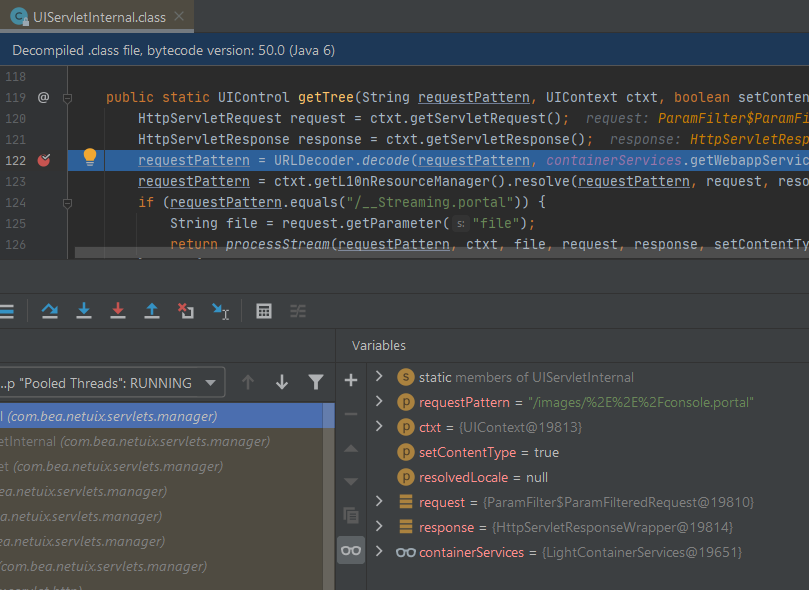


Vậy ra dấu “..” ở đây, là ý muốn hint sử dụng “..” double encode để trigger bug, thử lại thì đúng như thế thật:



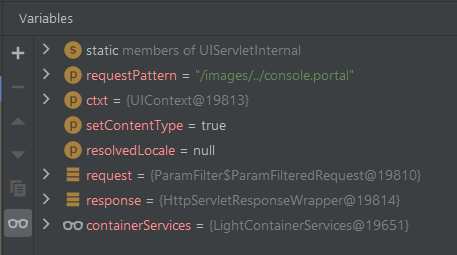
Tới đây thì đã có thể viết PoC hoàn thiện rồi, nhưng mình xin phép phân tích thêm đoạn này,

Sau khi đi qua MBeanUtilsInitSingleFileServlet.service() và thêm 1 số đoạn dài dài nữa, tại UIServletInternal.getTree(), url pattern lại được bóc tách và thực hiện decode URL 1 lần nữa:



Trước khi decode

Và sau khi decode:



Như vậy đã giải thích tại sao việc sử dụng double encoded url có thể vừa bypass được đoạn xử lý normalized url ban đầu, mà vẫn có thể làm cho đoạn xử lý servlet về sau hoạt động đúng!

**#Trigger RCE**

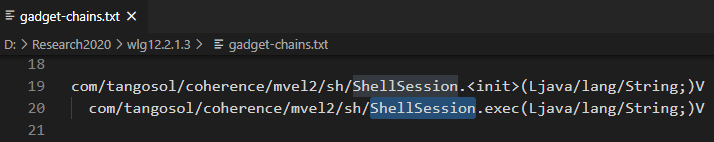
Trong PoC ăn xin được, thanh niên này sử dụng ***com.bea.core.repackaged.springframework.context.support.FileSystemXmlApplicationContext()***để trigger RCE, tuy nhiên chain này yêu cầu phải có outbound.

Mình là người cầu toàn, hơn nữa trong thực tế thì khá nhiều server chặn outbound nên mình quyết định tìm ra 1 chain mới, no outbound required, RCE in one hit! 🤣

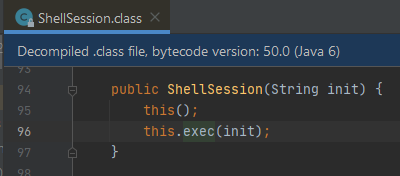
Ngày còn mần liferay, mình đã chỉnh sửa công cụ GadgetInspector rất nhiều, phục vụ cho việc trace code.

Tới lúc này thì nó lại phát huy tác dụng, lôi nó ra và chỉnh sửa một chút ít điều kiện trong code cho phù hợp với context mới, chạy tầm 5p thì ra được khá nhiều kết quả.

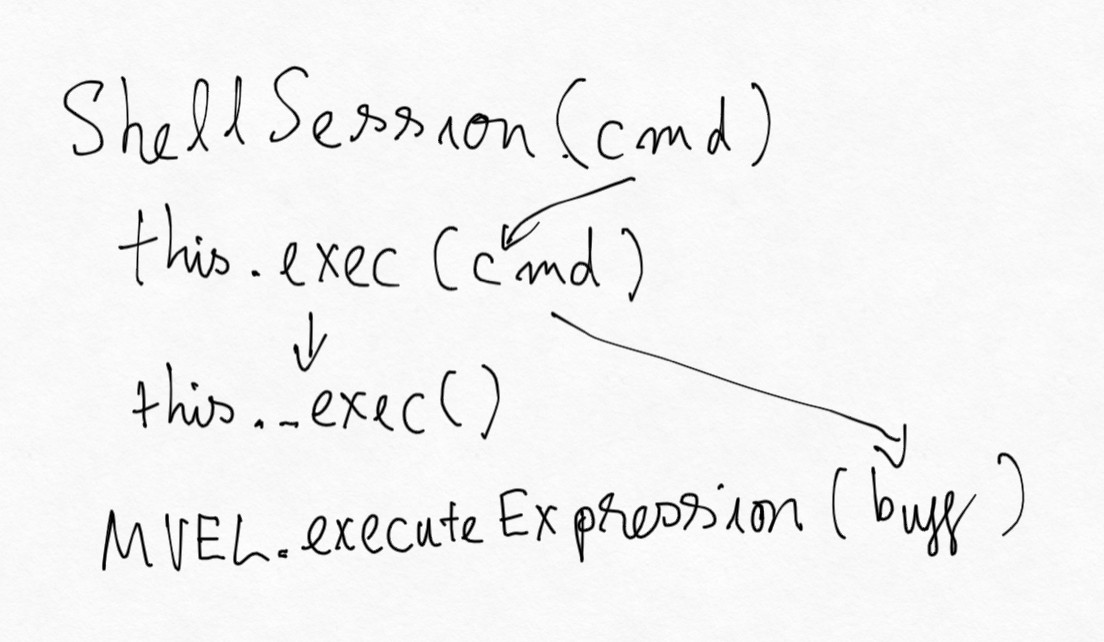
Trong đó có 1 chain khá là tiềm năng:



Chain này sau khi nhận vào arg từ constructor sẽ gọi *ShellSession.exec()* để thực thi với arg này luôn:



Chain phía sau *ShellSession.exec()*còn dài để có thể thực thi được, mình tóm tắt lại như sau:



Như vậy là có thể thực thi MVEL expression tùy ý (về MVEL thì hình như mình cũng đã từng nhắc tới trong các bài trước thì phải).

*Thực sự thì tới tận lúc PoC thành công với chain mới này mình cũng vẫn chưa hết bất ngờ, ko tin đc là có những thứ có sẵn để lợi dụng như vậy. Mình không chắc đây có phải là 1 backdoor nào đó được để lại trong code hay không =))).*

Sau khi kết hợp tất tần tật những thứ trên, thêm chút mắm muối, mình viết hẳn PoC execute command rồi có response luôn =)) cho khỏe. PoC cuối cùng được hoàn thiện bởi Đức chứ không phải mình,

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

=)) Ai cần PoC thì tự nghiên cứu trong này nhé, mình ko có đồ ăn sẵn ở đây.